VON JAVA ZU C

2. Auflage

Carsten VOGT

Zusatzmaterial unter
plus.hanser-fachbuch.de

HANSER

Vogt

Von Java zu C

Ihr Plus – digitale Zusatzinhalte!

Auf unserem Download-Portal finden Sie zu diesem Titel kostenloses Zusatzmaterial.

Geben Sie auf **plus.hanser-fachbuch.de** einfach diesen Code ein:

plus-34avT-Mg81w

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:

www.hanser-fachbuch.de/newsletter

■ Lehrbücher zur Informatik

Begründet von
Prof. Dr. Michael Lutz und *Prof. Dr. Christian Märtin*

weitergeführt von
Prof. Dr. Christian Märtin
Hochschule Augsburg, Fachbereich Informatik

■ Zu dieser Buchreihe

Die Werke dieser Reihe bieten einen gezielten Einstieg in grundlegende oder besonders gefragte Themenbereiche der Informatik und benachbarter Disziplinen. Alle Autoren verfügen über langjährige Erfahrung in Lehre und Forschung zu den jeweils behandelten Themengebieten und gewährleisten Praxisnähe und Aktualität.

Die Bände der Reihe können vorlesungsbegleitend oder zum Selbststudium eingesetzt werden. Sie lassen sich teilweise modular kombinieren. Wegen ihrer Kompaktheit sind sie gut geeignet, bestehende Lehrveranstaltungen zu ergänzen und zu aktualisieren.

Die meisten Werke stellen Ergänzungsmaterialien wie Lernprogramme, Software-Werkzeuge, Online-Kapitel, Beispielaufgaben mit Lösungen und weitere aktuelle Inhalte auf eigenen Websites zur Verfügung.

Titel in dieser Reihe

- Peter Forbrig, Objektorientierte Softwareentwicklung mit UML
- Arne Mayer, Kundenzentrierte Entwicklung von Digitalen/IT-Produkten [in Planung]
- Rainer Oechsle, Parallele und verteilte Anwendungen in Java
- Claudia Reuter, Requirements Engineering – klassisch, agil und hybrid [in Planung]
- Wolfgang Riggert/Ralf Lübben, Rechnernetze
- Rolf Socher, Theoretische Grundlagen der Informatik
- Georg Stark, Robotik mit MATLAB
- Carsten Vogt, Von Java zu C, 2. A.

Carsten Vogt

Von Java zu C

2., aktualisierte Auflage

HANSER

Der Autor:

Prof. Dr. Carsten Vogt, Bergisch Gladbach

Bibliografische Information der Deutschen Nationalbibliothek:

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

Lektorat: Brigitte Bauer-Schiewek
Copy editing: Petra Kienle, Fürstenfeldbruck
Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München
Umschlagrealisation: Thomas West
Titelmotiv: © shutterstock.com/PeachShutterStock; Composing: Thomas West
Schlusslayout: III-satz, Kiel
Druck und Bindung: CPI books GmbH, Leck
Printed in Germany

Print-ISBN: 978-3-446-48103-9

E-Book-ISBN: 978-3-446-48128-2

Inhalt

Vorwort

Dieses Buch gibt eine Einführung in die Programmiersprache C und setzt dabei Kenntnisse in der Sprache Java voraus. Auf den ersten Blick mag das ungewöhnlich erscheinen, ist doch C ein Vorläufer von Java und nicht umgekehrt. Der Ansatz ist dennoch sinnvoll, da in vielen Studiengängen Java als erste Programmiersprache gelehrt wird. In weiterführenden Fächern und der darauf aufbauenden Berufspraxis werden jedoch auch C-Kenntnisse benötigt, beispielsweise zur hardwarenahen Programmierung oder zur Programmierung an der Schnittstelle eines Betriebssystems. C muss also „nachgelernt“ werden.

Das Buch wendet sich daher an Studentinnen, Studenten und andere Interessierte, die bereits Erfahrung mit Java haben und C als weitere Programmiersprache lernen wollen oder müssen. Es ist keine grundständige Darstellung von C, sondern konzentriert sich auf die Besonderheiten der Sprache im Vergleich zu Java. Damit bietet es eine zwar vergleichsweise kurze, aber doch recht detaillierte und tiefgängige Einführung in C. Profitieren wird man auch, wenn man schon einmal mit C in Berührung gekommen ist und nun seine Kenntnisse vertiefen möchte.

Leserinnen und Leser lernen zunächst die grundlegenden Unterschiede in den Sprachansätzen von C und Java, aber auch die vielfältigen Gemeinsamkeiten beider Sprachen kennen. Sie werden dann mit den Besonderheiten von C vertraut gemacht und lernen, die C-spezifischen Konzepte praktisch anzuwenden. Insbesondere werden sie dazu befähigt, sicher mit Zeigern/Pointern (einem fundamentalen Sprachkonstrukt, das es in Java so nicht gibt) umzugehen und dynamische Datenstrukturen, die in Java durch vordefinierte Klassen bereitgestellt werden, in C selbst auszuprogrammieren.

Das Buch kann man auf drei Arten nutzen:

- Wenn man sich rasch einen Überblick über C verschaffen möchte, so sollte man die acht „Schnelleinstiege“ zu Beginn der einzelnen Kapitel lesen. Sie ermöglichen den unmittelbaren Einstieg in die praktische C-Programmierung.
- Wenn man C im Detail kennenlernen möchte, so sollte man die Kapitel des Buchs sukzessive durcharbeiten und die Beispielprogramme praktisch ausprobieren. Man lernt dabei nicht nur die sprachlichen Möglichkeiten von C, sondern auch typische Programmiertricks und -fallen kennen.
- Wenn man bei der späteren praktischen Arbeit bestimmte Details nachschlagen möchte, so sollte man dazu die Anhänge benutzen. Insbesondere findet man ganz am Ende des Buchs eine tabellarische Darstellung von Informationen, die man bei der C-Programmierung häufig benötigt.

Viele Beispiele und Grafiken verdeutlichen den Stoff und Verweise innerhalb des Buchs zeigen Zusammenhänge zwischen den Teilbereichen auf. Tricks, Fallen und Informationen für Fortgeschrittene sind typografisch hervorgehoben. Übungsaufgaben dienen zur Überprüfung des Lernerfolgs.

Die erste Auflage des Buchs erschien unter dem Titel „C für Java-Programmierer“. Diese zweite Auflage mit dem Titel „Von Java zu C“ wurde bezüglich einiger weniger technischer Details aktualisiert. Die Änderungen halten sich aber in engen Grenzen, da C eine sehr stabile Programmiersprache ist. Zudem wurden die Quellenhinweise und die Empfehlungen zu Programmierwerkzeugen aufgefrischt sowie Fehler korrigiert. Schließlich wurde der Text im Hinblick auf eine geschlechtergerechte Sprache überarbeitet, was auch der Grund für die Änderung des Buchtitels war. Sterne * treten aber nach wie vor nur als Operatoren der Programmiersprache C auf.

Köln/Bergisch Gladbach, im Sommer 2024 Carsten Vogt

Zusatzmaterial zum Buch

Zu diesem Buch stehen Ihnen weitere Inhalte digital zur Verfügung:

- die Beispielprogramme,
- die Lösungen der Übungsaufgaben,
- die nach Drucklegung entdeckten Fehler

Gehen Sie dazu einfach auf

`https://plus.hanser-fachbuch.de`

und geben Sie dort diesen Code ein:

`plus-34avT-Mg81w`

Hinweise auf Dokumentationen und Werkzeuge, die im Internet frei verfügbar sind, gibt der Literaturteil auf Seite 223.

Schnelleinstieg 1: C und Java im Vergleich

C ist wie Java eine *imperative* Programmiersprache: Ein C-Programm besteht aus Anweisungen, wie arithmetischen Operationen und Wertzuweisungen. Die Anweisungen werden, gesteuert durch Kontrollstrukturen, wie Verzweigungen und Schleifen, in einer bestimmten Reihenfolge ausgeführt. C und Java haben viele gemeinsame Konstrukte, denn das objektorientierte C++, das aus C hervorgegangen ist, hat die Entwicklung von Java beeinflusst.

C wird insbesondere *hardwarenah* eingesetzt, beispielsweise zur Programmierung von Mikrocontrollern sowie zur Implementierung und Nutzung von Betriebssystemen. Es kann aber auch wie Java zur problemorientierten *Anwendungsprogrammierung* benutzt werden. Im Laufe der Jahre wurde C mehrfach standardisiert und es stehen C-Compiler und -Programmierumgebungen für verschiedene Betriebssystemplattformen zur Verfügung.

C ist eine *prozedurale* Sprache: Ein C-Programm ist meist eine Sammlung von sogenannten Prozeduren oder Funktionen, die Teilprobleme lösen und sich gegenseitig aufrufen. Eine dieser Funktionen ist das Hauptprogramm `main()`:

```
#include <stdio.h> /* Import I/O-Funktionen printf/scanf */
/* Funktion zur Ermittlung des Maximums zweier float-Werte */
float max(float a, float b) {
  if (a>b) return a;
   else return b;
}
/* Hauptprogramm */
int main(void) {
  float x, y;
  printf("Bitte zwei Zahlen eingeben: ");
  scanf("%f %f",&x,&y);      /* Einlesen zweier float-Werte */
  printf("Maximum: %f\n",max(x,y)); /* Ausgabe des Maximums */
  return 0;
}
```

C ist *keine objektorientierte Sprache*, kennt also keine Klassen und Objekte. C besitzt aber ein differenziertes *Datentypkonzept*: Es stellt skalare Typen, wie ganze Zahlen, Gleitkommazahlen und Zeichen, und zusammengesetzte Typen, wie Arrays oder „structs“ (d.h. Strukturen aus mehreren Komponenten unterschiedlicher Typen), zur Verfügung. Zudem realisiert C sogenannte *Zeigervariablen*, die Speicheradressen enthalten und damit den unmittelbaren Zugriff auf Hauptspeicherzellen ermöglichen.

C-Programme werden nicht interpretiert, sondern *übersetzt*: Der C-Quellcode wird über Zwischenstufen in ein Maschinenprogramm übersetzt (beispielsweise eine `exe`-Datei), das unmittelbar durch die Prozessorhardware ausgeführt werden kann.

Der nächste Schnelleinstieg steht auf Seite 16.

1 Einführung

C und Java gehören zu den meistverbreiteten Programmiersprachen und sind damit grundlegendes Rüstzeug für die Programmierung von Computern. Bei der Entwicklung von Java hat C, die ältere der beiden Sprachen, unmittelbar und mittelbar Pate gestanden. Man findet daher viele Eigenschaften von C in Java wieder – und somit auch umgekehrt, so dass Java-Kenntnisse eine gute Basis für den Einstieg in C sind.

Diese Einführung gibt zunächst einen kurzen Überblick über Geschichte, Eigenschaften und Anwendungsgebiete von C und Java. Anschließend werden, ausgehend von drei einfachen Programmbeispielen, die wichtigsten Gemeinsamkeiten und Unterschiede der beiden Sprachen aufgezeigt. Zum Abschluss findet man einen Ausblick auf die folgenden Kapitel dieses Buchs sowie eine Anleitung zur schnellen Lektüre, zum Nachschlagen und zur Vertiefung des Stoffs.

1.1 C und Java von den Anfängen bis heute

1.1.1 Die Entwicklung von C

1.1.1.1 Der Ursprung

Die Anfänge der Programmiersprache C liegen ein halbes Jahrhundert zurück: Als **Dennis Ritchie** und **Ken Thompson** um das Jahr 1970 das Betriebssystem **UNIX** entwickelten, benötigten sie eine Programmiersprache zu seiner Implementierung. Um UNIX portabel, also zwischen verschiedenen Rechnerplattformen übertragbar zu machen, musste diese Sprache **maschinenunabhängig** sein, das heißt unabhängig von der Maschinensprache eines bestimmten Prozessors. Da ein Betriebssystem unmittelbar auf die Rechnerhardware aufsetzt, musste die Sprache aber **maschinennah** sein, also einen leichten Zugriff auf die Komponenten der Hardware (insbesondere Speicherzellen und Prozessorregister) erlauben und sich effizient in Maschinensprache übersetzen lassen.

Keine der Programmiersprachen, die damals zur Verfügung standen, erschien Ritchie und Thompson für diesen Zweck geeignet. Sie erarbeiteten daher zwei neue Sprachen: Ausgehend von der vorhandenen Sprache **BCPL** (= Basic Combined Programming Language) entwickelte zunächst Thompson die Sprache **B**, eine hardwarenahe Sprache mit Unterprogrammen, aber ohne differenzierte Datentypen. Ritchie wiederum entwickelte B zur Sprache **C** weiter (als Namen benutzte er dabei einfach den nächsten Buchstaben im Alphabet) und führte dabei unter anderem verschiedene Datentypen ein.

1.1.1.2 Grundlegende Eigenschaften

C ist eine **imperative, prozedurale Sprache**: „Imperativ“ bedeutet, dass sich ein C-Programm aus Anweisungen, wie arithmetischen Operationen und Wertzuweisungen, zusam-

mensetzt und dass diese Anweisungen in einer bestimmten Reihenfolge, gesteuert von Kontrollstrukturen, wie Verzweigungen und Schleifen, ausgeführt werden. „Prozedural“ bedeutet, dass Teilprogramme definiert werden können – also sogenannte „Prozeduren“ oder „Funktionen“, die (ähnlich den statischen Methoden von Java) Teilprobleme lösen und beliebig oft aufgerufen werden können.

C besitzt ein **differenziertes Datentypkonzept**: Es stellt skalare Typen, wie ganze Zahlen, Gleitkommazahlen und Zeichen, sowie zusammengesetzte Typen, wie Arrays oder Strukturen, zur Verfügung. Zudem realisiert C sogenannte Zeiger-/Pointervariablen, die Speicheradressen enthalten. Über diese Adressen kann man unmittelbar auf Speicherzellen zugreifen und darauf arithmetische und bitweise Operationen ausführen. Insbesondere aufgrund dieser Vielfalt von Datentypen kann man C nicht nur zur systemnahen Programmierung einsetzen, sondern auch problemnahe Anwendungen in C schreiben.

C ist eine **kompilierte Sprache**: C-Programme werden zur Ausführung in die Maschinensprache der jeweiligen Rechnerplattform übersetzt. Das resultierende Maschinenprogramm ist nicht frei portabel, kann also nicht von Prozessoren mit einem anderen Maschinenbefehlssatz ausgeführt werden.

1.1.1.3 Standards

C verbreitete sich zusammen mit UNIX rasch, gewann dann auch unabhängig von UNIX schnell an Bedeutung und wurde im Laufe der Jahre weiterentwickelt und standardisiert:

- 1978 veröffentlichten **Brian Kernighan** und **Dennis Ritchie** das Buch „The C Programming Language“ [KeRi78] und legten damit den ersten allgemeinen C-Standard fest. Die dort beschriebene C-Version wird als **Kernighan-Ritchie-C** (kurz **K&R C**) bezeichnet.
- 1989 wurde C in einer revidierten und erweiterten Fassung durch das **ANSI** (American National Standards Institute) standardisiert [ANSI89]. Dies war das erste **„ANSI-C“**, später auch **C89** genannt.
- 1990 übernahm die **ISO** (International Organization for Standardization, [ISO]) den ANSI-Standard [ISO90] und später dann auch das **DIN** (Deutsches Institut für Normung) [DIN94]. Das dort festgelegte C, das nur in wenigen Details von ANSI-C abweicht, wird **C90** genannt.
- Seither folgte eine Reihe weiterer ISO-Standards, nämlich **C95**, **C99**, **C11** und **C17** mit Änderungen und Erweiterungen, die aber jeweils die Kernideen und -konzepte der Sprache unverändert ließen.
- Zum Zeitpunkt der Drucklegung dieses Buchs ist **C17** der aktuelle Standard [ISO18] und damit auch Grundlage dieser Einführung in C. Die Nachfolgeversion **C23** ist in der finalen Abstimmung [ISO23]. Sie lässt keine Änderungen erwarten, die für die Darstellungen hier wesentlich wären.

1.1.2 Objektorientierte Nachfolgesprachen

1.1.2.1 C++

Vor und während der Entstehung von C kamen objektorientierte Programmiersprachen auf – zunächst in den 1960er-Jahren die Sprache **Simula**, danach in den 1970ern die Sprache **Smalltalk**. Anfang der 1980er-Jahre übertrug **Bjarne Stroustrup** die Ideen von Simula auf C und erweiterte es damit um Konzepte der objektorientierten Programmierung [Stro86]. Er nannte die resultierende Sprache **C++** und spielte dabei auf den ++-Operator an, mit dem der Wert einer Variablen um eins erhöht wird.

C++ ist keine rein objektorientierte, sondern eine hybride Sprache, das heißt eine Mischform: Einerseits kann man in C++ ganz ohne Klassen und Objekte programmieren, also wie in C ein Programm als Sammlung von Funktionen aufbauen, zwischen denen Parameter und Rückgabewerte ausgetauscht werden. Andererseits realisiert C++ die Konzepte der objektorientierten Programmierung, so insbesondere Objekte und Klassen, Vererbung und abstrakte Klassen. Dazu kommen weitere Merkmale wie statische Methoden und das Überladen und Überschreiben von Methoden.

1.1.2.2 Java

Die Programmiersprache **Java**, die es seit Anfang der 1990er-Jahre gibt, geht auf ein Team der Firma **Sun Microsystems** um **James Gosling** zurück. Java ist im Gegensatz zu C++ eine rein objektorientierte Sprache (wobei man natürlich auch in Java prozedural programmieren kann, indem man Hauptprogramm und weitere Prozeduren/Funktionen als statische Methoden einer einzigen Klasse definiert und auf die Erzeugung von Objekten verzichtet).

Java übernahm viele Syntaxeigenschaften und die darunter liegende Semantik von mehreren anderen Programmiersprachen, so insbesondere von C und C++. Die grundlegenden Kontrollstrukturen von C (Blöcke, bedingte Anweisungen, Schleifen), die Mehrzahl der skalaren Datentypen mit ihren Operationen sowie die Definition von Funktionen/Methoden mit Parametern und Wertrückgaben findet man in Java (bis auf geringe Abweichungen) identisch wieder. Auch gibt es eine Reihe von Übereinstimmungen zwischen C++ und Java hinsichtlich des Klassenkonzepts. Hier bestehen aber auch deutliche Unterschiede, beispielsweise in der Syntax einer Klassendefinition und hinsichtlich der Mehrfachvererbung (also des Erbens von mehreren Basisklassen), die in C++ erlaubt ist, in Java jedoch nicht.

Java verzichtet auf ein Zeigerkonzept, wie es C bietet, und nimmt zudem deutlich strengere Typprüfungen vor. Dies geschieht aus Sicherheitsgründen: Ein unkontrollierter Zugriff auf Variablen oder Speicherzellen und die beliebige Manipulation der dort gespeicherten Bitmuster sollen ausgeschlossen sein.

Zudem werden Java-Programme nicht wie C-Programme direkt in Maschinensprache übersetzt, sondern in Bytecode (also in eine Zwischensprache), der dann durch die Java Virtual Machine auf der jeweiligen Plattform zur Ausführung gebracht wird. Im Gegensatz zu übersetzten C-Programmen sind übersetzte Java-Programme damit portabel, können also auf unterschiedlichen Rechnerplattformen ausgeführt werden.

1.1.3 Einsatzgebiete von C und Java

C wird insbesondere **hardwarenah** eingesetzt: Man benutzt es zur Programmierung von Mikrocontrollern in eingebetteten Systemen, mit denen Geräte gesteuert werden, sowie zur Implementierung von Betriebssystemen wie UNIX/Linux und zugehörigen Dienstprogrammen. UNIX/Linux, Windows und andere Betriebssysteme bieten entsprechende Programmierschnittstellen (Application Programming Interfaces, APIs) an, also Sammlungen von C- und C++-Funktionen, über die Dienste des Betriebssystems aufgerufen werden können. Auf diese Basis werden dann, ebenfalls in C und/oder C++, Anwendungsprogramme realisiert.

Java als objektorientierte Sprache hat ein höheres Abstraktionsniveau als C und verfügt zudem über eine Vielzahl „mitgelieferter" Klassen und Pakete. Es ist damit zur Programmierung komplexer lokaler und verteilter Anwendungen geeignet, so insbesondere auch von Webservern sowie von Apps für Mobilgeräte.

In der Hochschulausbildung dienen C und Java häufig als „Einstiegssprachen", werden also in einer Grundvorlesung zur Einführung in die Programmierung eingesetzt. Dabei wird aber, oft aus Zeitgründen, vielfach nur eine dieser Sprachen gelehrt. Im weiteren Verlauf des Studiums benötigt man jedoch beide Sprachen. C wird dann eher in den hardware- und systemnnahen Fächern, wie Technische Informatik und Betriebssysteme, aber auch Datenbanken benutzt. Java findet man eher in anwendungsnahen Bereichen, wie Softwaretechnik, der Realisierung nebenläufiger und verteilter Anwendungen sowie der Programmierung von Mobilgeräten.

1.2 C und Java im Sprachvergleich

1.2.1 Drei Beispielprogramme

Zum Einstieg in die praktische C-Programmierung sollen drei einfache C-Programmbeispiele diskutiert und mit ihren „Gegenstücken" in Java verglichen werden. Dabei zeigen sich schon einige Unterschiede zwischen C und Java; es wird aber auch deutlich, dass die beiden Programmiersprachen viele Gemeinsamkeiten haben.

1.2.1.1 Einfaches Programm mit Ausgabe

Ein Programm, das die Zweierpotenzen von 1 bis 256 ausgibt, könnte in C wie folgt aussehen:

C ...

```
#include <stdio.h>
int main(void) {
 int zahl = 1;
 // Schleife bis zum Erreichen der Obergrenze
 while ( zahl <= 256 ) {
  printf("Zweierpotenz: %d\n",zahl);
  zahl = zahl * 2;
```

```
 }
 return 0;
}
```

Das entsprechende Java-Programm könnte wie folgt lauten:

... Java

```
import java.io.*;
class ZweierPotenzen_1 {
 public static void main(String args[]) {
  int zahl = 1;
  // Schleife bis zum Erreichen der Obergrenze
  while ( zahl <= 256 ) {
   System.out.println("Zweierpotenz: " + zahl);
   zahl = zahl * 2;
  }
 }
}
```

Die Programme sind sich offensichtlich sehr ähnlich; die Syntax vieler Programmbestandteile ist sogar identisch. Unterschiede bestehen lediglich an den folgenden Stellen:

- Während in Java externe Komponenten per `import` importiert werden, geschieht dies in C durch `#include`. Details zur `#include`-Anweisung und ähnlichen **Präprozessor-Instruktionen** werden in → 2.3 diskutiert.
- In Java muss das **Hauptprogramm `main()`** in einer eigenen Klasse definiert werden, in C kommt es ohne einen solchen Rahmen aus. Auch ist der `main()`-Kopf in C deutlich einfacher; so kann man beispielsweise auf Parameter verzichten, und Ausnahmen, die geworfen werden könnten, gibt es in C generell nicht. Dafür hat das C-`main()` den Rückgabetyp `int` statt `void`: Mit dem Rückgabewert 0 („`return 0`“) wird der Aufrufumgebung ein fehlerfreies Programmende angezeigt; ein Rückgabewert ungleich 0 meldet einen Fehler. Der Aufbau eines C-Programms wird in → 2.1 genauer beschrieben.
- Die Java-Ausgabemethode `println()` erhält als einzigen Parameter eine Zeichenkette, die sich im Allgemeinen durch den +-Operator aus konstanten Zeichenketten und Variablenwerten zusammensetzt und entsprechend auf den Bildschirm ausgegeben wird. Die **C-Ausgabefunktion `printf()`** kann dagegen mehrere Parameter besitzen: Der erste Parameter ist stets eine Zeichenkettenkonstante, die Format- oder Konversionsangaben (eingeleitet mit `%`) enthalten kann. Diese Konversionsangaben sind „Platzhalter“, für die bei der Ausgabe die Werte der nachfolgenden Parameter eingesetzt werden. Der Buchstabe in einer Konversionsangabe charakterisiert den Typ des auszugebenden Werts, so z.B. `%d` für dezimale `int`-Werte. Nähere Informationen zu `printf()` findet man in → 7.2.1. Ein entsprechende Java-Methode findet man übrigens in der Klasse `PrintStream`.

1.2.1.2 Programm mit Eingabe und C-spezifischen Datentypen

Das Programm wird nun so erweitert, dass man die Obergrenze eingeben kann, bis zu der die Potenzen berechnet werden sollen. Zudem wird der Wertebereich der Ganzzahlvariablen auf nichtnegative Zahlen beschränkt. Die C-Version lautet jetzt wie folgt:

C ...

```
#include <stdio.h>
int main(void) {
 unsigned int obergrenze;
 unsigned int zahl = 1;
 printf("Eingabe der Obergrenze: ");
 scanf("%u",&obergrenze);
 while ( zahl <= obergrenze ) {
  printf("Zweierpotenz: %u\n",zahl);
  zahl = zahl * 2;
 }
 return 0;
}
```

Das erweiterte Java-Programm sieht so aus:

... Java

```
import java.io.*;
class ZweierPotenzen_2 {
 public static void main(String args[]) throws IOException {
  int zahl = 1;
  int obergrenze;
  BufferedReader in =
   new BufferedReader(new InputStreamReader(System.in));
  System.out.print("Eingabe der Obergrenze: ");
  obergrenze = Integer.parseInt(in.readLine());
  while ( zahl <= obergrenze ) {
   System.out.println("Zweierpotenz: " + zahl);
   zahl = zahl * 2;
  }
 }
}
```

Hier erkennt man zwei weitere Unterschiede:

- C verfügt, im Gegensatz zu Java, über einen Datentyp **`unsigned int`**, dessen Wertebereich die nichtnegativen ganzen Zahlen (bis zu einem bestimmten Maximalwert) sind. Mit den Standardtypen von C beschäftigt sich → 4.1.
- Ein Java-Programm liest Werte üblicherweise mit Hilfe eines `BufferedReader`- oder eines `Scanner`-Objekts ein. In C dient dazu die **Eingabefunktion `scanf()`**: Ähnlich wie `printf()` verwendet diese Funktion im ersten Parameter Konversionsangaben, die Anzahl und Typen der einzugebenden Werte bestimmen. Weitere Parameter geben die Variablen an, in denen die Werte gespeichert werden sollen. `scanf()` wird in → 7.2.2 näher beschrieben.

Wichtig ist hier zudem der `&`-Operator, der im Beispiel dem zweiten `scanf()`-Parameter vorangestellt ist. Dieser **Adressoperator** liefert die Speicheradresse der Variablen und gibt somit die Stelle im Speicher an, an der der eingegebene Wert abgelegt werden soll. C ermöglicht einen sehr flexiblen Umgang mit solchen Adressen und erlaubt unter anderem, sie in **Zeigervariablen** (**Pointern**) zu speichern. Die → Kapitel 5 und 8 befassen sich ausführlich mit dem Zeigerkonzept von C und seiner Anwendung.

1.2.1.3 Programm mit einer Funktion

Schließlich wird noch eine einfache **Funktion** eingeführt, die das Doppelte eines Werts berechnet. Das C-Programm hat nun die folgende Form:

C ...

```
#include <stdio.h>
int doppelt(int x) {
 return 2*x;
}
int main(void) {
 unsigned int obergrenze;
 unsigned int zahl = 1;
 printf("Eingabe der Obergrenze: ");
 scanf("%u",&obergrenze);
 while ( zahl <= obergrenze ) {
  printf("Zweierpotenz: %u\n",zahl);
  zahl = doppelt(zahl);
 }
 return 0;
}
```

In Java lautet das Programm jetzt:

... Java

```
import java.io.*;
class ZweierPotenzen_2 {
 public static int doppelt(int x) {
  return 2*x;
 }
 public static void main(String args[]) throws IOException {
  int zahl = 1;
  int obergrenze;
  BufferedReader in =
   new BufferedReader(new InputStreamReader(System.in));
  System.out.println("Eingabe der Obergrenze: ");
  obergrenze = Integer.parseInt(in.readLine());
  while ( zahl <= obergrenze ) {
   System.out.println("Zweierpotenz: " + zahl);
   zahl = doppelt(zahl);
  }
 }
}
```

C-Funktionen entsprechen also den statischen (d.h. nicht auf ein bestimmtes Objekt bezogenen) Methoden von Java. Sie sind in C aber nicht einer bestimmten Klasse zugeordnet (da es ja Klassen in C überhaupt nicht gibt), sondern werden in der Programmdatei (oder auch in mehreren Dateien) ohne einen umschließenden Rahmen hintereinander definiert.

1.2.2 Eigenschaften von Java vs. Eigenschaften von C

1.2.2.1 Tabellarischer Vergleich

Die drei Programmbeispiele zeigten einige, aber bei weitem nicht alle Unterschiede zwischen C und Java auf. Tabelle 1.1 gibt eine knappe Gesamtübersicht über die Gemeinsamkeiten und Unterschiede von Java und C. Zudem verweist sie auf die Kapitel des Buchs, in denen die einzelnen Themenbereiche behandelt werden.

Tabelle 1.1 Vergleich zwischen Java und C

	Java	**C**	**siehe**
Art der Sprache	imperativ, objektorientiert	imperativ, prozedurorientiert	1.2.2.2
Programmaufbau	Konstrukt aus Klassen, Schachtelung und Verteilung auf mehrere Dateien/Pakete möglich	Sammlung von Funktionen, Schachtelung nicht möglich, Verteilung auf mehrere Dateien möglich	2.1
Ausführung von Programmen	Übersetzung in Bytecode, Ausführung des Bytecodes durch die Java Virtual Machine (durch Interpretation oder mit Just-in-time Compilation)	Vorarbeit durch Präprozessor, Übersetzung in Maschinensprache des Prozessors, Ausführung durch die Prozessor-Hardware	1.2.2.3, 2.2, 2.3
Kontrollstrukturen	nahezu identische Blöcke, bedingte Anweisungen und Schleifen; nur wenige Detailunterschiede		3
skalare Datentypen	ähnliche Zahlen- und Zeichentypen mit Unterschieden in den Wertebereichen		4.1
zusammengesetzte Datentypen	Objekte, darunter auch Arrays und Collections mit zahlreichen vordefinierten Operationen	Arrays, Strukturen, Unions, Bitfelder, keine vordefinierten Typen oder Operationen für Mengen, Listen usw.	4.3, 4.4, 4.5, 8
Definition weiterer Typen	Klassen, Enumerationen	`typedef`-Operator, `struct`-Typen, Enumerationen	4.4.2, 4.6
Zeiger/Pointer	Objektvariablen mit Referenzen auf Objekte	Zeigervariablen mit Adressen von Speicherzellen und Adressarithmetik	5
Prozeduren/ Funktionen	klassenbezogene und objektbezogene Methoden	Sammlung gleichberechtigter Funktionen	6
Ein-/Ausgabe- u. Dateifunktionen	definiert durch Java-Standardklassen	definiert durch die C-Standardbibliothek	7
Ausnahmebehandlung	mit `class Exception` und `try`/`catch`/`throw`	u.a. mit Sprungbefehl `goto`	3.4

Zwei grundlegende Unterschiede zwischen C und Java sind also,

- dass es sich bei C um eine **prozedurorientierte Sprache** handelt, während Java eine **objektorientierte Sprache** ist, und
- dass C-Programme in **Maschinensprache** übersetzt werden, während Java-Programme in eine **Zwischensprache** übersetzt werden, die dann durch die **Java Virtual Machine** interpretiert oder vor der Ausführung weiter in Maschinensprache übersetzt wird.

Im Folgenden werden diese beiden Unterschiede etwas ausführlicher diskutiert; mit den übrigen Eigenschaften beschäftigen sich die nachfolgenden Kapitel dieses Buchs.

1.2.2.2 Objektorientierung vs. Prozedurorientierung

Java und C sind beides **imperative Sprachen**: Ihre grundlegenden Anweisungen definieren relativ einfache Operationen und Kontrollstrukturen legen fest, in welcher Reihenfolge die Anweisungen ausgeführt werden. In diesem gemeinsamen Rahmen sind Java und C jedoch unterschiedlich ausgerichtet – Java ist objektorientiert, C dagegen prozedurorientiert:

- In einer **objektorientierten Sprache** stehen die Datenstrukturen im Vordergrund, mit denen gearbeitet wird. Eine Datenstruktur wird durch ein Objekt realisiert, das ihre Werte speichert und die Operationen definiert, die hierauf angewendet werden können. Ein Beispiel sind Objekte einer Klasse `Bruch` zur Darstellung rationaler Zahlen. Ein `Bruch`-Objekt kann mit Hilfe seiner Methode `kuerzen()` gekürzt werden, wobei seine Hilfsmethode `ggt()` den größten gemeinsamen Teiler von Zähler und Nenner berechnet:

Java ...

```
class Bruch {
 private int zaehler, nenner;
 ... Konstruktor ...
 private int ggt(int x, int y) {
  while (x!=y)
   if (x<y)
     y=y-x;
    else x=x-y;
  return x;
 }
 ... set- und get-Methoden ...
 public void kuerzen() {
  int teiler=ggt(zaehler,nenner);
  zaehler=zaehler/teiler;
  nenner=nenner/teiler;
 }
}
class Hauptprogramm {
 public static void main(String args[]) {
  Bruch b = new Bruch(...);
  b.kuerzen();
 }
}
```