IN DIESEM KAPITEL

- Einen Zahlenwert abspeichern
- Eine Bedingung abfragen
- Wahrheitswerte als Datentyp

Kapitel 1
Programmieren in Java

Programmieren ist schwer. Wenn's einfach wär', dann könnt's ja jeder ... Aber Programmieren ist ein Handwerk, das Sie lernen können. Schritt für Schritt, eine Sache nach der anderen. Und damit soll es nun losgehen.

Sie fangen quasi bei null an – wenn Sie jedoch schon ein wenig Programmiererfahrung haben, dann lesen Sie die ersten Kapitel etwas schneller. Überprüfen Sie dabei aber immer, ob sich das Gelesene mit Ihren Vorstellungen deckt.

Die Programmiersprache ist Java. Aber Sie können das meiste auch auf andere Programmiersprachen übertragen. Im Vordergrund steht nicht, die Syntax einer bestimmten Programmiersprache zu lernen, sondern programmieren zu lernen. Es gibt zahlreiche syntaktische Einzelheiten, in denen sich Programmiersprachen voneinander unterscheiden (zum Glück hat sich als Zeichen für die Addition von Zahlen das +-Zeichen weitestgehend durchgesetzt ;-)) Für das Programmieren-Können sind aber diese Unterschiede nicht entscheidend.

Wenn Sie das, was Sie programmiert haben, ausprobieren wollen, brauchen Sie eine sogenannte Programmierumgebung auf Ihrem Computer, also eine Software, mit der Sie bequem Ihr Programm schreiben, testen und ausführen können. Eine Anleitung dazu erhalten Sie am Ende dieses Kapitels bei Ihrer ersten Übungsaufgabe.

Wertzuweisung

Während einer umfangreicheren Berechnung muss der Computer viele Zwischenergebnisse »im Kopf behalten«. Er speichert diese Zwischenergebnisse in seinem Arbeitsspeicher ab. Wenn Sie das entsprechende Programm schreiben, geben Sie jedem dieser Zwischenergebnisse einen Namen. Dann findet der Computer jederzeit das richtige Zwischenergebnis wieder.

Technisch gesprochen verwenden Sie *Variablen*, denen Sie die jeweiligen Zwischenergebnisse als Werte zuweisen. Sie schreiben also beispielsweise `a = b + 1;`. Damit verwenden Sie das Zwischenergebnis, das Sie vorher der Variablen *b* zugewiesen haben, erhöhen dessen Wert um 1 und weisen das Ergebnis der Variablen *a* zu.

Haben Sie es gemerkt? *Sie* verwenden ein Zwischenergebnis, *Sie* erhöhen dessen Wert um 1, *Sie* weisen das Ergebnis einer anderen Variablen zu. Tatsächlich aber machen nicht Sie das, sondern der Computer tut es. Aber während Sie programmieren, begeben Sie sich immer in die Rolle des Computers und vollziehen nach, was der Effekt Ihrer Programmanweisungen ist. Auf diese Weise erzielen Sie zum Schluss ein funktionierendes Programm.

Variablen deklarieren

Als Erstes *deklarieren* Sie die Variablen, die Sie verwenden möchten. Für jede Variable geben Sie an, wie sie heißt und welchen *Datentyp* (oder kurz: *Typ*) die Werte haben, die Sie der Variablen später zuweisen möchten. Eine solche Deklaration sieht beispielsweise so aus:

```
int a, b;
```

Damit geben Sie an, dass es zwei Variablen gibt, dass diese *a* und *b* heißen und dass sie später Werte des Datentyps `int` aufnehmen sollen, also ganzzahlige Zahlenwerte (engl.: *integer* – ganzzahlig). In der Programmiersprache Java sind die Variablen jeweils auf einen bestimmten Datentyp festgelegt. Daher kann man auch sagen, dass die *Variablen a* und *b* vom Datentyp `int` sind, obwohl tatsächlich die *Werte*, die sie später aufnehmen, vom Datentyp `int` sind.

Wozu Datentypen?

In Java werden ganze Zahlen standardmäßig als 32-Bit-Wörter dargestellt und entsprechend abgespeichert (Abbildung 1.1). Ein solches Wort besteht aus 32 Bit, also Nullen und Einsen, mit denen der Zahlenwert im Binärsystem dargestellt ist – alles Genauere dazu erfahren Sie in Kapitel 21.

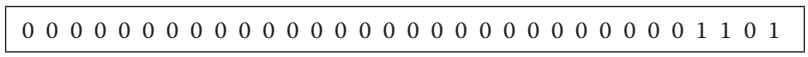

Abbildung 1.1: 32-Bit-Binärzahl (hier: Zahlenwert 13)

Für einen Wert des Datentyps `int` benötigt der Computer also 32 Bit Speicherplatz. Mit der Deklaration einer Variablen *a* vom Datentyp `int` weisen Sie den Computer an, 32 Bit Speicherplatz an irgendeiner bestimmten Stelle im Speicher zu reservieren und mit dem symbolischen Adresse *a* zu versehen. Der Computer führt intern Buch darüber, unter welchen symbolischen Adressen er Speicherplatz reserviert hat. Und er merkt sich, wie die Daten dort zu interpretieren sind. Denn letztlich sind dort ja nur irgendwelche Nullen und Einsen abgespeichert, und daher ist es wichtig, dass der Computer sich merkt, dass diese Nullen und Einsen beispielsweise eine `int`-Zahl darstellen.

Auf diese Weise kann der Computer jederzeit, wenn der Variablen *a* ein Wert zugewiesen wird, auf den entsprechenden Speicherplatz zugreifen und den Wert dort abspeichern. Oder

umgekehrt, wenn der Wert der Variablen *a* gebraucht wird, kann er an dem entsprechenden Speicherplatz nachschauen und den Wert dort ablesen.

Andere Datentypen, so etwa Kommazahlen oder Zeichenketten, aber auch Datenlisten oder andere Datenstrukturen, erfordern mehr Speicherplatz und werden auf andere Art und Weise abgespeichert.

Es gibt noch einen anderen Grund dafür, dass Datentypen verwendet werden, denn nach dem Datentyp richten sich die möglichen Rechenoperationen, die jeweils anwendbar sind. So können Sie beispielsweise zwei Zahlen addieren. Dagegen können Sie zwei Zeichenketten nicht addieren, aber Sie können sie miteinander verketten, also quasi hintereinanderschreiben.

Und für die strukturierten Datentypen, die Sie in der objektorientierten Programmierung verwenden, können Sie jeweils ganz bestimmte Operationen, die dort als Methoden bezeichnet werden, anwenden.

Ein Datentyp kennzeichnet also immer die Art der Daten sowie die darauf anwendbaren Operationen.

Die »Art der Daten« ist etwas schwammig ausgedrückt – genau genommen besteht ein Datentyp aus der Menge der möglichen Werte der Daten. Der Datentyp `int` beispielsweise besteht aus der Menge der ganzen Zahlen $\{-2.147.483.648, \ldots, +2.147.483.647\}$. Aufgrund der 32-Bit-Binärdarstellung sind dies die Zahlen $-2^{31}, \ldots, +2^{31} - 1$.

Etwas weiter unten werden Sie einen Datentyp kennenlernen, der aus nur zwei Werten besteht.

Einen Wert zuweisen

Nachdem Sie eine Variable deklariert haben, können Sie ihr einen Wert zuweisen. So schreiben Sie beispielsweise

```
a=17;
```

Sie haben ja die Variable *a* weiter oben als Variable vom Datentyp `int` deklariert. Daher können Sie dieser Variablen nun einen Wert vom Datentyp `int`, also eine ganze Zahl, zuweisen. Eine derartige Anweisung heißt Wertzuweisung.

Sie lesen diese Wertzuweisung am besten als »a ergibt sich zu 17«. Denn das Gleichheitszeichen in der Wertzuweisung ist kein Gleichheitszeichen, sondern das Wertzuweisungszeichen. Wenn Sie etwas weniger umständlich »a gleich 17« sagen, dann seien Sie sich bitte bewusst, dass dies nicht korrekt ist – im übernächsten Beispiel werden Sie einen Fall kennenlernen, wo dies offenbar wird.

Die Wertzuweisung

```
b=a+4;
```

bewirkt Folgendes:

- ✔ Als Erstes wertet der Computer den Ausdruck auf der rechten Seite des Wertzuweisungszeichens aus. Dazu schaut er im Speicher unter der symbolischen Adresse *a* nach, welcher Wert dort gespeichert ist, dies ist die ganze Zahl 17. Dann addiert er die ganze Zahl 4 dazu und erhält als Ergebnis die ganze Zahl 21.
- ✔ Dann prüft der Computer, ob der Datentyp dieses Ergebnisses mit dem Datentyp der Variablen, die links des Wertzuweisungszeichens steht, übereinstimmt. Dies ist hier der Fall, denn Sie haben weiter oben die Variable *b* als vom Datentyp `int` deklariert.
- ✔ Dann weist der Computer der Variablen *b* den Wert 21 zu, d. h., er speichert diesen Wert unter der symbolischen Adresse *b* ab.

Wenn es schnell gehen soll, können Sie auch Deklaration und Wertzuweisung in einem Rutsch durchführen, d. h., Sie *initialisieren* die Variablen gleich bei der Deklaration:

```
int a=17, b=a+4;
```

Einen Wert überschreiben

Und nun kommt etwas Verrücktes, nämlich die Wertzuweisung

```
a=a+1;
```

Wie gesagt, es handelt sich um eine Wertzuweisung und nicht um eine Gleichung. Die Gleichung geht nicht auf, denn wenn Sie auf beiden Seiten *a* subtrahieren, erhalten Sie $0 = 1$. Etwas Falscheres ist in der Mathematik kaum denkbar.

Aber das Gleichheitszeichen ist kein Gleichheitszeichen, sondern das Wertzuweisungszeichen.

Die Wertzuweisung funktioniert genau wie im vorigen Beispiel: Der Computer wertet den Ausdruck auf der rechten Seite des Wertzuweisungszeichens aus, indem er den Wert von *a* (also 17) aus dem Speicher liest, die 1 dazuzählt und das Ergebnis 18 der Variablen *a* als neuen Wert wieder zuweist. Der alte Wert von *a* wird dabei überschrieben.

Dies ist etwas ganz Normales: Wenn Sie einer Variablen einen Wert zuweisen, dann wird der Wert, den die Variable vorher hatte, überschrieben. Er ist unwiederbringlich verloren. Es sei denn, Sie haben sich diesen Wert in einer anderen Variablen gemerkt und können ihn bei Bedarf wieder aus dem Ärmel zaubern ...

Genauso gehen Sie nämlich vor, wenn Sie die Werte der Variablen *a* und *b* vertauschen wollen. Denn so geht es *nicht*:

```
b=a;
a=b;
```

Obwohl die Idee richtig ist: Sie weisen der Variablen *b* den Wert von *a* zu, und der Variablen *a* weisen Sie den Wert von *b* zu. Aber weil der Computer diese Anweisungen *nacheinander* ausführt, ist der Wert von *b* schon überschrieben worden, wenn Sie eigentlich den ursprünglichen Wert von *b* noch brauchen.

Es hilft nichts: Sie brauchen eine Hilfsvariable *h*, unter der Sie sich den alten Wert von *b* merken. Dann haben Sie diesen Wert noch zur Verfügung, auch wenn *b* bereits einen neuen Wert hat. Und hier ist das Programm zur Vertauschung der Werte von *a* und *b*:

```
h=b;
b=a;
a=h;
```

Die Variable *h* deklarieren Sie natürlich vorher. In der Variablen *h* merken Sie sich den ursprünglichen Wert von *b*. Nach der Vertauschung benötigen Sie die Variable *h* nicht mehr. Sie steht einsam im Speicher, hat irgendeinen Wert und wartet darauf, dass sie vielleicht noch einmal gebraucht wird: zum Beispiel, wenn Sie noch eine Vertauschung durchführen wollen. Dann können Sie die Variable *h* wiederverwenden. Denn das Vertauschen von Werten im Speicher ist eine grundlegende Operation, die Sie bei Sortierverfahren stets und ständig benutzen. Später mehr dazu ...

Python

Übrigens, in der Programmiersprache Python können Sie die Werte von *a* und *b* parallel, also nicht nacheinander, sondern gleichzeitig, vertauschen. Dann benötigen Sie keine Hilfsvariable:

```
a, b = b, a
```

Numerische Datentypen und Operationen

Den Datentyp `int` für ganzzahlige Werte haben Sie kennengelernt. Daneben gibt es den Datentyp `double` für Kommazahlen. Die seltsame Bezeichnung `double` kommt daher, dass Kommazahlen im Computer mit 64 Bit dargestellt werden, also mit doppelt so vielen Bits wie mit 32 Bit.

Sie können mit numerischen Daten in gewohnter Weise rechnen: addieren, subtrahieren, multiplizieren und dividieren. Meistens jedenfalls, manchmal aber auch nicht. Dies hängt mit der Darstellung der Zahlen im Computer zusammen.

Programmieren Sie drauflos, aber wenn Sie unerwartete Ergebnisse erhalten, dann rufen Sie sich bitte folgende Punkte in Erinnerung:

Der Computer kann nicht die unendlich vielen Nachkommastellen beispielsweise der Zahl π darstellen, sondern er muss die Zahl auf eine bestimmte Stellenzahl runden. Noch nicht einmal die Zahl 0,4 kann er exakt darstellen, denn im Binärsystem ist 0,4 eine periodische Zahl. Es kommt also zu Rundungsfehlern.

Der Computer kann keine beliebig großen ganzen Zahlen darstellen, sondern er macht bei $2^{31} - 1$ Schluss (bzw. bei $2^{63} - 1$ beim Ganzzahl-Datentyp `long`). Was passiert, wenn im Verlauf einer Berechnung eine zu große Zahl herauskommt? Wenn Sie zum Beispiel 46.000 und 47.000 miteinander multiplizieren? Dann kommt, obwohl Sie es wohl kaum erwarten, eine Zahl heraus, die schon zu groß ist, größer als $2^{31} - 1$.

Und nun kommt eine Fehlentscheidung der Entwickler der Programmiersprache ins Spiel: Es wird keine Fehlermeldung ausgegeben, dass die Zahl zu groß ist, sondern was nicht passt, wird passend gemacht. Eine zu große Zahl wird einfach modulo 2^{32} reduziert. Was das genau bedeutet, erfahren Sie in Kapitel 17, wenn es um die Modulo-Rechnung geht. Das Ergebnis in diesem Beispiel ist eine negative Zahl. Komisch, hatten Sie nicht zwei positive Zahlen miteinander multipliziert?

Welche Operationen Sie mit Daten durchführen können, hängt von den Datentypen ab. Nun sind `int` und `double` zwei verschiedene Datentypen, und die ganzzahlige Division ist etwas anderes als die normale Division. Dummerweise verwendet Java für beide Operationen dasselbe Rechenzeichen /. Java interpretiert das Zeichen / als ganzzahlige Division, wenn beide Operanden vom Typ `int` sind. Wundern Sie sich also nicht, dass $1/2 = 0$ ist.

Wenn Sie jetzt, nach all diesen Warnungen, das dumpfe Gefühl beschleicht, dass Sie lieber die Finger von Java lassen, so ist das verständlich. Aber es ist so wie im wirklichen Leben: Es kann einmal etwas schiefgehen, aber meistens geht alles gut!

Altlasten

Auch moderne Programmiersprachen enthalten noch Relikte aus grauer Vorzeit der Datenverarbeitung. Teilweise wohlbedacht, um kompatibel zu alten Versionen zu bleiben – in denen häufig die Perspektive der Programmiersprachen-Entwickler im Vordergrund stand. Und die Entwickler hatten mit harten Rahmenbedingungen zu kämpfen: Computer waren langsam und hatten wenig Speicher.

Darum müssen Sie Nerd sein, jedenfalls bis zu einem gewissen Grade– schade. Aber lassen Sie sich dadurch nicht entmutigen. Wenn Dinge wie eben dargestellt passieren, ist es nicht Ihr Fehler, sondern Fehler der Entwickler – oder wenn nicht Fehler, dann Folge der Beschränkungen der (noch gar nicht so lange zurückliegenden) Frühzeit der Datenverarbeitung.

Typumwandlung bei numerischen Datentypen

Nun aber einmal etwas Positives: Java kann automatisch ganze Zahlen in Kommazahlen umwandeln. Sie können also schreiben:

```
double x=3;
```

Die Zahl 3 ist ein Wert vom Typ `int`. Bevor Java diesen Wert der `double`-Variablen *x* zuweist, wandelt es den Wert in den Typ `double` um, also in den Wert 3.0.

Umgekehrt geht es nicht, jedenfalls nicht automatisch. Wenn Sie schreiben

```
int n=3.0;
```

so erhalten Sie eine Fehlermeldung. Auch wenn 3.0 offensichtlich ein ganzzahliger Wert ist, so ist er jedoch eine Kommazahl, also vom Typ `double`. Und Werte vom Typ `double` werden nicht automatisch in den Typ `int` umgewandelt.

Bedingte Anweisung

Was ist, wenn Sie die Werte von *a* und *b* nur dann vertauschen möchten, wenn diese »in falscher Reihenfolge« stehen? Beim Sortieren von Zahlen möchten Sie ja, dass die kleineren Zahlen vor den größeren Zahlen erscheinen. Wenn Sie nun möchten, dass die Variable *a* den kleineren Wert erhält und die Variable *b* den größeren Wert, dann müssen Sie zunächst die Werte von *a* und *b* miteinander vergleichen. Wenn sich dann herausstellt, dass es genau umgekehrt ist, dass *a* den größeren und *b* den kleineren Wert enthält, dann müssen Sie die Werte von *a* und *b* vertauschen. Aber nur dann. Ansonsten brauchen Sie nichts zu tun, nämlich wenn die Werte von *a* und *b* schon von vorneherein in der gewünschten Reihenfolge stehen.

If-Anweisung

Was der Computer tun soll, ist also abhängig von einer Bedingung, in diesem Fall von einem Vergleich von zwei Werten. Mal soll er das eine tun, mal soll er das andere tun (bzw. soll er gar nichts tun). Der Programmfluss verzweigt sich also an dieser Stelle – je nachdem, wie der Vergleich ausgeht, schlägt das Programm die eine Richtung ein oder die andere. Sie werden dies gleich in grafischer Veranschaulichung sehen.

Zunächst einmal sehen Sie hier das entsprechende Java-Programmstück:

```
if (a>b)
{
    h=b;
    b=a;
    a=h;
}
```

Das englische Wort *if* bedeutet *wenn* oder *falls*. Falls also der Wert von *a* größer als der Wert von *b* ist, dann sollen die Werte von *a* und *b* vertauscht werden. Ansonsten soll nichts getan werden.

Und noch etwas Neues sehen Sie hier: Die drei Wertzuweisungen werden durch die geschweiften Klammern zu einem Anweisungsblock zusammengefasst. Dadurch wirken sie gewissermaßen wie eine einzelne Anweisung.

Der Computer wertet den Ausdruck *a* > *b* folgendermaßen aus: Er schaut im Speicher nach, welche Werte die Variablen *a* und *b* haben und vergleicht diese miteinander. Nun gibt es zwei Möglichkeiten: Entweder ist der Wert von *a* größer als der Wert von *b*, oder er ist es nicht. Oder anders ausgedrückt: Entweder ist die Aussage, dass der Wert von *a* größer als der Wert von *b* ist, *wahr*, oder sie ist *falsch*.

Die Auswertung des Ausdrucks *a* > *b* ergibt also einen Wahrheitswert, nämlich einen der Werte wahr oder falsch, oder in Java *true* oder *false*. Damit lernen Sie einen neuen Datentyp kennen, den Datentyp `boolean`, dieser besteht nur aus den beiden Werten *true* und *false*. Mehr dazu weiter unten in diesem Kapitel.

Wie geht es weiter, nachdem der Computer die Bedingung ausgewertet hat? Wenn die Bedingung den Wert *true* hat, so führt der Computer die unmittelbar darauffolgende Anweisung aus. Die Anweisung kann auch ein Anweisungsblock sein, also mehrere durch geschweifte Klammern zusammengefasste Anweisungen. Wenn dagegen die Bedingung *false* ist, so überspringt er die unmittelbar darauffolgende Anweisung. In beiden Fällen fährt der Computer anschließend mit den Anweisungen fort, die danach kommen.

Ein Flussdiagramm veranschaulicht den Verlauf der Programmausführung (Abbildung 1.2). Der Programmfluss kommt von oben und verzweigt dann abhängig von einer Bedingung. Je nachdem, ob die Bedingung *true* oder *false* ist, werden entweder die Anweisungen ausgeführt oder übersprungen. Danach vereinigen sich die Zweige wieder.

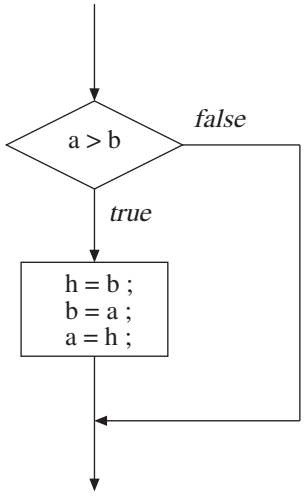

Abbildung 1.2: Flussdiagramm der If-Anweisung aus dem Beispiel

If-Else-Anweisung

Das englische Wort *else* bedeutet *sonst*. In einer If-Else-Anweisung sagen Sie dem Computer nicht nur, was er tun soll, *wenn* eine Bedingung erfüllt ist, sondern auch, was er *sonst* tun soll.

Um beispielsweise das Maximum von zwei Zahlen *a* und *b* zu berechnen, schreiben Sie folgendes Programmstück:

```
if (a>b)
    max=a;
else
    max=b;
```

Auch hier veranschaulichen Sie sich den Verlauf der Programmausführung wieder mit einem Flussdiagramm (Abbildung 1.3).

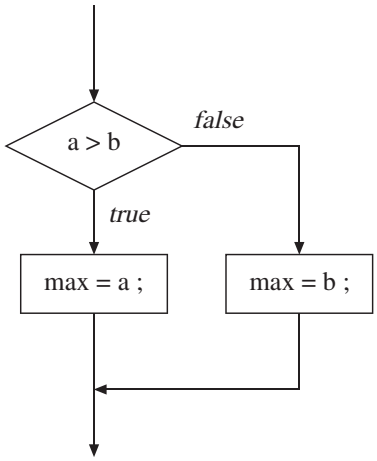

Abbildung 1.3: Flussdiagramm der If-Else-Anweisung aus dem Beispiel

Je nachdem, ob $a > b$ ist oder nicht, weisen Sie der Variablen *max* entweder den Wert von *a* oder den Wert von *b* zu. Der If-Teil der If-Else-Anweisung wird ausgeführt, wenn die Bedingung erfüllt ist, also den Wert *true* hat; der Else-Teil der If-Else-Anweisung wird ausgeführt, wenn die Bedingung nicht erfüllt ist, also den Wert *false* hat.

Sowohl der If-Teil als auch der Else-Teil können auch aus mehreren Anweisungen bestehen, dann schließen Sie diese Anweisungen in geschweifte Klammern ein.

Flussdiagramme zeichnen

Für das Zeichnen von Flussdiagrammen verwenden Sie im einfachsten Fall die drei folgenden Elemente (Abbildung 1.4):

- ✔ die Verzweigung abhängig von einer Bedingung (a),
- ✔ den Anweisungsblock (b),
- ✔ die Vereinigung von zwei Zweigen (c).

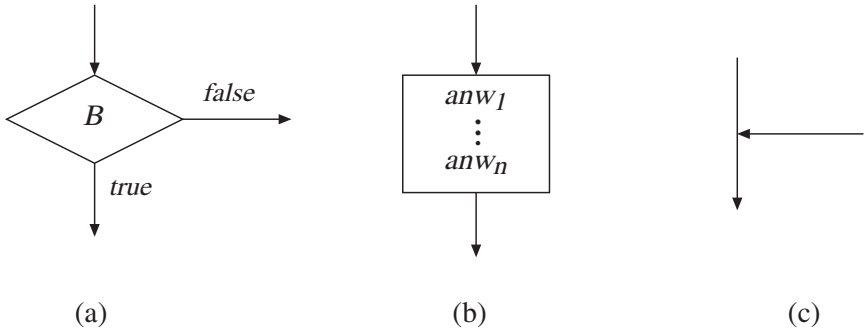

Abbildung 1.4: Elemente, aus denen ein Flussdiagramm besteht

Benutzen Sie diese Elemente in der dargestellten Form. Vergessen Sie nicht die Pfeilspitzen, um die Richtung des Programmflusses deutlich zu machen.

- ✔ Eine Verzweigung ist eine Raute, in der eine Bedingung steht. Der Programmfluss kommt von oben. Wenn die Bedingung erfüllt ist, verzweigt das Programm nach unten, ansonsten nach rechts. Schreiben Sie *true* und *false* an die beiden Zweige (oder *ja* und *nein*), um deutlich zu machen, in welche Richtung es weitergeht. Verzweigungen des Programms gibt es ausschließlich aufgrund von Bedingungen.
- ✔ Ein Anweisungsblock ist ein rechteckiger Kasten, in dem die betreffenden Anweisungen stehen. Der Programmfluss kommt von oben und führt unten wieder heraus.
- ✔ Eine Vereinigung von zwei Zweigen wird dadurch dargestellt, dass der eine Zweig auf den anderen trifft, der dann weiterführt.

Am besten lassen Sie das Programm immer von oben nach unten fließen, außer bei Verzweigungen, dann geht es auch einmal seitwärts, und bei Schleifen, dann geht es zurück (siehe nächstes Kapitel). Aber führen Sie keine Pfeile seitwärts in Anweisungsblöcke hinein.

Datentyp boolean

Zahlen als Datentypen sind für Sie nichts Besonderes – Sie kennen den Datentyp `int` für ganze Zahlen und den Datentyp `double` für Kommazahlen. Wie Sie wissen, ist ein Datentyp im Grunde nichts anderes als ein Wertebereich, zusammen mit den Operationen, mit denen sich diese Werte verknüpfen lassen. So ist der Wertebereich des Datentyps `int` die Zahlenmenge {-2.147.483.648 , ..., +2.147.483.647}, zulässige Operationen sind Addition, Subtraktion, Multiplikation und ganzzahlige Division mit Rest. Außerdem Vergleichsoperationen und (in diesem Buch nicht behandelt) Bit-Operationen.

Das Verrückte ist: Es gibt auch einen Datentyp, der nur aus zwei Werten besteht, nämlich den Werten *false* und *true*. Dies ist der Datentyp `boolean`, benannt nach dem englischen Mathematiker George Boole, dem Begründer der formalen Logik. Und tatsächlich entsprechen *false* und *true* genau den Wahrheitswerten, die Sie in Kapitel 12 über Logik kennenlernen werden. Dort werden sie mit 0 und 1 bezeichnet; in Java werden die Wortsymbole `false` und `true` verwendet, um Verwechslungen mit den Zahlen 0 und 1 zu vermeiden.

Vergleiche ergeben boolean-Werte

Das Ergebnis eines Vergleichs von zwei Zahlenwerten, wie etwa $a > b$ im Beispiel der If-Anweisung, ergibt entweder *false* oder *true*, also einen Wert vom Typ `boolean`. In der Programmiersprache Java gibt es folgende Vergleichsoperatoren:

`<`	kleiner
`<=`	kleiner oder gleich
`>`	größer
`>=`	größer oder gleich
`==`	gleich
`!=`	ungleich

Denken Sie daran, dass Sie für einen Vergleich auf Gleichheit das doppelte Gleichheitszeichen benutzen. Das einfache Gleichheitszeichen ist das Wertzuweisungszeichen.

Wertzuweisungen an Variablen vom Typ boolean

Deklaration und Wertzuweisung funktionieren beim Datentyp `boolean` genauso, wie Sie es beispielsweise vom Datentyp `int` kennen. Sie deklarieren Variablen vom Typ `boolean`, auch boolesche Variablen genannt:

```
boolean a, b, c;
```

Als Nächstes weisen Sie den Variablen Werte zu. Dies können die Konstanten *false* oder *true* sein, oder es können Werte vom Typ `boolean` sein, die sich aus der Auswertung von Ausdrücken ergeben, zum Beispiel als das Ergebnis eines Vergleichs.

```
a=true;
int x=3;
b=x>0;
```

Betrachten Sie die Wertzuweisung `b=x>0;`. Die boolesche Variable *b* erhält den Wert *true*, wenn $x > 0$ gilt, ansonsten erhält sie den Wert *false*. Deshalb brauchen Sie *nicht* zu schreiben

```
if (x>0)
    b=true;
else
    b=false;
```

sondern Sie können das Ergebnis des Vergleichs, *true* oder *false*, der booleschen Variablen direkt zuweisen: `b=x>0;`.

Und wenn Sie eine boolesche Variable *b* mit einem Wert belegt haben und diesen Wert in einer If-Anweisung abfragen, so brauchen Sie *nicht* zu schreiben

```
if (b==true)
    ...
```

sondern schreiben Sie einfach

```
if (b)
    ...
```

Denn *b* hat ja einen der Werte *true* oder *false* – genau das, was Sie in der Bedingung der If-Anweisung brauchen.

Boolesche Operationen

Nachdem der Wertebereich des Datentyps `boolean` feststeht, fehlen noch die zulässigen Operationen. Wie nicht anders zu erwarten, sind dies die logischen Verknüpfungen Und, Oder und Nicht. In der Logik werden diese Verknüpfungen durch die Zeichen ∧, ∨ und ¬ dargestellt; in Java werden die folgenden Zeichen verwendet:

Verknüpfung	Logik	Java
logisches Und	∧	&
logisches Oder	∨	|
logisches Nicht	¬	!

Sie verknüpfen also boolesche Variablen und Ausdrücke, die einen Wert vom Typ `boolean` ergeben, mit diesen Operationen beispielsweise wie folgt:

```
boolean found=false;
if (!found & i<n)
    ...
```

Bedingtes Und und bedingtes Oder

Darüber hinaus gibt es in Java noch das *bedingte Und* und das *bedingte Oder*. Was hat es damit auf sich?

In der Logik gilt, dass eine Aussage, die aus zwei Teilaussagen besteht, die mit Und verknüpft sind, bereits dann falsch ist, wenn die erste Teilaussage falsch ist. Dann kommt es auf die zweite Teilaussage gar nicht mehr an. Auf die zweite Teilaussage kommt es nur dann an, wenn die erste Teilaussage wahr ist.

In Java ist die Situation entsprechend: Zwei Ausdrücke vom Typ *boolean* werden durch Und miteinander verknüpft. Sie haben nun die Wahl: Entweder möchten Sie,

- ✔ dass beide Ausdrücke ausgewertet werden oder
- ✔ dass der erste Ausdruck ausgewertet wird und der zweite nur dann, wenn der erste Ausdruck *true* ist.

Im ersten Fall verwenden Sie den Operator `&`. Im zweiten Fall verwenden Sie den Operator `&&`, das bedingte Und.

Es gibt in der Programmierung manchmal Situationen, in denen beide Ausdrücke ausgewertet werden *müssen*. Und es gibt Situationen, in denen der zweite Ausdruck nur dann ausgewertet werden *darf*, wenn der erste Ausdruck *true* ist.

Deswegen ist es gut, dass es das bedingte Und gibt. Beim bedingten Oder verhält es sich entsprechend: Der zweite Ausdruck wird nur dann ausgewertet, wenn der erste Ausdruck *false* ist. Der Operator für das bedingte Oder ist `||`.

Kommentare

Noch sind Sie am Anfang, aber wenn Ihr Softwareprojekt umfangreicher wird, ist es wichtig, dass Sie sorgfältig dokumentieren, was Sie programmiert haben – und zwar an Ort und Stelle im Programmtext in Form von Kommentaren.

Ein *Kommentar* wird durch `//` eingeleitet und reicht bis zum Ende der Zeile. Aber Sie können auch mehrzeilige Kommentare verfassen, dann schließen Sie den Kommentar zwischen `/*` und `*/` ein. Im Kommentar geben Sie ein paar verständliche Erläuterungen zu den folgenden Programmanweisungen. Kommentare werden bei der Programmausführung vom Computer nicht beachtet, sie dienen nur dem Verständnis eines menschlichen Lesers.

Sagen Sie nicht: »Wozu ein Kommentar, ich weiß doch, was ich programmiert habe!« Denn nach einem halben Jahr wissen Sie es vielleicht schon nicht mehr. Und ein anderer, der Ihren Programmtext liest, weiß es schon gar nicht.

Schauen Sie sich an, wie Kommentare in folgendem ersten Programm zum Üben eingesetzt sind. Später werden Sie mit Ihren Kommentaren jedoch nicht mehr in allen Einzelheiten die Programmstruktur erläutern, sondern mehr inhaltlich den Zweck bestimmter Programmstücke beschreiben.

Zum Üben

Zum Programmieren brauchen Sie eine Programmierumgebung. Dies ist eine Software, die es Ihnen erleichtert, ein Programm zu schreiben, es auf korrekte Syntax zu überprüfen und schließlich zu starten.

Am besten installieren Sie auf Ihrem Computer die Programmierumgebung *Eclipse*. Vorher installieren Sie noch *Java*. Eclipse ist zwar für den Hausgebrauch überdimensioniert, aber das schadet nicht.

Erzeugen Sie in Eclipse ein Projekt *Vorkurs*, in diesem Projekt ein Package *teil1* und in diesem Package eine Klasse *MeinErstesProgramm*.

Nun erscheint ein Editor-Fenster, in welchem Sie diese Klasse mit Inhalt füllen können. Das Wichtigste ist die Main-Funktion. Dies ist dasjenige Programmstück, das ausgeführt wird, wenn Sie anschließend auf *Run* klicken.

Schreiben Sie folgende Main-Funktion in die Klasse *MeinErstesProgramm*:

```
public class MeinErstesProgramm
{
    public static void main(String[] args)
    {
        // Variablen deklarieren
        int a, b, c;
        // Berechnungen ausführen
        a=17;
        b=4;
        c=a+b;
        // Ergebnis ausgeben
        System.out.println(c);
    }
}
```

Fragen Sie nicht, was *public*, *static* und *void* bedeuten. Schreiben Sie es einfach erst einmal so ab. Wenn Sie dann in Eclipse im Menü `Run | Run as | Java Application` wählen, startet das Programm. Daraufhin erscheint im Ausgabefenster das Ergebnis 21.

Wichtig ist, dass Sie jetzt mit dem Inhalt der Funktion *main* ausgiebigst herumprobieren. Ersetzen Sie das +-Zeichen durch ein ∗-Zeichen. Probieren Sie aus, ob Sie die Werte von *a* und *b* vertauschen können. Schreiben Sie eine If-Anweisung. Gewinnen Sie so nach und nach Vertrauen zu Ihren ersten Programmierkünsten.