Das Java-ABC

Ein Java-Programm besteht aus Klassen und Objekten, diese sind aus Methoden und Variablen aufgebaut. Methoden wiederum bestehen aus Anweisungen und Ausdrücken, die ihrerseits aus Operatoren bestehen.

An dieser Stelle könnte sich bei Ihnen die Befürchtung breit machen, dass Java wie eine verschachtelte russische Puppe, eine Matrjoschka, sei. Scheinbar hat jede dieser Puppen eine weitere kleinere Puppe in sich, die genauso kompliziert und detailreich wie die größere Mama ist.

Dieses Kapitel schiebt die großen Puppen beiseite, um die kleinsten Elemente der Java-Programmierung zu enthüllen. Sie lassen Klassen, Objekte und Methoden hinter sich und untersuchen die elementaren Dinge, die Sie in einer einzelnen Zeile Java-Code tun können.

Die folgenden Themen werden heute behandelt:

- Java-Anweisungen und -Ausdrücke
- Variablen und Datentypen
- Konstanten
- Kommentare
- Literale
- Arithmetik
- Vergleiche
- Logische Operatoren

Da Java sehr eng an C und C++ angelehnt ist, wird ein großer Teil der Themen in diesem Kapitel Programmierern, die in diesen Sprachen versiert sind, bekannt sein.

3.1 Anweisungen und Ausdrücke

Alle Aufgaben, die Sie in einem Java-Programm ausführen wollen, können in eine Folge von Anweisungen aufgeteilt werden.

Eine *Anweisung* (engl. Statement) ist ein einzelner Befehl einer Programmiersprache, der dafür sorgt, dass etwas passiert.

Anweisungen stehen für eine einzelne Aktion, die in einem Java-Programm ausgeführt wird. Die folgenden Zeilen stellen allesamt einfache Java-Anweisungen dar:

```
int weight = 295;

System.out.println("Free the bound periodicals!");

song.duration = 230;
```

Einige Anweisungen erzeugen einen Wert, wie das beim Addieren zweier Zahlen in einem Programm der Fall ist. Derartige Anweisungen werden als Ausdrücke bezeichnet.

Ein *Ausdruck* (engl. Expression) ist eine Anweisung, die als Ergebnis einen Wert produziert. Dieser Wert kann zur späteren Verwendung im Programm gespeichert, direkt in einer anderen Anweisung verwendet oder überhaupt nicht beachtet werden. Der Wert, den eine Anweisung erzeugt, wird *Rückgabewert* genannt.

Manche Ausdrücke erzeugen numerische Rückgabewerte, wie das beim Addieren zweier Zahlen in dem oben erwähnten Beispiel der Fall war. Andere erzeugen einen booleschen Wert – `true` oder `false` – oder sogar ein Java-Objekt. Diese werden etwas später am heutigen Tag besprochen.

Obwohl die meisten Java-Programme je eine Anweisung pro Zeile enthalten, sagt diese nicht aus, wo eine Anweisung beginnt und wo sie endet. Es handelt sich dabei lediglich um eine Layout-Konvention. Jede Anweisung wird mit einem Strichpunkt (`;`) abgeschlossen. Ein Programmierer kann mehr als eine Anweisung in eine Zeile setzen, und das Programm wird trotzdem erfolgreich kompiliert werden, wie das im folgenden der Fall ist:

```
dante.speed = 2; dante.temperature = 510;
```

Anweisungen werden in Java mit einem Paar geschweifter Klammern (`{}`) zusammengefasst. Eine Gruppe von Anweisungen, die sich in diesen Klammern befindet, wird als *Block* oder *Blockanweisung* bezeichnet. Sie werden darüber an Tag 5 mehr lernen.

3.2 Variablen und Datentypen

In der `VolcanoRobot`-Applikation, die Sie gestern erstellt haben, haben Sie Variablen verwendet, um bestimmte Informationen abzulegen.

Variablen sind Orte, an denen Informationen gespeichert werden können, während ein Programm läuft. Der Wert der Variablen kann jederzeit im Programm geändert werden – daher auch der Name.

Um eine Variable zu erstellen, müssen Sie ihr einen Namen geben und festlegen, welchen Typ von Informationen sie speichern soll. Sie können einer Variablen auch bei der Erzeugung einen Wert zuweisen.

In Java gibt es drei Arten von Variablen: Instanzvariablen, Klassenvariablen und lokale Variablen.

Instanzvariablen werden, wie Sie gestern gelernt haben, zur Definition der Attribute eines Objekts verwendet. *Klassenvariablen* definieren die Attribute einer gesamten Klasse von Objekten und beziehen sich auf alle Instanzen einer Klasse.

Lokale Variablen werden innerhalb von Methodendefinitionen oder in noch kleineren Blockanweisungen innerhalb von Methoden verwendet. Diese Variablen können nur verwendet werden, während die Methode oder der Block vom Java-Interpreter ausgeführt wird, und hören sofort danach auf zu existieren.

Während alle diese Variablen mehr oder weniger auf dieselbe Art erzeugt werden, erfolgt die Verwendung von Klassen- und Instanzvariablen anders als die lokaler Variablen. Sie lernen heute lokale Variablen kennen. Instanz- und Klassenvariablen werden wir am Tag 4 durchnehmen.

Anders als andere Sprachen verfügt Java nicht über *globale Variablen* – Variablen, auf die überall in einem Programm zugegriffen werden kann. Instanz- und Klassenvariablen werden für den Informationsaustausch zwischen einzelnen Objekten verwendet, weshalb keine Notwendigkeit für globale Variablen besteht.

Variablen erstellen

Bevor Sie eine Variable in einem Javaprogramm verwenden können, müssen Sie die Variable erst einmal erzeugen, indem Sie ihren Namen und die Art der Information deklarieren, die die Variable speichern soll. Als Erstes wird dabei die Art der Information angegeben, gefolgt von dem Namen der Variablen. Hier einige Beispiele für Variablendeklarationen:

```
int loanLength;

String message;

boolean gameOver;
```

Sie lernen etwas später am heutigen Tag mehr über Variablentypen. Eventuell sind Sie aber schon mit den Typen, die in diesem Beispiel verwendet wurden, vertraut. Der Typ `int` repräsentiert Integer (ganze Zahlen), der Typ `boolean` wird für `true-false`-Werte verwendet, und `String` ist ein spezieller Variablentyp, der zum Speichern von Text verwendet wird.

Die Deklaration einer lokalen Variable kann an jeder Stelle einer Methode stehen, wie jede andere Java-Anweisung auch. Bevor eine lokale Variable verwendet werden kann, muss sie deklariert werden. Normalerweise werden Variablendeklarationen direkt im Anschluss an die Anweisung platziert, die eine Methode identifiziert.

Im folgenden Beispiel werden drei Variablen am Beginn der `main()`-Methode des Programms deklariert:

```
public static void main (String[] arguments ) {
    int total;
    String reportTitle;
    boolean active;
}
```

Wenn Sie mehrere Variablen desselben Typs deklarieren, können Sie dies in einer einzigen Zeile tun. Dazu trennen Sie die einzelnen Variablennamen mit Kommata. Die folgende Anweisung erzeugt drei `String`-Variablen mit den Namen `street`, `city` und `state`:

```
String street, city, state;
```

Variablen kann bei ihrer Erstellung ein Wert zugewiesen werden. Dazu verwenden Sie das Gleichheitszeichen (=), gefolgt von dem Wert. Die folgenden Anweisungen erzeugen neue Variablen und weisen diesen Werte zu:

```
int zipcode = 02134;

int box = 350;

boolean pbs = true;

String name = "Zoom", city = "Boston", state ="MA";
```

Wie die letzte Anweisung bereits zeigt, können Sie mehreren Variablen desselben Typs Werte zuweisen, indem Sie sie mit Kommata voneinander trennen.

Lokalen Variablen müssen Werte zugewiesen werden, bevor sie in einem Programm verwendet werden können. Ansonsten kann das Programm nicht erfolgreich kompiliert werden. Aus diesem Grund ist es eine gute Angewohnheit, allen lokalen Variablen Initialisierungswerte zuzuweisen.

Instanz- und Klassenvariablen erhalten automatisch einen Initialisierungswert – abhängig davon, welchen Typ von Information sie aufnehmen sollen:

- Numerische Variablen: `0`
- Zeichen : `'\0'`
- Boolesche Variablen: `false`
- Objekte: `null`

Variablen benennen

Variablennamen müssen in Java mit einem Buchstaben, einem Unterstrich (`_`) oder einem Dollarzeichen (`$`) beginnen. Sie dürfen nicht mit einer Ziffer anfangen. Nach dem ersten Zeichen können Variablennamen jede beliebige Kombination von Buchstaben und Ziffern enthalten.

Java unterstützt den Unicode-Zeichensatz, der neben dem Standardzeichensatz tausende weiterer Zeichen beinhaltet, um internationale Alphabete wiedergeben zu können. Zeichen mit Akzenten und andere Symbole können in Variablennamen verwendet werden, solange sie über eine Unicode-Zeichennummer verfügen.

Wenn Sie eine Variable benennen und diese in einem Programm verwenden, ist es wichtig, daran zu denken, dass Java die Groß-/Kleinschreibung beachtet. Das heißt, die Verwendung von Groß- und Kleinbuchstaben muss konsistent sein. Aus diesem Grund kann es in einem Programm eine Variable `X` und eine andere Variable `x` geben – und eine `rose` ist keine `Rose` und auch keine `ROSE`.

In den Programmen in diesem Buch und auch außerhalb werden Variablen oft mit aussagekräftigen Namen versehen, die aus mehreren zusammengeschriebenen Wörtern bestehen. Um einzelne Wörter leichter innerhalb des Namens erkennen zu können, wird die folgende Faustregel verwendet:

- Der erste Buchstabe eines Variablennamens ist klein.
- Jedes darauffolgende Wort in dem Namen beginnt mit einem Großbuchstaben.
- Alle anderen Buchstaben sind Kleinbuchstaben.

Die folgenden Variablendeklarationen folgen diesen Konventionen:

```
Button loadFile;

int areaCode;

boolean quitGame;
```

Variablentypen

Neben dem Namen muss eine Variablendeklaration auch den Typ der Information, die in der Variablen gespeichert werden soll, beinhalten. Als Typ kann einer der folgenden verwendet werden:

- einer der elementaren Datentypen
- der Name einer Klasse oder Schnittstelle
- ein Array

Sie lernen am Tag 5, wie Sie Arrays deklarieren und verwenden. Diese Lektion konzentriert sich auf Variablentypen.

Datentypen

Es gibt acht elementare Variablentypen zum Speichern von Integern, Fließkommazahlen, Zeichen und booleschen Werten. Diese werden oft auch als *primitive Typen* bezeichnet, da sie feste Bestandteile der Sprache und keine Objekte sind. Aus diesem Grund sind diese Typen bei der Anwendung effizienter. Diese Datentypen haben im Gegensatz zu einigen Datentypen in anderen Programmiersprachen unabhängig von der Plattform oder dem Betriebssystem dieselbe Größe und Charakteristik.

Vier der Datentypen können Integer-Werte speichern. Welchen Sie verwenden, hängt von der Größe des zu speichernden Integers ab (siehe Tabelle 3.1)

Typ	Größe	Wertebereich
`byte`	8 Bit	`-128` bis `127`
`short`	16 Bit	`-32.768` bis `32.767`
`int`	32 Bit	`-2.147.483.648` bis `2.147.483.647`
`long`	64 Bit	`-9.223.372.036.854.775.808` bis `9.223.372.036.854.775.807`

Tabelle 3.1: Integer-Typen

Alle diese Typen besitzen ein Vorzeichen, d.h. sie können sowohl positive als auch negative Werte aufnehmen. Welchen Typ Sie für eine Variable verwenden, hängt von dem Wertebereich ab, den die Variable aufnehmen soll. Keine Integer-Variable kann einen Wert, der zu groß oder zu klein für den zugewiesenen Variablentyp ist, verlässlich speichern. Sie sollten bei der Zuweisung des Typs Vorsicht walten lassen.

Eine andere Art von Zahlen, die gespeichert werden können, sind die Fließkomma-Zahlen, die die Typen `float` und `double` haben. Fließkomma-Zahlen sind Zahlen mit einem Dezimalanteil. Der Typ `float` sollte für die meisten Benutzer ausreichend sein, da er jede beliebige Zahl zwischen 1.4E-45 bis 3.4E+38 verarbeiten kann. Falls nicht, kann `double` für Zahlen zwischen `4.9E-324` bis `1.7E+308` eingesetzt werden.

Der Typ `char` wird für einzelne Zeichen, wie z.B. Buchstaben, Ziffern, Interpunktionszeichen und andere Symbole verwendet.

Der letzte der acht elementaren Datentypen ist `boolean`. Wie Sie bereits gelernt haben, speichern boolesche Variablen in Java entweder den Wert `true` oder den Wert `false`.

All diese Variablentypen sind in Kleinbuchstaben definiert, und Sie müssen sie auch in dieser Form in Programmen verwenden. Es gibt Klassen, die denselben Namen wie einige dieser Datentypen besitzen, allerdings mit anderer Groß-/Kleinschreibung – z.B. `Boolean` und `Char`. Diese haben eine andere Funktionalität in einem Java-Programm, sodass Sie diese Schreibungen nicht einfach austauschen können. Sie werden morgen erfahren, wie Sie diese speziellen Klassen verwenden.

Klassentypen

Neben den acht elementaren Typen kann eine Variable eine Klasse als Typ haben, wie das in den folgenden Beispielen der Fall ist:

```
String lastName = "Hopper";

Color hair;

VolcanoRobot vr;
```

Wenn eine Variable eine Klasse als Typ hat, dann bezieht sich diese Variable auf ein Objekt dieser Klasse oder einer ihrer Subklassen.

Das letzte Beispiel in der obigen Liste, `VolcanoRobot vr;` erzeugt eine Variable mit dem Namen `vr`, die für das Objekt `VolcanoRobot` reserviert ist, und zwar unabhängig davon, ob das Objekt bereits existiert. Morgen lernen Sie, wie man Objekte mit Variablen assoziiert.

Eine Superklasse als Variablentyp zu referenzieren, kann sinnvoll sein, wenn die Variable eine von mehreren Subklassen sein kann. Nehmen Sie z.B. eine Klassenhierarchie mit der Superklasse `CommandButton` und den drei Subklassen `RadioButton`, `CheckboxButton` und `ClickButton`. Wenn Sie eine `CommandButton`-Variable mit dem Namen `widget` erzeugen, kann diese auf ein `RadioButton`-, ein `CheckboxButton`- oder ein `ClickButton`-Objekt verweisen.

Wenn Sie eine Variable vom Typ `Object` deklarieren, heißt das, dass diese Variable mit jedem beliebigen Objekt assoziiert werden kann.

In Java gibt es nichts, was der `typedef`-Anweisung aus C und C++ entsprechen würde. Um neue Typen in Java zu deklarieren, müssen Sie eine Klasse deklarieren. Diese Klasse können Sie dann als Typ für Variablen verwenden.

Variablen Werte zuweisen

Sobald eine Variable deklariert wurde, kann ihr über den Zuweisungsoperator (das Gleichheitszeichen =) ein Wert zugewiesen werden. Hier finden Sie zwei Beispiele für Zuweisungsanweisungen:

```
idCode = 8675309;

accountOverdrawn = false;
```

Konstanten

Variablen sind sehr nützlich, wenn Sie Informationen speichern wollen, die man zur Laufzeit eines Programms ändern können soll. Soll sich der Wert allerdings zur Laufzeit eines Programms nicht ändern, können Sie einen speziellen Variablentyp verwenden: die Konstanten.

Eine *Konstante* ist eine Variable, deren Wert sich nie ändert (was angesichts des Namens »Variable« ein Widerspruch in sich ist).

Konstanten sind sehr nützlich für die Definition von Werten, die allen Methoden eines Objekts zur Verfügung stehen sollen. Mit anderen Worten kann man mit Konstanten unveränderlichen, objektweit genutzten Werten einen aussagekräftigen Namen geben. In Java können Sie mit allen Variablenarten Konstanten erzeugen: mit Instanzvariablen, Klassenvariablen und lokalen Variablen.

Konstante lokale Variablen waren in Java 1.0 nicht möglich, stehen aber in allen späteren Versionen der Sprache zur Verfügung. Dies wird wichtig, wenn Sie ein Applet erstellen, das zu Java 1.0 völlig kompatibel sein soll, wie Sie das am 7. Tag lernen werden.

Um eine Konstante zu deklarieren, benutzen Sie das Schlüsselwort `final` vor der Variablendeklaration und geben für diese Variable einen Anfangswert an:

```
final float PI = 3.141592;

final boolean DEBUG = false;

final int PENALTY = 25;
```

In diesen Beispielen werden die Namen der Konstanten mit Großbuchstaben geschrieben: `PI`, `DEBUG` und `PENALTY`. Das ist nicht verpflichtend, aber diese Konvention wird von vielen Java-Programmierern eingehalten – Sun macht das so in Java-Klassenbibliothek. Die Schreibung mit Großbuchstaben macht klar, dass Sie eine Konstante verwenden.

Konstanten sind auch nützlich zur Benennung verschiedener Zustände eines Objekts, das dann auf diese Zustände getestet werden kann. Nehmen wir beispielsweise an, Sie haben ein Programm, das Richtungseingaben vom Zahlenblock der Tastatur erwartet – 8 für oben, 4 für links usw. Diese Werte können Sie als konstante Ganzzahlen definieren:

```
final int LEFT = 4;
final int RIGHT = 6;
final int UP = 8;
final int DOWN = 2;
```

Die Verwendung von Variablen macht Programme oft verständlicher. Um diesen Punkt zu verdeutlichen, sollten Sie sich einmal die beiden folgenden Anweisungen ansehen und vergleichen, welche mehr über ihre Funktion aussagt:

```
this.direction = 4;
this.direction = LEFT;
```

3.3 Kommentare

Eine der wichtigsten Möglichkeiten, die Lesbarkeit eines Programms zu verbessern, sind Kommentare.

Kommentare sind Informationen, die in einem Programm einzig für den Nutzen eines menschlichen Betrachters eingefügt wurden, der versucht, herauszufinden, was das Programm tut. Der Java-Compiler ignoriert die Kommentare komplett, wenn er eine ausführbare Version der Java-Quelldatei erstellt.

Es gibt drei verschiedene Arten von Kommentaren, die Sie in Java-Programmen nach Ihrem eigenen Ermessen verwenden können.

Die erste Methode, einen Kommentar in ein Programm einzufügen, ist, dem Kommentartext zwei Schrägstriche (`//`) voranzustellen. Dadurch wird alles nach den Schrägstrichen bis zum Ende der Zeile zu einem Kommentar, wie in der folgenden Anweisung:

```
int creditHours = 3; // Bonusstunden für den Kurs festlegen
```

In diesem Beispiel wird alles ab dem `//` bis zum Ende der Zeile zu einen Kommentar und wird daher vom Java-Compiler nicht beachtet.

Wenn Sie einen Kommentar einfügen wollen, der länger als eine Zeile ist, dann starten Sie den Kommentar mit der Zeichenfolge `/*` und beenden ihn mit `*/`. Alles zwischen diesen beiden Begrenzern wird als Kommentar gesehen, wie in dem folgenden Beispiel:

```
/*Dieses Programm löscht ab und zu alle Dateien auf Ihrer Festplatte. Sollten Sie
eine Rechtschreibprüfung durchführen, wird Ihre Festplatt komplett zerstört. */
```

Der letzte Kommentartyp soll sowohl vom Computer als auch vom Menschen lesbar sein. Wenn Sie einen Kommentar mit der Zeichenfolge `/**` (anstelle von `/*`) einleiten und ihn mit der Zeichenfolge `*/` beenden, wird der Kommentar als offizielle Dokumentation für die Funktionsweise der Klasse und ihrer `public`-Methoden interpretiert.

Diese Art von Kommentar kann von Tools wie `javadoc`, das sich im SDK befindet, gelesen werden. Das Programm `javadoc` verwendet diese offiziellen Kommentare, um eine Reihe von HTML-Dokumenten zu erzeugen, die das Programm, seine Klassenhierarchie und seine Methoden dokumentieren.

Die gesamte offizielle Dokumentation der Klassenbibliothek von Java ist das Ergebnis von `javadoc`-artigen Kommentaren. Sie können sich die Dokumentation von Java 2 im Web unter der folgenden Adresse ansehen:

```
http://java.sun.com/j2se/1.3/docs
```

Sie können sich auch die gesamte Dokumentation auf Ihren Computer herunterladen, um so schneller die einzelnen Seiten aufzurufen. Allerdings haben die Dokumente insgesamt einen Umfang von mehr als 23 MB. Der Link ist:

```
http://java.sun.com/j2se/1.3/docs.html
```

3.4 Literale

Neben Variablen werden Sie in Java-Anweisungen auch Literale verwenden.

Literale sind Zahlen, Texte oder andere Informationen, die direkt einen Wert darstellen.

Literal ist ein Begriff aus der Programmierung, der im Wesentlichen aussagt, dass das, was Sie eingeben, auch das ist, was Sie erhalten. Die folgende Zuweisungsanweisung verwendet ein Literal:

```
int jahr = 2000;
```

Das Literal ist `2000`, da es direkt den Integer-Wert 2000 darstellt. Zahlen, Zeichen und Strings sind sämtlich Beispiele für Literale.

Obwohl die Bedeutung und Anwendung von Literalen in den meisten Fällen sehr intuitiv erscheint, verfügt Java über einige Sondertypen von Literalen, die unterschiedliche Arten von Zahlen, Zeichen, Strings und booleschen Werten repräsentieren.

Zahlen-Literale

Java besitzt mehrere Zahlen-Literale. Die Zahl `4` ist z.B. ein Integer-Literal des Variablentyps `int`. Es könnte aber auch Variablen des Typs `byte` oder `short` zugewiesen werden, da die Zahl klein genug ist, um in einen dieser beiden Typen zu passen. Ein Integer-Literal, das größer ist, als der Typ `int` aufnehmen kann, wird automatisch als Typ `long` betrachtet. Sie können auch angeben, dass ein Literal ein `long`-Integer sein soll, indem Sie den Buchstaben L (`L` oder `l`) der Zahl hinzufügen. Die folgende Anweisung behandelt z.B. den Wert 4 als `long`-Integer:

```
pennyTotal = pennyTotal + 4L;
```

Um eine negative Zahl als Literal darzustellen, stellen Sie dem Literal ein Minuszeichen (`-`) voran (z.B. `-45`).

Wenn Sie ein Integer-Literal benötigen, das das Oktal-System verwendet, dann stellen Sie der Zahl eine `0` voran. Der Oktal-Zahl `777` entspricht z.B. das Literal `0777`. Hexadezimal-Integer werden als Literalen verwendet, indem man ihnen die Zeichen `0x` voranstellt, wie z.B. `0x12` oder `0xFF`.

Das oktale und das hexadezimale Zahlensystem sind für komplexere Programmieraufgaben sehr bequem. Allerdings ist es unwahrscheinlich, dass Anfänger sie benötigen werden. Das *Oktal-System* basiert auf der 8, das heißt, dass dieses System je Stelle nur die Ziffern 0 bis 7 verwenden kann. Der Zahl 8 entspricht die Zahl 10 im Oktal-System (oder `010` als Java-Literal).

Das Hexadezimal-System verwendet hingegen als Basis die 16 und kann deshalb je Stelle 16 Ziffern verwenden. Die Buchstaben A bis F repräsentieren die letzten sechs Ziffern, sodass `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `A`, `B`, `C`, `D`, `E`, `F` die ersten 16 Zahlen sind.

Das Oktal- und das Hexadezimal-System eignen sich für bestimmte Programmieraufgaben besser als das Dezimal-System. Wenn Sie schon einmal mit HTML die Hintergrundfarbe einer Webseite festgelegt haben, dann haben Sie bereits Hexadezimal-Zahlen verwendet.

Fließkomma-Literale verwenden einen Punkt (`.`) als Dezimaltrennzeichen. Die folgende Anweisung weist einer `double`-Variablen einen Wert mit einem Literal zu:

```
double myGPA = 2.25;
```

Alle Fließkomma-Literale werden standardmäßig als `double` betrachtet und nicht als `float`. Um ein `float`-Literal anzugeben, müssen Sie den Buchstaben F (`F` oder `f`) dem Literal anhängen, wie in dem folgenden Beispiel:

```
float piValue = 3.1415927F;
```

Sie können in Fließkomma-Literalen Exponenten verwenden, indem Sie den Buchstaben `e` oder `E`, gefolgt von dem Exponenten, angeben. Dieser kann auch eine negative Zahl sein. Die folgenden Anweisungen verwenden die Exponentialschreibweise:

```
double x = 12e22;
double y = 19E-95;
```

Boolesche Literale

Die booleschen Werte `true` und `false` sind auch Literale. Dies sind die beiden einzigen Werte, die Sie bei der Wertzuweisung zu einer Variablen vom Typ `boolean` oder beim Gebrauch von booleschen Werten in einer Anweisung verwenden können.

Wenn Sie andere Sprachen wie z.B. C kennen, dann könnten Sie erwarten, dass der Wert `1` gleich `true` und der Wert `0` gleich `false` ist. Dies trifft bei Java allerdings nicht zu – Sie müssen die Werte `true` oder `false` verwenden, um boolesche Werte anzugeben. Die folgende Anweisung weist einer `boolean`-Variablen einen Wert zu:

```
boolean chosen = true;
```

Beachten Sie bitte, dass das Literal `true` nicht in Anführungszeichen eingeschlossen wird. Wenn dem so wäre, würde der Java-Compiler annehmen, dass es sich um einen String handelt.

Zeichen-Literale

Zeichen-Literale werden durch ein einzelnes Zeichen, das von einfachen Anführungszeichen umgeben ist, wie z.B. `'a'`, `'#'` und `'3'`, dargestellt. Vermutlich kennen Sie den ASCII-Zeichensatz, der 128 Zeichen umfasst, darunter Buchstaben, Ziffern, Interpunktionszeichen und andere häufige Zeichen. Java unterstützt den 16-Bit-Unicode-Standard und damit tausende weiterer Zeichen, vor allem Schriftzeichen.

Einige Zeichenliterale repräsentieren Zeichen, die nicht druckbar sind oder nicht über die Tastatur direkt eingegeben werden können. Die Tabelle 3.2 führt die Codes auf, die diese

Sonderzeichen repräsentieren, ferner den Code für Zeichen aus dem Unicode-Zeichensatz. Der Buchstabe *d* in den Oktal-, Hex- und Unicode-Escape-Codes steht für eine Zahl oder eine Hexadezimalziffer (`a-f` oder `A-F`).

Escape-Sequenz	Bedeutung
`\n`	Neue Zeile
`\t`	Tabulator (Tab)
`\b`	Rückschritt (Backspace)
`\r`	Wagenrücklauf (Carriage return)
`\f`	Seitenvorschub (Formfeed)
`\\`	Inverser Schrägstrich (Backslash)
`\'`	Einfaches Anführungszeichen
`\"`	Doppeltes Anführungszeichen
`\d`	Oktal
`\xd`	Hexadezimal
`\ud`	Unicode-Zeichen

Tabelle 3.2: Escape-Codes für Sonderzeichen

C- bzw. C++-Programmierer sollten beachten, dass Java keine Zeichencodes für `\a` (Klingel) und `\v` (vertikaler Tabulator) kennt.

String-Literale

Die letzte Literalart, die Sie in Java verwenden können, repräsentiert Strings. Ein Java-String ist ein Objekt und kein elementarer Datentyp. Strings werden auch nicht in Arrays gespeichert, wie das in Sprachen wie C der Fall ist.

Da String-Objekte echte Objekte in Java sind, stehen Methoden zur Kombination und Modifikation von Strings zur Verfügung sowie um festzustellen, ob zwei Strings denselben Wert haben.

String-Literale bestehen aus einer Reihe von Zeichen, die in doppelte Anführungszeichen eingeschlossen sind, wie das in den folgenden Anweisungen der Fall ist:

```
String quitMsg = "Are you sure you want to quit?";

String password = "swordfish";
```

Strings können die Escape-Sequenzen aus Tabelle 3.2 enthalten, wie das auch im nächsten Beispiel gezeigt wird:

```
String example = "Socrates asked, \"Hemlock is poison?\"";

System.out.println("Sincerely,\nMillard Fillmore\n");

String title = "Sams Teach Yourself Rebol While You Sleep\u2122"
```

In dem letzten Beispiel erzeugt die Unicode-Escape-Sequenz `\u2122` auf Systemen, die Unicode unterstützen, das Trademarksymbol (™).

Viele Anwender sehen keine Unicode-Zeichen, wenn sie Java-Programme ausführen. Java unterstützt zwar die Übertragung von Unicode-Zeichen, doch muss auch das System des Benutzers Unicode unterstützen, damit diese Zeichen angezeigt werden können. Unicode bietet lediglich eine Möglichkeit zur Kodierung von Zeichen für Systeme, die den Standard unterstützen. Während Java 1.0 lediglich den Teilzeichensatz *Latin* des gesamten Unicode-Zeichensatzes unterstützte, sind alle Java-Versionen ab 1.1 in der Lage, jedes beliebige Unicode-Zeichen darzustellen, das von einer Schrift auf dem Host unterstützt wird.

Mehr Informationen über Unicode finden Sie auf der Web-Site des Unicode-Konsortiums unter `http://www.unicode.org/`.

Obwohl String-Literale in einem Programm auf ähnliche Art und Weise verwendet werden wie andere Literale, werden sie hinter den Kulissen anders verarbeitet.

Wenn ein String-Literal verwendet wird, speichert Java diesen Wert als ein `String`-Objekt. Sie müssen nicht explizit ein neues Objekt erzeugen, wie das bei der Arbeit mit anderen Objekten der Fall ist. Aus diesem Grund ist der Umgang mit ihnen genauso einfach wie mit primitiven Datentypen. Strings sind in dieser Hinsicht ungewöhnlich – keiner der anderen primitiven Datentypen wird bei der Verwendung als Objekt gespeichert. Sie lernen am heutigen und morgigen Tag mehr über Strings und die `String`-Klasse.

3.5 Ausdrücke und Operatoren

Ein *Ausdruck* ist eine Anweisung, die einen Wert erzeugt. Mit die gebräuchlichsten Ausdrücke sind die mathematischen wie in dem folgenden Code-Beispiel:

```
int x = 3;
int y = x;
int z = x * y;
```

Alle drei Anweisungen sind Ausdrücke. Sie übermitteln Werte, die Variablen zugewiesen werden können. Der erste Ausdruck weist der Variable `x` das Literal `3` zu. Der zweite weist den Wert der Variablen `x` der Variablen `y` zu. Der Multiplikations-Operator `*` wird verwendet, um die Integer `x` und `y` miteinander zu multiplizieren. Der Ausdruck erzeugt das Ergebnis dieser Multiplikation, das dann im Integer `z` gespeichert wird.

Ein Ausdruck kann jede beliebige Kombination von Variablen, Literalen und Operatoren sein. Er kann auch ein Methodenaufruf sein, da eine Methode dem Objekt oder der Klasse, das bzw. die sie aufrief, einen Wert zurück übermitteln kann.

Der Wert, der von einem Ausdruck erzeugt wird, wird als *Rückgabewert* bezeichnet, wie Sie ja bereits gelernt haben. Dieser Wert kann einer Variablen zugewiesen und auf viele andere Arten in Ihren Java-Programmen verwendet werden.

Die meisten Ausdrücke in Java beinhalten Operatoren wie `*`.

Operatoren sind spezielle Symbole, die für mathematische Funktionen, bestimmte Zuweisungsanweisungen und logische Vergleiche stehen.

Arithmetische Operatoren

Es gibt in Java fünf Operatoren für die elementare Arithmetik. Diese werden in Tabelle 3.3 aufgeführt.

Operator	Bedeutung	Beispiel
`+`	Addition	`3 + 4`
`-`	Subtraktion	`5 - 7`
`*`	Multiplikation	`5 * 5`
`/`	Division	`14 / 7`
`%`	Modulo	`20 % 7`

Tabelle 3.3: Arithmetische Operatoren

Jeder Operator benötigt zwei Operanden, einen auf jeder Seite. Der Subtraktions-Operator (`-`) kann auch dazu verwendet werden, einen einzelnen Operanden zu negieren, was der Multiplikation des Operanden mit -1 entspricht.

Eine Sache, die man bei Divisionen unbedingt beachten muss, ist die Art der Zahlen, die man dividiert. Wenn Sie das Ergebnis einer Division in einem Integer speichern, wird das Ergebnis zu einer ganzen Zahl abgerundet, da der `int`-Typ nicht mit Fließkomma-Zahlen umgehen kann. Das Ergebnis des Ausdrucks `31 / 9` wäre `3`, wenn es in einem Integer gespeichert würde.

Das Ergebnis der Modulo-Operation, die den %-Operator verwendet, ist der Rest einer Division. `31 % 9` würde `4` ergeben, da bei der Division von 31 durch 9 der Rest 4 übrig bleibt.

Beachten Sie bitte, dass viele arithmetische Operationen, an denen ein Integer beteiligt ist, ein Ergebnis vom Typ `int` haben, unabhängig vom ursprünglichen Typ der Operanden. Wenn Sie mit anderen Zahlen, wie z.B. Fließkomma-Zahlen oder `long`-Integern arbeiten, sollten Sie sicherstellen, dass die Operanden denselben Typ aufweisen, den das Ergebnis haben soll.

Das Listing 3.1 zeigt ein Beispiel für einfache Arithmetik in Java.

Listing 3.1: Die Quelldatei Weather.Java

```
 1: class Weather {
 2:     public static void main(String[] arguments) {
 3:         float fah = 86;
 4:         System.out.println(fah + " degrees Fahrenheit is ...");
 5:         // Umrechnung von Fahrenheit in Celsius
 6:         // ziehe 32 ab
 7:         fah = fah - 32;
 8:         // teile das Ergebnis durch 9
 9:         fah = fah / 9;
10:         // multipliziere dieses Ergebnis mit 5
11:         fah = fah * 5;
12:         System.out.println(fah + " degrees Celsius\ n");
13:
14:         float cel = 33;
15:         System.out.println(cel + " degrees Celsius is ...");
16:         // Celsius in Fahrenheit umrechnen
17:         // multipliziere mit 9
18:         cel = cel * 9;
19:         // teile das Ergebnis durch 5
20:         cel = cel / 5;
21:         // addiere 32 zu diesem Ergebnis
22:         cel = cel + 32;
23:         System.out.println(cel + " degrees Fahrenheit");
24:     }
25: }
```

Wenn Sie diese Applikation ausführen, erhalten Sie die folgende Ausgabe:

```
86.0 degrees Fahrenheit is ...
30.0 degrees Celsius

33.0 degrees Celsius is ...
91.4 degrees Fahrenheit
```

In den Zeilen 3-12 dieser Java-Applikation wird eine Fahrenheit-Temperatur in Celsius mithilfe arithmetischer Operatoren umgewandelt:

- Zeile 3: Die Fließkomma-Variable `fah` wird mit dem Wert 86 erzeugt.
- Zeile 4: Der aktuelle Wert von `fah` wird angezeigt.
- Zeile 5: Der erste von mehreren Kommentaren, die beim Verständnis des Programms helfen sollen. Diese Kommentare werden vom Java-Kompiler ignoriert.
- Zeile 7: `fah` wird auf seinen aktuellen Wert minus 32 gesetzt.
- Zeile 9: `fah` wird auf seinen aktuellen Wert dividiert durch 9 gesetzt.
- Zeile 11: `fah` wird auf seinen aktuellen Wert multipliziert mit 5 gesetzt.
- Zeile 12: Nunmehr ist `fah` auf einen Celsius-Wert umgerechnet und wird deshalb wiederum angezeigt.

In den Zeilen 14-23 geschieht umgekehrt das Entsprechende. Eine Celsius-Temperatur wird in Fahrenheit umgerechnet.

Dieses Programm verwendet auch die Methode `System.out.println()` in diversen Anweisungen. Die Methode `System.out.println()` wird in einer Applikation verwendet, um Strings oder andere Informationen auf dem Standardausgabegerät anzuzeigen – gewöhnlich der Bildschirm.

`System.out.println()` erwartet ein einziges Argument innerhalb der Klammern: einen String. Um mehr als eine Variable oder mehr als ein Literal als Argument für `println()` zu verwenden, können Sie mit dem +-Operator diese Elemente zu einem einzigen String verknüpfen.

Sie lernen heute noch mehr über diese Verwendung des +-Operators.

Mehr über Zuweisungen

Die Zuweisung eines Wertes an eine Variable ist ein Ausdruck, da dies einen Wert erzeugt. Aus diesem Grund können Sie Zuweisungsanweisungen hintereinander schreiben, wie in folgendem Beispiel:

```
x = y = z = 7;
```

In dieser Anweisung haben am Ende alle drei Variablen den Wert 7.

Die rechte Seite eines Zuweisungausdrucks wird immer vor der Zuweisung ausgewertet. Dies macht es möglich, Ausdrücke wie den folgenden zu verwenden:

```
int x = 5;
x = x + 2;
```

In dem Ausdruck `x = x + 2` wird als Erstes `x + 2` berechnet. Das Ergebnis dieser Berechnung, 7, wird anschließend `x` zugewiesen.

Es ist eine ganz gewöhnliche Vorgehensweise in der Programmierung, den Wert einer Variablen durch einen Ausdruck zu verändern. Es gibt eine ganze Reihe von Operatoren, die ausschließlich in diesen Fällen verwendet werden.

Die Tabelle 3.4 zeigt diese Zuweisungsoperatoren und die Ausdrücke, denen sie funktional entsprechen.

Ausdruck	Bedeutung
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`

Tabelle 3.4: Zuweisungsoperatoren

Die Zuweisungsoperatoren sind funktional äquivalent mit den längeren Zuweisungsausdrücken, die sie ersetzen. Wenn allerdings eine der Seiten Ihres Zuweisungsausdrucks Teil eines komplexen Ausdrucks ist, gibt es Fälle, in denen die Operatoren nicht äquivalent sind. Wenn z.B. `x` gleich `20` ist und `y` gleich `5`, dann ergeben die beiden folgenden Anweisungen nicht dasselbe Resultat:

```
x = x / y + 5;
x /= y + 5;
```

In Zweifelsfällen sollten Sie einen Ausdruck vereinfachen, indem Sie mehrere Zuweisungsanweisungen verwenden anstatt der Zuweisungsoperatoren.

Inkrementieren und Dekrementieren

Eine weitere sehr häufig vorkommende Aufgabe ist es, zu einem Integer eins hinzuzuzählen oder eins abzuziehen. Es gibt für diese Ausdrücke spezielle Operatoren, die Inkrement- bzw. Dekrement-Operatoren genannt werden.

Eine Variable zu *inkrementieren* bedeutet, zu deren Wert eins hinzuzuzählen. Eine Variable zu *dekrementieren* bedeutet dagegen, von deren Wert eins abzuziehen.

Der Inkrement-Operator ist ++ und der Dekrement-Operator --. Diese Operatoren werden direkt nach oder direkt vor einen Variablennamen platziert, wie das im folgenden Beispiel der Fall ist:

```
int x = 7;
x = x++;
```

In diesem Beispiel inkrementiert die Anweisung `x = x++` die Variable `x` von 7 auf 8.

Die Inkrement- und Dekrement-Operatoren können vor oder nach dem Namen einer Variablen stehen. Dies beeinflusst den Wert von Ausdrücken, die diese Operatoren beinhalten.

Inkrement- und Dekrement-Operatoren werden als *Präfix-Operatoren* bezeichnet, wenn sie vor dem Namen der Variablen aufgeführt werden, und als Postfix-Operatoren, wenn sie sich hinter dem Variablennamen befinden.

In einem einfachen Ausdruck wie z.B. `standards--;` ist es für das Ergebnis unerheblich, ob Sie einen Präfix- oder einen Postfix-Operator verwenden. Wenn Inkrement- oder Dekrement-Operatoren allerdings Teil größerer Ausdrücke sind, ist die Wahl zwischen Präfix- und Postfix-Operatoren wichtig.

Nehmen Sie die beiden folgenden Ausdrücke:

```
int x, y, z;
x = 42;
y = x++;
z = ++x;
```

Diese beiden Ausdrücke erzeugen unterschiedliche Ergebnisse aufgrund des Unterschieds zwischen der Präfix- und der Postfix-Operation. Wenn Sie Postfix-Operatoren wie in `y = x++` verwenden, erhält `y` den Wert von `x`, bevor dieser um eins inkrementiert wurde. Wenn Sie dagegen Präfix-Operatoren wie in `z = ++x` verwenden, wird `x` um eins inkrementiert, bevor der Wert `z` zugewiesen wird. Das Endergebnis dieses Beispiels ist, dass `y` den Wert `42` und `z` den Wert `44` hat. `x` selbst hat auch den Wert `44`.

Für den Fall, dass Ihnen noch nicht ganz klar ist, was hier passiert, habe ich Ihnen im Folgenden noch einmal das Beispiel aufgeführt. Diesmal allerdings mit Kommentaren, die jeden einzelnen Schritt beschreiben:

```
int x, y, z; // x, y, und z werden deklariert
x = 42;      // x wird der Wert 42 zugewiesen
y = x++;     // y wird der Wert von x (42) zugewiesen, bevor x inkrementiert wird
             // anschließend wird x auf 43 inkrementiert
z = ++x;     // x wird auf 44 inkrementiert, und z wird der Wert von x zugewiesen
```

Wie auch Zuweisungsoperatoren können Inkrement- und Dekrement-Operatoren unerwünschte Ergebnisse erzeugen, wenn diese in extrem komplexen Ausdrücken verwendet werden. Das Konzept »x wird y zugewiesen, bevor x inkrementiert wird« stimmt nicht ganz, da Java alles auf der rechten Seite eines Ausdrucks auswertet, bevor das Ergebnis der linken Seite zugewiesen wird. Java speichert einige Werte, bevor es einen Ausdruck verarbeitet, damit die Postfix-Notation wie in diesem Abschnitt beschrieben funktioniert. Wenn Sie nicht die Ergebnisse erhalten, die Sie von einem komplexen Ausdruck mit Präfix- und Postfix-Operatoren erwarten, versuchen Sie den Ausdruck in mehrere Ausdrücke aufzuspalten, um ihn zu vereinfachen.

Vergleiche

Java besitzt diverse Operatoren, die bei Vergleichen von Variablen mit Variablen und Variablen mit Literalen oder anderen Informationsarten verwendet werden.

Diese Operatoren werden in Ausdrücken verwendet, die boolesche Werte (`true` oder `false`) zurückgeben. Dies ist abhängig davon, ob der Vergleich aufgeht oder nicht. Die Tabelle 3.5 zeigt die einzelnen Vergleichsoperatoren.

Operator	Bedeutung	Beispiel
==	Gleich	`x == 3`
!=	Ungleich	`x != 3`
<	Kleiner als	`x < 3`
>	Größer als	`x > 3`
<=	Kleiner als oder gleich	`x <= 3`
>=	Größer als oder gleich	`x >= 3`

Tabelle 3.5: Vergleichsoperatoren

Die folgenden Beispiele zeigen die Verwendung eines Vergleichsoperators:

```
boolean hip;
int age = 33;
hip = age < 25;
```

Der Ausdruck `age < 25` ergibt entweder `true` oder `false`. Dies hängt von dem Wert des Integers `age` ab. Da `age` in diesem Beispiel den Wert 33 hat (was nicht kleiner als 25 ist), wird `hip` der boolesche Wert `false` zugewiesen.

Logische Operatoren

Ausdrücke, die boolesche Werte ergeben (also z.B. Vergleiche), können kombiniert werden, um komplexere Ausdrücke zu formen. Dies geschieht mit logischen Operatoren. Diese Operatoren werden für die logischen Verknüpfungen `AND` (UND), `OR` (ODER), `XOR` (exklusives ODER) und `NOT` (logisches NICHT) verwendet.

Für `AND`-Verknüpfungen werden die Operatoren `&` und `&&` verwendet. Wenn zwei boolesche Ausdrücke mit `&` oder `&&` verknüpft werden, ergibt der kombinierte Ausdruck nur dann `true`, wenn beide Teilausdrücke `true` sind.

Nehmen Sie das folgende Beispiel, das direkt aus dem Film *Harold & Maude* stammt:

```
boolean unusual = (age < 21) & (girlfriendAge > 78);
```

Dieser Ausdruck kombiniert zwei Vergleichsausdrücke: `age < 21` und `girlfriendAge > 78`. Wenn beide Ausdrücke `true` ergeben, wird der Variablen `unusual` der Wert `true` zugewiesen, in allen anderen Fällen der Wert `false`.

Der Unterschied zwischen `&` und `&&` liegt in der Menge der Arbeit, die sich Java mit kombinierten Ausdrücken macht. Wenn `&` verwendet wird, werden immer die Ausdrücke auf beiden Seiten des `&` ausgewertet. Wenn dagegen `&&` verwendet wird und die linke Seite von `&&` `false` ergibt, wird der Ausdruck auf der rechten Seite nicht ausgewertet.

Für `OR`-Verknüpfungen werden die logischen Operatoren `|` und `||` verwendet. Kombinierte Ausdrücke mit diesen Operatoren geben `true` zurück, wenn einer der beiden booleschen Ausdrücke `true` ergibt.

Nehmen Sie das von *Harold & Maude* inspirierte Beispiel:

```
boolean unusual = (grimThoughts > 10) || (girlfriendAge > 78);
```

Dieser Ausdruck verknüpft zwei Vergleichsausdrücke: `grimThoughts > 10` und `girlfriendAge > 78`. Wenn einer dieser Ausdrücke `true` ergibt, wird der Variablen `unusual` `true` zugewiesen. Nur wenn beide Ausdrücke `false` ergeben, wird `unusual` `false` zugewiesen.

In diesem Beispiel wurde `||` , nicht `|` verwendet. Daher wird `unusual` auf `true` gesetzt, wenn `grimThoughts > 10` `true` ergibt, und der zweite Ausdruck wird nicht ausgewertet.

Für die `XOR`-Operation gibt es nur einen Operator, `^`. Ausdrücke mit diesem Operator ergeben nur dann `true`, wenn die booleschen Ausdrücke, die damit verknüpft werden, entgegengesetzte Werte haben. Wenn beide `true` oder beide `false` sind, ergibt der `^`-Ausdruck `false`.

Die `NOT`-Verknüpfung verwendet den logischen Operator `!`, gefolgt von einem einzelnen Ausdruck. Diese Verknüpfung kehrt den Wert eines booleschen Ausdrucks um, wie ein `-` (Minus) ein negatives oder positives Vorzeichen bei einer Zahl umkehrt.

Wenn z. B. der Ausdruck `age < 30` `true` zurückgibt, dann gibt `!(age < 30)` `false` zurück.

Diese logischen Operatoren erscheinen zunächst einmal überhaupt nicht logisch, wenn man das erste Mal auf sie trifft. Sie werden in den folgenden Kapiteln, besonders an Tag 5, viel Gelegenheit erhalten, mit ihnen zu arbeiten.

Operatorpräzedenz

Sobald mehr als ein Operator in einem Ausdruck benutzt wird, verwendet Java eine definierte Präzedenz, um die Reihenfolge festzulegen, mit der die Operatoren ausgewertet werden. In vielen Fällen legt diese Präzedenz den Gesamtwert des Ausdrucks fest.

Nehmen Sie den folgenden Ausdruck als Beispiel:

```
y = 6 + 4 / 2;
```

Die Variable `y` erhält den Wert 5 oder den Wert 8, abhängig davon, welche arithmetische Operation zuerst ausgeführt wird. Wenn der Ausdruck `6 + 4` als Erstes ausgewertet wird, hat `y` den Wert `5`. Andernfalls erhält `y` den Wert `8`.

Im allgemeinen gilt folgende Reihenfolge, wobei der erste Eintrag der Liste die höchste Priorität besitzt:

- Inkrement- und Dekrement-Operationen
- Arithmetische Operationen
- Vergleiche
- Logische Operationen
- Zuweisungsausdrücke

Wenn zwei Operationen dieselbe Präzedenz besitzen, wird die auf der linken Seite des Ausdruck vor der auf der rechten Seite ausgewertet. Tabelle 3.6 zeigt die Präzedenz der einzelnen Operatoren. Operatoren weiter oben in der Tabelle werden vor denen darunter ausgewertet.

Operator	Anmerkung
`. [] ()`	Klammern (`()`) werden verwendet, um Ausdrücke zu gruppieren. Der Punkt (`.`) dient für den Zugriff auf Methoden und Variablen in Objekten und Klassen (wird morgen behandelt). Eckige Klammern (`[]`) kommen bei Arrays zum Einsatz (wird später in dieser Woche besprochen).
`++ -- ! ~ instanceof`	Der Operator `instanceof` gibt `true` oder `false` zurück, abhängig davon, ob ein Objekt eine Instanz der angegebenen Klasse oder deren Subklassen ist (wird morgen besprochen).

Operator	Anmerkung
`new (Typ)Ausdruck`	Mit dem `new`-Operator werden neue Instanzen von Klassen erzeugt. Die Klammern (`()`) dienen in diesem Fall dem Casting eines Wertes in einen anderen Typ (wird morgen behandelt).
`* / %`	Multiplikation, Division, Modulo
`+ -`	Addition, Subtraktion
`<< >> >>>`	Bitweiser Links- und Rechts-Shift
`< > <= >=`	Relationale Vergleiche
`== !=`	Gleichheit
`&`	AND
`^`	XOR
`|`	OR
`&&`	Logisches AND
`||`	Logisches OR
`? :`	Kurzform für `if...then...else` (wird an Tag 5 behandelt)
`= += -= *= /= %= ^=`	Verschiedene Zuweisungen
`&= |= <<= >>= >>>=`	Weitere Zuweisungen

Tabelle 3.6: Operatorpräzedenz

Lassen Sie uns nun zu dem Ausdruck `y = 6 + 4 / 2` zurückkehren. Tabelle 3.6 zeigt, dass eine Division vor einer Addition ausgewertet wird, sodass `y` den Wert 8 haben wird.

Um die Reihenfolge zu ändern, in der Ausdrücke ausgewertet werden, schließen Sie den Ausdruck, der zuerst ausgeführt werden soll, in Klammern ein. Sie können Klammernebenen ineinander verschachteln, um sicherzustellen, dass die Ausdrücke in der gewünschten Reihenfolge ausgeführt werden – der Ausdruck in der innersten Klammer wird als Erstes ausgeführt.

Der folgende Ausdruck ergibt den Wert 5:

```
y = (6 + 4) / 2
```

Das Ergebnis ist hier 5, da `6 + 4` berechnet wird, ehe das Ergebnis durch `2` geteilt wird.

Klammern können auch nützlich sein, um die Lesbarkeit eines Ausdrucks zu erhöhen. Wenn Ihnen die Präzedenz eines Ausdrucks nicht sofort klar ist, dann können Sie den Ausdruck leichter verständlich machen, indem Sie Klammern hinzufügen und so die gewünschte Präzedenz erzwingen.

3.6 String-Arithmetik

Wie zuvor schon erwähnt, führt der Operator + ein Doppelleben außerhalb der Welt der Mathematik. Er kann dazu verwendet werden, zwei oder mehr Strings miteinander zu verketten.

Verketten bedeutet das Aneinanderhängen zweier Dinge. Aus unbekannten Gründen wurde »verketten« zum Mot Juste für das Kombinieren zweier Strings und – siegte damit über aussichtsreiche Kandidaten wie ankleben, anhängen, kombinieren, verknüpfen usw.

In vielen Beispielen haben Sie Anweisungen wie die Folgende gesehen:

```
String firstName = "Raymond";
System.out.println("Everybody loves " + firstName);
```

Als Ergebnis der beiden Zeilen wird auf dem Bildschirm Folgendes ausgegeben:

```
Everybody loves Raymond
```

Der Operator + kombiniert Strings, andere Objekte und Variablen zu einem einzigen String. In dem vorangegangenen Beispiel wird das Literal `Everybody loves` mit dem Wert des `String`-Objektes `firstName` verkettet.

Der Umgang mit dem Verkettungsoperator ist in Java einfach, da er alle Variablentypen und Objektwerte wie Strings behandelt. Sobald ein Teil einer Verkettung ein `String` oder ein `String`-Literal ist, werden alle Elemente der Operation wie Strings behandelt.

Zum Beispiel:

```
System.out.println(4 + " score and " + 7 + " years ago.");
```

Diese Zeile erzeugt die Ausgabe `4 score and 7 years ago`, als ob die Integer-Literale 4 und 7 Strings wären.

Es gibt auch eine Kurzform, den Operator +=, um etwas an das Ende eines Strings anzuhängen. Nehmen Sie z.B. folgenden Ausdruck:

```
myName += " Jr.";
```

Dieser Ausdruck ist gleichwertig mit:

```
myName = myName + " Jr.";
```

In diesem Beispiel wird dem Wert von `myName` (z.B. `Efrem Zimbalist`) am Ende der String `Jr.` hinzugefügt (was `Efrem Zimbalist Jr.` ergäbe).

3.7 Zusammenfassung

Jeder, der eine Matrjoschka-Puppe zerlegt, wird ein bisschen enttäuscht sein, wenn er die kleinste Puppe der Gruppe erreicht. Idealerweise sollten es die Fortschritte in der Mikrotechnik den russischen Handwerkern ermöglichen, immer kleinere Puppen zu erzeugen, bis einer die Schwelle zum Subatomaren erreicht und als Gewinner feststeht.

Sie haben heute die kleinste Puppe von Java erreicht, das sollte aber kein Grund zur Enttäuschung sein. Anweisungen und Ausdrücke ermöglichen Ihnen, mit der Erstellung effektiver Methoden zu beginnen, die wiederum effektive Objekte und Klassen ermöglichen.

Heute haben Sie gelernt, Variablen zu erstellen und diesen Werte zuzuweisen. Dazu haben Sie Literale, die numerische Werte, Zeichen und String-Werte repräsentieren, verwendet und mit Operatoren gearbeitet. Morgen werden Sie dieses Wissen verwenden, um Objekte für Javaprogramme zu erstellen.

Um den heutigen Stoff zusammenzufassen, listet Tabelle 3.7 die Operatoren auf, die Sie heute kennen gelernt haben.

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo
<	Kleiner als
>	Größer als
<=	Kleiner als oder gleich
>=	Größer als oder gleich
==	Gleich
!=	Nicht gleich
&&	Logisches AND
||	Logisches OR
!	Logisches NOT
&	AND
|	OR
^	XOR
=	Zuweisung

Operator	Bedeutung
++	Inkrement
--	Dekrement
+=	Addieren und zuweisen
-=	Subtrahieren und zuweisen
*=	Multiplizieren und zuweisen
/=	Dividieren und zuweisen
%=	Modulo und zuweisen

Tabelle 3.7: Zusammenfassung der Operatoren

3.8 Fragen und Antworten

F *Was passiert, wenn man einen Integer-Wert einer Variablen zuweist und der Wert zu groß für die Variable ist?*

A Es wäre nahe liegend zu vermuten, dass die Variable in den nächstgrößeren Wert konvertiert wird. Doch dies geschieht nicht. Stattdessen tritt ein Überlauf auf. Dies ist eine Situation, in der eine Zahl von einer Extremgröße in eine andere umkippt. Ein Beispiel für einen Überlauf wäre eine `byte`-Variable, die von dem Wert 127 (akzeptabler Wert) zu dem Wert 128 (nicht akzeptabel) springt. Der Wert der Variablen würde zu dem kleinsten zulässigen Wert (-128) umkippen und von dort aus fortfahren hochzuzählen. Überläufe können Sie in Programmen nicht auf die leichte Schulter nehmen. Aus diesem Grund sollten Sie Ihre Variablen mit reichlich Spielraum ausstatten.

F *Warum gibt es in Java die Kurzformen für arithmetische Operationen und Zuweisungen? Es ist wirklich schwierig, Quelltexte mit diesen Operatoren zu lesen.*

A Die Syntax von Java basiert auf C++, das wiederum auf C basiert (schon wieder eine russische Puppe). C ist eine Sprache für Experten, die Programmiereffizienz über die Lesbarkeit des Quellcodes stellt. Die Zuweisungsoperatoren sind eine Hinterlassenschaft dieser Priorität beim Design von C. Es besteht keine Notwendigkeit, sie in einem Programm zu verwenden, da man sie vollständig ersetzen kann. Wenn Sie es vorziehen, können Sie auf diese Operatoren in Ihren eigenen Programmen ganz verzichten.

3.9 Quiz

Überprüfen Sie den heutigen Stoff, indem Sie dieses Quiz mit drei Fragen beantworten.

Fragen

1. Welche der drei folgenden Antworten ist ein gültiger Wert einer `boolean`-Variable?

   (a) `"false"`

   (b) `false`

   (c) `10`

2. Welche dieser Konventionen wird bei der Benennung von Variablen in Java nicht benutzt?

   (a) Jedes Wort nach dem ersten in einer Variable beginnt mit einem Großbuchstaben.

   (b) Der erste Buchstabe des Variablennamens ist klein.

   (c) Alle Buchstaben sind Großbuchstaben.

3. Welcher der drei folgenden Datentypen beinhaltet Zahlen zwischen -32.768 bis 32.767?

   (a) `char`

   (b) `byte`

   (c) `short`

Antworten

1. b. In Java kann ein `boolean` nur `true` oder `false` sein. Wenn Sie den Wert in Anführungszeichen setzen, dann wird er als String und nicht als einer der beiden `boolean`-Werte betrachtet.
2. c. Die Namen von Konstanten werden komplett in Großbuchstaben geschrieben, damit sie unter anderen Variablen hervorstechen.
3. c.

3.10 Übungen

Um Ihr Wissen über die heute behandelten Themen zu vertiefen, können Sie sich an folgenden Übungen versuchen:

- Erstellen Sie ein Programm, das berechnet, was aus einer Investition von 14.000 Dollar würde, die im ersten Jahr um 40% wächst, im zweiten 1.500 Dollar Wert verliert und im dritten Jahr wiederum um 12% wächst.
- Schreiben Sie ein Programm, das zwei Zahlen anzeigt und mithilfe der Operatoren `/` und `%` das Ergebnis und den Rest ihrer Division darstellt. Verwenden Sie den Escape-Code `\t` , um Ergebnis und Rest in Ihrer Ausgabe zu trennen.

Soweit einschlägig, finden Sie die Lösungen zu den Übungen auf der Webseite zum Buch: `http://www.java21days.com`.