Das Java-ABC

Woche 1

Wie Sie bereits gelernt haben, besteht ein Java-Programm aus Klassen und Objekten, diese sind aus Methoden und Variablen aufgebaut. Methoden wiederum bestehen aus Anweisungen und Ausdrücken, in denen sich Operatoren finden.

An dieser Stelle könnte sich bei Ihnen die Befürchtung breit machen, daß Java wie die verschachtelten russischen Puppen, die Matryoska ist. Es scheint so, das jede dieser Puppen eine weitere kleinere Puppe beinhaltet, die genauso kompliziert und detailreich wie die größere Freundin ist.

Entspannen Sie sich ..., 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 erreichen können.

Die folgenden Themen werden heute behandelt:

- Java-Anweisungen und -Ausdrücke
- Variablen und Datentypen
- 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. Wenn nötig, werden technische Hinweise wie dieser spezifische Unterschiede zwischen Java und den anderen Sprachen erklären.

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 age = 30;
import java.awt.dnd;
System.out.println("You're not the boss of me!");
player.score = 41367;
```

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 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 einer Zeile anordnen, und das Programm wird trotzdem erfolgreich kompiliert werden, wie das im folgenden der Fall ist:

```
j.color = "lemon yellow"; j.hungry = false;
```

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

Variablen und Datentypen

In der Jabberwock-Applikation, die Sie gestern erstellt haben, haben Sie Variablen verwendet, um bestimmte Informationen ablegen zu können.

Variablen sind Orte, an denen, während ein Programm läuft, Informationen gespeichert werden können. Der Wert der Variablen kann unter deren Namen von jedem Punkt im Programm aus geändert werden.

Um eine Variable zu erstellen, müssen Sie dieser 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 verwendet oder sogar in kleineren Blockanweisungen innerhalb von Methoden. Diese Variablen können nur verwendet werden, während die Methode oder der Block von dem Java-Interpreter ausgeführt wird. Die Existenz dieser Variablen endet anschließend.

Obwohl alle diese Variablen fast auf dieselbe Art erzeugt werden, werden Klassen- und Instanzvariablen anders verwendet als lokale Variablen. Sie lernen heute mehr über lokale Variablen. 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 Java-Programm verwenden können, müssen Sie die Variable erst einmal erzeugen, indem Sie deren 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. Im Anschluß an diesen Absatz sehen Sie einige Beispiele für Variablendeklarationen:

```
int highScore;
String username;
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.

Lokale Variablen können an jeder Stelle in einer Methode, wie jede andere Java-Anweisung auch, deklariert werden. Bevor sie allerdings verwendet werden können, müssen sie deklariert werden. Normalerweise werden Variablendeklarationen direkt im Anschluß an die Anweisung, die eine Methode deklariert, plaziert.

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. Dazu trennen Sie die einzelnen Variablennamen mit Kommas. Die folgende Anweisung erzeugt drei `String`-Variablen mit den Namen `street`, `city` und `state`:

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

Variablen kann bei deren 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 = 90210;
String name = "Brandon";
boolean cheatedOnKelly = true;
int age = 28, height = 70, weight = 140;
```

Wie die letzte Anweisung bereits andeutet, können Sie mehreren Variablen desselben Typs Werte zuweisen, indem Sie sie mit Kommas 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 diese aufnehmen sollen:

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

Variablen benennen

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

Java unterstützt auch den Unicode-Zeichensatz, der den Standardzeichensatz plus Tausende anderer Zeichen beinhaltet, um internationale Alphabete zu repräsentieren. Zeichen mit Akzenten und andere Symbole können in Variablennamen verwendet werden, solange diese über eine Unicode-Nummer oberhalb von `00C0` verfügen.

Wenn Sie eine Variable benennen und diese in einem Programm verwenden, dann 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` ist keine `ROSE`.

In den Programmen in diesem Buch, und auch außerhalb, werden Variablen oft mit aussagekräftigen Namen versehen, die aus mehreren miteinander verbundenen Wörtern bestehen. Um einzelne Worte 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 entsprechen diesen Regeln:

```
Button loadFile;
int areaCode;
boolean playerSetNewHighScore;
```

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, Zeilen 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 auch 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 sind vorzeichenbehaftet, 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.

Einen anderen Typ Zahlen, die gespeichert werden können, stellen die Fließkomma-Zahlen dar, die die Typen `float` oder `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.

Der Typ `char` wird für einzelne Zeichen, wie z.B. Buchstaben, Ziffern, Interpunktionszeichen und andere Symbole verwendet, da Java den Unicode-Zeichensatz unterstützt.

Der letzte der acht elementaren Datentypen ist `boolean`. Wie Sie bereits gelernt haben, speichern boolesche Variablen unter 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, so daß Sie diese nicht im Austausch füreinander verwenden 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 = "Walsh";
Color hair;
Jabberwock firstMonster;
```

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, `Jabberwock firstMonster;` erzeugt eine Variable mit dem Namen `firstMonster`, die sich auf ein `Jabberwock`-Objekt bezieht.

Einer Variablen den Typ einer Superklasse zuzuweisen, kann sinnvoll sein, wenn die Variable sich auf ein Objekt einer von vielen Subklassen beziehen kann. Nehmen Sie z.B. eine Klassenhierarchie mit der Superklasse `Frucht` und den drei Subklassen `Apfel`, `Birne` und `Erdbeere`. Wenn Sie eine Variable des Typs `Frucht` mit dem Namen `meineLieblingsFrucht` erzeugen, kann diese auf ein `Apfel`-, ein `Birne`- oder ein `Erdbeere`-Objekt zu verweisen.

Wenn Sie eine Variable vom Typ `Object` deklarieren, heißt das, daß diese Variable jedes beliebige Objekt aufnehmen kann.

In Java gibt es nichts, was der `typedef`-Anweisung aus C und C++ entspricht. 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. Im Anschluß an diesen Absatz finden Sie zwei Beispiele für Zuweisungsanweisungen:

```
idCode = 8675309;
snappyDresser = false;
```

Kommentare

Eine der wichtigsten Methoden, 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 verschiedene Arten von Kommentaren, die Sie in Java-Programmen nach Ihrem eigenen Ermessen verwenden können.

Die erste Methode, einen Kommentar in einem 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, angefangen bei // bis zum Ende der Zeile, von dem 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 wurde spät nachts unter dem Einfluß von Antihistamin-
Medikamenten, deren Verfallsdatum abgelaufen war, und Soda-
Wasser aus dem Supermarkt geschrieben. Ich übernehme keine Garantie, weder explizi
t noch implizit, dafür, daß dieses Programm einen sinnvollen wie auch immer gearte
ten Zweck erfüllt. */
```

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 deren `public`-Methoden interpretiert.

Diese Art von Kommentar kann von Tools, wie `javadoc`, das sich in dem JDK befindet, gelesen werden. Das Programm `javadoc` verwendet diese offiziellen Kommentare, um eine Reihe von Webseiten zu erzeugen, die das Programm, seine Klassenhierarchie und die Methoden dokumentieren.

Die gesamte offizielle Dokumentation der Klassenbibliothek von Java ist das Ergebnis von `javadoc`-Kommentaren. Sie können sich die Dokumentation von Java 1.2 im Web unter der folgenden Adresse ansehen:

```
http://java.sun.com:80/products/jdk/1.2/docs
```

Literale

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

Literale sind Zahlen, Text 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 = 1998;
```

Das Literal ist `1998`, da es direkt den Integer-Wert 1998 darstellt. Zahlen, Zeichen und Strings sind alles Beispiele für Literale.

Obwohl die Bedeutung und Anwendung von in den meisten Fällen Literalen 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 sehr viele Zahlen-Literale. Die Zahl `4` ist z.B. ein Integer-Literal des Variablentyps `int`. Es können aber auch Variablen des Typs `byte` und `short` zugewiesen werden, da die Zahl klein genug ist, um in beide Typen zu passen. Ein Integer-

Literal, das größer ist, als was der Typ `int` aufnehmen kann, wird automatisch als Typ `long` verarbeitet. 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 speichert z.B. den Wert 4 in einem `long`-Integer:

```
long 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`. Bei Hexadezimal-Integern werden den Literalen die Zeichen `0x` vorangestellt, wie z.B. `0x12` oder `0xFF`.

Das oktale und das hexadezimale Zahlensystem sind für viele fortgeschrittenere Programmieraufgaben sehr bequem. Allerdings ist es unwahrscheinlich, daß Anfänger diese benötigen werden. Das Oktal-System hat als Basis die 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 die 16 Ziffern verwenden. Die Buchstaben A bis F repräsentieren die letzten sechs Ziffern, so daß 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 Hexadezimal-Zahlen verwendet.

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

```
double myGPA = 2.25;
```

Alle Fließkomma-Literale werden standardmäßig als `double` verarbeitet 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 einzigen beiden Werte, die Sie bei der Wertzuweisung zu einer Variablen vom Typ `boolean` oder bei der Anwendung von booleschen Werten in einer Anweisung oder anderswo verwenden können.

Wenn Sie andere Sprachen, wie z.B. C, verwendet haben, dann erwarten Sie eventuell, daß der Wert `1` `true` und der Wert `0` `false` entspricht. Dies trifft auf 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 toThineOwnSelf = true;
```

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

Zeichenliterale

Zeichenliterale werden durch ein einzelnes Zeichen, das von einfachen Anführungszeichen umgeben ist, wie z.B. `'a'`, `'#'` und `'3'`, dargestellt. Sie sind eventuell mit dem ASCII-Zeichensatz vertraut, der 128 Zeichen, darunter Buchstaben, Ziffern, Interpunktionszeichen und andere Zeichen, die in bezug auf den Computer nützlich sind, beinhaltet. Java unterstützt über den 16-Bit-Unicode-Standard Tausende weiterer Zeichen.

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 wie auch Zeichen aus dem Unicode-Zeichensatz repräsentieren. 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)
`\'`	Einfache Anführungszeichen
`\"`	Doppelte Anführungszeichen
`\d`	Oktal
`\xd`	Hexadezimal
`\ud`	Unicode-Zeichen

Tabelle 3.2: Escape-Codes für Sonderzeichen

C/C++-Programmierer sollten beachten, daß Java keine Codes für `\a` (Bell) und `\v` (vertikaler Tabulator) beinhaltet.

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 zwei doppelte Anführungszeichen eingeschlossen sind, wie das in den folgenden Anweisungen der Fall ist:

```
String coAuthor = "Laura Lemay, killer of trees";
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("Bob Kemp\nOne on One Sports\n2 a.m. to 6 a.m.");
String title = "Teach Yourself Java in a 3-Day Weekend\u2122"
```

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

Die meisten Windows-95-Anwender in englischsprachigen Ländern werden wahrscheinlich keine Unicode-Zeichen sehen, wenn sie Java-Programme ausführen. Obwohl Java die Übertragung von Unicode-Zeichen unterstützt, muß auch das System des Benutzers Unicode unterstützen, damit diese Zeichen angezeigt werden können. Die Unterstützung von Unicode bietet lediglich eine Möglichkeit zur Kodierung von Zeichen für Systeme, die den Standard unterstützen. Java 1.0.2 unterstützte lediglich den Teilzeichensatz *Latin* des gesamten Unicode-Zeichensatzes. Java 1.1 und 1.2 sind 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 Website 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 diesen 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.

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 = 4;
int z = x * y;
```

Die letzte Anweisung in diesem Beispiel ist ein Ausdruck. Der Multiplikations-Operator * wird verwendet, um die Integer `x` und `y` miteinander zu multiplizieren. Der Ausdruck erzeugt das Ergebnis dieser Multiplikation. Dieses Ergebnis wird in dem Integer `z` gespeichert.

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 Zuweisungsarten 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 gerundet, da der `int`-Typ keine Fließkomma-Zahlen aufnehmen kann. Das Ergebnis des Ausdrucks `31 / 9` wäre 3, wenn es in einem Integer gespeichert wird.

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 übrigbleibt.

Beachten Sie bitte, daß die meisten arithmetischen Operationen, an denen ein Integer beteiligt ist, ein Ergebnis vom Typ `int` haben, unabhängig von dem Originaltyp 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 AmoebaMath.Java

```
 1: class AmoebaMath {
 2:     public static void main (String arguments[]) {
 3:         int x = 6;
 4:         short y = 4;
 5:         float a = .12f;
 6:
 7:         System.out.println("You start with " + x + " pet amoebas.");
 8:         System.out.println("\tTwo get married and their spouses move in.");
 9:         x = x + 2;
10:         System.out.println("You now have " + x);
11:
12:         System.out.println("\tMitosis occurs, doubling the number of
          amoebas.");
13:         x = x * 2;
14:         System.out.println("You now have " + x);
15:
16:         System.out.println("\tThere's a fight. " + y + " amoebas move out.");
17:         x = x - y;
18:         System.out.println("You now have " + x);
19:
20:         System.out.println("\tParamecia attack! You lose one-third of the
          colony.");
21:         x = x - (x / 3);
22:         System.out.println("You end up with " + x + " pet amoebas.");
23:         System.out.println("Daily upkeep cost per amoeba: $" + a);
24:         System.out.println("Total daily cost: $" + (a * x));
25:     }
26: }
```

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

```
You start with 6 pet amoebas.
    Two get married and their spouses move in.
You now have 8
    Mitosis occurs, doubling the number of amoebas.
You now have 16
    There's a fight. 4 amoebas move out.
You now have 12
    Paramecia attack! You lose one-third of the colony.
You end up with 8 pet amoebas.
Daily upkeep cost per amoeba: $0.12
Total daily cost: $0.96
```

In dieser einfachen Java-Applikation werden in den Zeilen 3–5 drei Variablen mit Initialisierungswerten erzeugt: der Integer `x`, der `short`-Integer `y` und die Fließkomma-Zahl `a`. Da der Standardtyp für Fließkomma-Zahlen `double` ist, wird ein `f` an das Literal `.12` angehängt. Dies sagt aus, daß es sich um einen `float` handelt.

Der Rest des Programms verwendet arithmetische Operatoren, um die Population der Amöben-Kolonie (keine Sorge: während ich dieses Kapitel schrieb, wurde keine Amöbe verletzt) zu berechnen.

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 ein Literal als Argument für `println()` zu verwenden, können Sie mit dem +-Operator diese Elemente zu einem einzigen String verknüpfen.

Sie lernen über diese Verwendung des +-Operators später mehr.

Mehr über die Zuweisung

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 dem folgenden 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 von der Funktion her entsprechen.

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

Tabelle 3.4: Zuweisungsoperatoren

Die Zuweisungsoperatoren sind von der Funktionsweise her 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. 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-Ooperator `--`. Diese Operatoren werden direkt nach oder direkt vor einen Variablennamen plaziert, 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, funktionieren kann. 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, Variablen mit Literalen oder anderen Informationsarten in einem Programm 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 = 31;
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 31 hat (was nicht kleiner als 25 ist), wird `hip` der boolesche Wert `false` zugewiesen.

Logische Operatoren

Ausdrücke, die boolesche Werte ergeben, wie das bei Vergleichen der Fall ist, können kombiniert werden, um komplexere Ausdrücke zu formen. Dies erreicht man 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 `|` oder `||` 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 = (suicideAttempts > 10) || (girlfriendAge > 78);
```

Dieser Ausdruck verknüpft zwei Vergleichsausdrücke: `suicideAttempts > 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.

Beachten Sie bitte, daß hier `||` anstelle von `|` verwendet wurde. Aus diesem Grund wird `unusual true` zugewiesen, wenn `suicideAttempts > 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 auch 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 völlig unlogisch, wenn man das erste Mal auf sie trifft. Sie werden in den folgenden Kapiteln, besonders an Tag 5, viele Gelegenheiten erhalten, mit diesen zu arbeiten.

Operatorpräzedenz

Sobald mehr als ein Operator in einem Ausdruck verwendet 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 gibt es folgende Reihenfolge, wobei der erste Eintrag der Liste die höchste Priorität besitzt:

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

Wenn zwei Operationen dieselbe Präzedenz besitzen, wird die auf der linken Seite in dem aktuellen 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 (wird morgen besprochen) ist.
`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).

Tabelle 3.6: Operatorpräzedenz

Operator	Anmerkung
`* / %`	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 3.7 zeigt, daß eine Division vor einer Addition ausgewertet wird, so daß `y` den Wert 8 haben wird.

Um die Reihenfolge, in der Ausdrücke ausgewertet werden, zu ändern, 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 innerste geklammerte Ausdruck wird als erstes ausgeführt.

Der folgende Ausdruck ergibt den Wert 5:

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

Das Ergebnis ist hier 5, da `6 + 4` hier berechnet wird, bevor 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 kann das Hinzufügen von Klammern, um die geforderte Präzedenz zu erzwingen, den Ausdruck leichter verständlich machen.

String-Arithmetik

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

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.";
```

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

Zusammenfassung

Jeder, der eine Matryoska-Puppe zerlegt, wird ein bißchen enttäuscht sein, wenn er die kleinste Puppe der Gruppe erreicht. Idealerweise sollten es die Fortschritte in der Mikrotechnik den Künstlern 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, es sollte aber kein Grund zur Enttäuschung sein. Anweisungen und Ausdrücke ermöglichen es Ihnen, mit der Erstellung effektiver Methoden zu beginnen, die wiederum effektive Objekte und Klasse ermöglichen.

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

Um den heutigen Stoff zusammenzufassen, listet Tabelle 3.7 die Operatoren auf, die Sie heute kennengelernt 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
++	Inkrement
-	Dekrement
+=	Addieren und zuweisen
-=	Subtrahieren und zuweisen

Tabelle 3.7: Zusammenfassung der Operatoren

Operator	Bedeutung
*=	Multiplizieren und zuweisen
/=	Dividieren und zuweisen
%=	Modulo und zuweisen

Tabelle 3.7: Zusammenfassung der Operatoren

Fragen und Antworten

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

A *Logischerweise werden Sie denken, daß die Variable in den nächstgrößeren Wert konvertiert wird. Allerdings ist dies genau das, was nicht passiert. Statt dessen 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 Ihren Variablen reichlich Lebensraum für deren Datentyp geben.*

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 Mächtigkeit bei der Programmierung über die Lesbarkeit des Quellcodes stellt. Die Zuweisungsoperatoren sind eine Hinterlassenschaft dieser Priorität beim Design von C. Es besteht keine Notwendigkeit, diese in einem Programm zu verwenden, da es effektive Ersetzungsmöglichkeiten gibt. Wenn Sie es vorziehen, können Sie auf diese Operatoren ganz verzichten.*