Münzwerte

```
public class Muenzen1
{
    public static void main(String[] args) {
        int zehnerl = 8; // Anzahl 10 ct Muenzen
        int zwanzgerl = 4; // Anzahl 20 ct Muenzen
        int fuchzgerl = 3; // Anzahl 50 ct Muenzen

        double gesamt = zehnerl * 0.10 +
            zwanzgerl * 0.20 + fuchzgerl * 0.50;

        System.out.print("Gesamtwert = ");
        System.out.println(gesamt);
    }
}
```


Alles was nach dem `//` kommt ist Kommentar.

Zwei verschiedene Arten von Zahlen:

- Integers (ganze Zahlen). In Java `int`. Werte: `8`, `4`, `3`.
- Doubles (Fließkommazahlen mit doppelter Präzision). In Java `double`. Werte: `0.10`, `0.20`, `0.50`, aber auch `3E9` ($=3 \cdot 10^9$).

Man sollte Integers für ganzzahlige Größen verwenden, z.B. Anzahlen, Pixelkoordinaten.

Vorteile: Weniger Speicherplatz, Genauigkeit, vermeidet, „Äpfel mit Birnen zu verwechseln“.

Variablen

Das Statement

`int zehnerl = 8;`

deklariert eine ganzzahlige Variable des Wertes 8.

Vorteil hier: bessere Dokumentation.

Semantisch äquivalent:

`System.out.println(8 * 0.10 + 4 * 0.20 + 3 * 0.50);`

aber schlechter lesbar.

Wertzuweisung

Man kann einer Variablen neue Werte zuweisen:

```
zwanzgerl = 5;
int zw = zwanzgerl;
zw = 6;
System.out.print(Wert von \"zwanzgerl\":);
System.out.println(zwanzgerl);
System.out.print(Wert von \"zw\":);
System.out.println(zw);
```

Was wird gedruckt?

Antwort

```
Wert von "zwanzgerl": 5
Wert von "zw": 6
```

Der Grund ist, dass Integer- und Double-Variablen keine Verweise sind (wie Objektvariablen) sondern den jeweiligen Wert direkt enthalten.

Mit anderen Worten: eine Integer-Variable enthält einen Integer-Wert, eine Objekt-Variable enthält eine Speicheradresse (unter der sich ein Objekt befindet).

Initialisierung

Man muss Variablen nicht initialisieren:

```
int a;
int b = 4;
a = b + 2;
```

Sie müssen aber vor der ersten Verwendung einen Wert bekommen:

```
int a;
int b = 4;
System.out.println(a);
```

ist ein Programmierfehler (den Java schon beim Compilieren erkennt, C aber nicht.)

Nochmal Wertzuweisung

Man kann auch schreiben:

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

Dadurch wird der Wert von `a` um eins erhöht.

Manche Programmierer verwenden dafür die Kurzform

```
a++;
```

Daher auch der Name C++ für „Nachfolger von C“.

Alles ist endlich

Integer rangieren von $-2.147.483.648$ bis $2.147.483.647$, d.h. insgesamt 2^{32} verschiedene Werte.

```
int weltbevoelkerung = 2000000000;
weltbevoelkerung = 2 * weltbevoelkerung;
System.out.println(weltbevoelkerung);
```

Ergebnis:

```
-294967296
```

Alles ist endlich

Doubles rangieren von -10^{300} bis 10^{300}, haben aber nur ca. 15 Nachkommastellen:

```
double originalPreis = 3E14; // 300 Billiarden Euro
double sonderPreis = originalPreis - 0.05; // um 5 Cent reduziert
double rabatt = originalPreis - sonderPreis;
System.out.println(rabatt);
```

gibt `0.0625` aus.

Wenn das nicht reicht

Es gibt den Datentyp `long`, der insgesamt 2^{64} verschiedene Werte aufweist. Long-Konstanten schreibt man so: `1340000000000000L`.

Und dann gibt es noch die Klassen `BigInteger` und `BigDecimal` die praktisch unlimitiert sind.

Nachteil: Langsam und umständlich zu benutzen.

```
import java.math.BigInteger;

...

BigInteger b = new BigInteger("100000000");
b = b.multiply(b) ; b = b.multiply(b) ; b = b.multiply(b);
b = b.subtract(new BigInteger("1"));
System.out.println(b);
```

Druckt:
9999999999999999999999999999999999999999999999999999999999999999

Typkonversion

```
int euros = 2;
double gesamt = euros; // ok
```

```
double euros = 2.0;
int anzahlEuros = euros; // geht nicht
```

Im ersten Beispiel wird der Integer-Wert automatisch in Double konvertiert.

Im zweiten Beispiel nicht.

Typkonversion

Man kann aber schreiben:

```
double euros = 2.50;
int anzahlEuros = (int)euros;
System.out.println(anzahlEuros);
```

Das ist eine explizite Typkonversion (*typecast*).

Hier werden einfach alle Dezimalstellen abgeschnitten.

Will man runden, so verwende man

```
double a = 3.759;
System.out.println((int)Math.round(a));
```

Die (statische) Methode `Math.round` berechnet den nächstgelegenen ganzzahligen Double-Wert.

Rundungsfehler

```
double f = 4.35;
int n = (int)(100 * f);
System.out.print(n);
```

Druckt `434`.

Grund: In Binärdarstellung ist 4,35 ein echt periodischer Bruch.

```
(int)Math.round(100 * f);
```

hat Wert 435.

Konstanten

```
int flaschen = 3;
int dosen = 5;
int mengeFanta = flaschen * 0.5 + dosen * 0.33;
```

ist schlechter Stil, da `0.5` und `0.33` einfach so dastehen.

Besser:

```
final double FLASCHEN_INHALT = 0.5;
final double DOSEN_INHALT = 0.33;
double mengeFanta = flaschen * FLASCHEN_INHALT + dosen * DOSEN_INHAL
```

Werte, die mit `final` deklariert werden, können nur einmal initialisiert und danach nicht mehr verändert werden.

Vorteil gegenüber Variablen: Effizienz + Dokumentation.

Numerische Konstanten wie `0.5` mitten im Programm sind schlechter Stil.

Vordefinierte Double-Konstanten: `Math.PI` und `Math.E`.

Arithmetik

Plus `+` und Mal `*` hatten wir schon.

Division wird als `/` notiert.

Vorsicht: Sind beide Operanden von `/` Integers, so wird abgerundet.

```
int s1 = 5;
int s2 = 6;
int s3 = 3;
double mittelwert = (s1 + s2 + s3) / 3;
```

`mittelwert` hat den Wert 4 (statt `4.666666...`)

Beliebter Programmierfehler.

Richtig:

```
double mittelwert = (s1 + s2 + s3) / 3.0;
```


Was gibt's sonst noch

- Die „Punkt vor Strich" Regel gilt.
- Mathematische Funktionen sind in der Klasse `Math` definiert.
- Automatische Typkonversionen erfolgen von innen nach außen.

Beispiel: Die „Lösungsformel":

```
double a;
double b;
double c;
/* Wertzuweisung */
x1 = (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a);
x2 = (-b - Math.sqrt(b * b - 4 * a * c)) / (2 * a);
```

Ebenso: `a * a + b * b - 2 * a * b * Math.sin(phi);`

Details in JBuilder → Help.

Zeichenketten

Der Datentyp `String` besteht aus Zeichenketten, d.h. Folgen von Buchstaben und Sonderzeichen.

```
String name = "Matthias";
String name = "Johanna";
System.out.println(name);
```

Druckt: `Johanna`.

```
int n = name.length();
```

Die Variable `n` hat den Wert 7.

Teile einer Zeichenkette

Der Ausdruck

```
s.substring(anfang, endePlusEins)
```

bezeichnet die Teilzeichenkette von `s` angefangen vom Zeichen an der Position `anfang` bis (ausschließlich) zum Zeichen an der Position `endePlusEins`.

Positionen beginnen immer bei Null.

```
String s = "Hello, World!";
String sub1 = s.substring(0,5);
String sub2 = s.substring(4,8);
```

Was sind die Werte von `sub1` und `sub2`?

Welcher `substring`-Ausdruck hat den Wert `World` ?

Teile einer Zeichenkette

Antwort: `sub1` den Wert `Hello` und `sub2` den Wert `o, W`. Der Ausdruck `s.substring(7, 12)` hat den Wert `World`.

Will man alle Zeichen von `anfang` bis zum Ende der Zeichenkette, dann kann man

```
s.substring(anfang, s.length())
```

schreiben. Das letzte Zeichen hat nämlich die Position `s.length()` − 1.

Eine Kurzform dafür ist `s.substring(anfang)`.

Fehlerbehandlung

Ruft man `s.substring` mit unpassenden Argumenten auf, so gibt es einen Fehler. Z.B.: `s.substring(4,30)` führt zu:

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
String index out of range: 20
        at java.lang.String.substring(String.java:1473)
        at Namen.main(Namen.java:5)
```

Man sagt: der Ausdruck wirft eine Ausnahme (*throws an exception*).

Es ist möglich, so eine Ausnahme im Programm “aufzufangen” und benutzerdefinierte Befehle auszuführen, z.B. eine ordentliche Fehlermeldung.

Noch besser ist es, das Auftreten solcher Ausnahmen von vornherein zu vermeiden.

Verkettung

Zeichenketten kann man aneinanderhängen ("verketten", "konkatenieren" von lat. *catena* = Kette).

In Java verwendet man dafür das Pluszeichen:

Der Ausdruck `"Matthias" + "Johanna"` hat den Wert `MatthiasJohanna`.

Der Ausdruck

```
"Euro" + "s".substring(0,n)
```

hat den Wert `Euro` oder `Euros` ja nachdem, ob `n` gleich 0 oder 1 ist. Alle anderen Werte von `n` sind nicht erlaubt.

Was "sind" Zeichenketten?

Eine Zeichenkette ist ein Objekt.

Es versteht u.a. die Methoden `length` und `substring`.

Die Methode `length` liefert die Länge zurück.

Die Methode `substring` ein neues String-Objekt, das den jeweiligen Teilstring enthält.

Man kann eine Zeichenkette nie verändern (im Gegensatz zu veränderlichen Objekten wie `Rectangles`).

Im Computer ist ein String eine Speicheradresse. Unter dieser Adresse befindet sich die Länge, z.B. n. In den n darauffolgenden Speicherstellen befinden sich dann die Zeichen.

In der Sprache Java gibt es keine Möglichkeit, diese Zeichen oder die Länge zu verändern, obwohl die Maschinensprache das im Prinzip zuließe.

Vorteil: Weniger Programmierfehler.

Was "sind" Zeichenketten?

Nachteil: JRE muss die nicht mehr benötigten Zeichenketten automatisch "aufräumen" (sog. *garbage collection*), was Zeit kostet.

In C darf man (und muss man) Zeichenketten verändern und explizit Platz für sie reservieren etc.

Das ist die Hauptursache für Sicherheitslücken!

Z.B. *buffer overflow*.

Aktuelle Forschung: Speicherkonzept wie in Java aber gleichzeitig Effizienz von C.

Verkettung mit Zahlwerten

```
double betrag = 34.99;
int nummerMahnung = 2;
String anweisung = nummerMahnung + ". Mahnung: " +
        "Bitte zahlen Sie " + betrag + " EUR.";

System.out.println(anweisung);
```

Druckt: `2. Mahnung: Bitte zahlen Sie 34.99 EUR.`

Verkettung mit Zahlwerten

Ist ein Operand von + eine Zeichenkette, so wird der andere automatisch in eine Zeichenkette umgewandelt. Das ist keine Typkonversion:

```
String betrag = 34.99 * 2;
```

löst aus:

```
incompatible types
found   : double
required: java.lang.String
        String betrag = 34.99 * 2;
```


Verkettung mit Zahlwerten

Man kann schreiben `""+x` um `x` in eine Zeichenkette umzuwandeln.

Aber Vorsicht:

```
String anweisung = nummerMahnung + ". Mahnung: " +
    "Bitte zahlen Sie " + betrag/3 + " EUR.";
System.out.println(anweisung);
```

Ergebnis:

```
2. Mahnung: Bitte zahlen Sie 11.663333333333334 EUR.
```


Abhilfe: Formatierte Ausgabe

```
import java.text.NumberFormat;
...
NumberFormat formatierer = NumberFormat.getNumberInstance();

formatierer.setMaximumFractionDigits(2);
formatierer.setMinimumFractionDigits(2);

double betrag = 34.99;
int nummerMahnung = 2;
String anweisung = nummerMahnung + ". Mahnung: " +
   "Bitte zahlen Sie " + formatierer.format(betrag/3) + " EUR.";
```

Ergebnis:

```
2. Mahnung: Bitte zahlen Sie 11,66 EUR.
```

Sogar Tausenderpunkte werden eingesetzt, z.B.: `1.192.279,33 EUR`.

Benutzernamen

Wir möchten aus dem ersten und letzten Buchstaben des Namens und einer laufenden Nummer einen Benutzernamen erzeugen:

```
String name = "Johanna";
int lfdNo = 1728;
```

Der Benutzername sollte `Ja1728` sein.

Kein Problem

```
String benutzerName;
benutzerName = name.substring(0,1) +
     name.substring(name.length() - 1) + lfdNo;
```


Parsing von Zeichenketten

Wie erhalten wir aus einem Benutzernamen die laufende Nummer?

`benutzerName.substring(2)`

enthält zwar die Ziffern der lfd. Nr. ist aber immer noch ein String.

Die Lösung:

```
int nummer = Integer.parseInt(benutzerName.substring(2));
```

`parseInt` ist eine statische Methode der Klasse `Integer` und dient dazu, eine Zeichenkette in einen integer umzuwandeln.

Statische Methoden werden nicht an ein Objekt geschickt, sondern können “einfach so” ausgeführt werden.

Details später.

Parsing von Doubles

. . . geht analog mit `Double.parseDouble`, z.B.:

```
double c = Double.parseDouble("2.97E9"); /* oder so */
```

Aber Vorsicht: eine “deutsche” Zahl, wie 1.234,59 kann `parseDouble` nicht verarbeiten.

Wie das geht, siehe Java Reference Manual