

Der Datentyp String

Zeichenketten bzw. **Strings** sind eine häufig benutzte Datenstruktur. Man könnte auch mit Arrays aus Zeichen arbeiten. Da aber diese Datenstruktur so häufig vorkommt, stellt Java einen Bibliothekstyp **String** zur Verfügung. Dieser Typ ist sogar in die Sprache Java integriert, weil der Compiler speziellen Code für Strings erzeugt.

Stringvariable und -vergleiche

Stringvariable werden mit dem Typ **String** deklariert:

```
String s1, s2;
```

Eine Zuweisung geschieht am einfachsten direkt durch Angabe des Strings eingeschlossen in doppelte Hochkommata:

```
s1 = "Hallo";  
s2 = "Hallo";
```

Die Zuweisung kann auch direkt bei der Deklaration erfolgen. Achtung: 'x' ist eine Zeichenkonstante, "x" eine Stringkonstante. Da Strings wie Arrays Objekte sind, müssten sie eigentlich mit **new** erzeugt werden:

```
String s1 = new String("Hallo");
```

Dies kann man auch machen, man muss es aber nicht. Java erlaubt bei Stringobjekten die einfachere Syntax.

Vergleicht man die oben angegebenen Strings auf folgende Art

```
if (s1==s2) {Out.print ("Strings sind gleich.");}  
else {Out.print ("Strings sind nicht gleich.");}
```

erhält man *Strings sind nicht gleich*. Warum? Strings sind Objekte und die Stringvariablen enthalten dementsprechend nur Zeiger auf die Stringobjekte. Da **s1** und **s2** verschiedene Speicheradressen enthalten, wird keine Gleichheit festgestellt. Schreibt man stattdessen

```
s1 = "Hallo";  
s2 = s1;
```

so zeigen **s1** und **s2** auf das selbe Stringobjekt, sie enthalten also beide die gleiche Speicheradresse. Dann ergibt **s1==s2** den Wert **true**. Zum richtigen Vergleich zweier Stringobjekte wird die Methode **equals** benutzt:

```
s1 = "Hallo";
s2 = "Hallo";
if (s1.equals(s2)) {Out.print ("Strings sind gleich.");}
else {Out.print ("Strings sind nicht gleich.");}
```

Stringobjekte sind in der Größe unveränderlich. Trotzdem kann man

```
s1 = "Hallo";
s1 = s1 + " Welt";
Out.print(s1);
```

schreiben. Bei diesem Anhängen wird automatisch ein neues Stringobjekt mit dem neuen Inhalt und der neuen Größe erzeugt und `s1` zeigt auf das neue Objekt. Das alte "Hallo" steht jetzt ohne Zeiger im Speicher herum und wird bei Gelegenheit vom Garbage Collector entsorgt.

In Strings können die schon bei den Zeichen angegebenen Konstanten benutzt werden:

```
s1 = "Der String \"Hallo\" steht in Anf\u00fchrungszeichen.";
```

Weitere Stringoperationen

Auf die einzelnen Zeichen eines Strings kann nicht wie bei Arrays direkt über einen Index zugegriffen werden. Dazu benötigt man die Methoden der String-Bibliothek.

Für die folgenden Beschreibungen sei angenommen, dass die folgenden Variablen deklariert seien:

```
String s = "a long string";
String s2;
char ch;
int i;
boolean jaOderNein;
```

Die Operation

```
i = s.length();
```

liefert die Anzahl der Zeichen im String `s`. Achtung: Bei Arrays schreibt man z.B. `a.length` ohne Klammern. Bei Strings geschieht der Zugriff aber über eine Methode `length`, also muss man Klammern setzen.

```
ch = s.charAt(3);
```

liefert das Zeichen mit dem Index 3 aus dem String `s`, also hier ein „o“. Die Indizierung beginnt wie bei Arrays mit 0.

```
i = s.compareTo("also a long string");
```

liefert -1, weil `s` vor dem angegebenen String in der lexikographischen Ordnung liegt. Der Wert 0 ergibt sich bei Gleichheit und der Wert +1, wenn der angegebene String hinter dem String `s` steht.

```
i = s.compareToIgnoreCase("Also a long string");
```

liefert das gleiche Ergebnis wie vorher, weil Groß- und Kleinschreibung nicht beachtet werden.

```
jaOderNein = s.equalsIgnoreCase("Also a long string");
```

liefert das entsprechende Ergebnis beim Vergleich ohne Beachtung von Groß- und Kleinschreibung.

```
i = s.indexOf("ng") + s.indexOf('a');
```

liefert die Position des ersten Vorkommens von „ng“ in `s` oder -1, falls „ng“ nicht gefunden wurde. Hier erhält man also 4 + 1. Man sieht, dass man auch eine `char`-Konstante als Parameter übergeben kann.

```
i = s.indexOf("ng", 5);
```

liefert die Position des ersten Vorkommens von „ng“ in `s`, beginnt die Suche aber erst ab Position 5. Im Beispiel wird also 11 zurück geliefert.

```
i = s.lastIndexOf("ng");
```

liefert die Position des letzten Vorkommens von „ng“ in `s`.

```
s2 = s.substring(2);
```

liefert den Teilstring von `s` ab der Position 2, hier „long string“.

```
s2 = s.substring(2,6);
```

liefert den Teilstring von `s` ab der Position 2 und bis ausschließlich 6, hier „long“.

```
s2 = s.trim();
```

liefert den `s` ohne führende oder abschließende Leerzeichen.

```
s2 = s.toLowerCase();
```

liefert den `s`, wobei alle Groß- in Kleinbuchstaben umgewandelt werden.

```
s2 = s.toUpperCase();
```

liefert den `s`, wobei alle Klein- in Großbuchstaben umgewandelt werden.

```
jaOderNein = s.startsWith("abc")
```

liefert `true`, wenn `s` mit „abc“ beginnt.

```
jaOderNein = s.endsWith("abc")
```

liefert `true`, wenn `s` mit „abc“ endet.

Aufbauen von Strings

Da String-Objekte konstante Länge haben, eignen sie sich nicht zum schrittweisen Aufbau einer Zeichenkette. Daher wird ein String entweder in seiner endgültigen Form direkt erzeugt oder in einem `char`-Array oder einem `StringBuffer`-Objekt aufgebaut und anschließend in ein `String`-Objekt umgewandelt.

Direkter Aufbau

```
String s = "einfach hinschreiben!";
```

Aufbau aus einem char-Array

```
char[] a = new char[80];  
for (int i=0; i<80; i++) a[i] = In.read();  
String s = new String(a);
```

Aufbau aus einem StringBuffer

Der in der Java-Bibliothek enthaltene Typ `StringBuffer` verhält sich in vielen Dingen wie ein `String`, läßt aber Veränderungen zu.

```
StringBuffer b = new StringBuffer();
```

erzeugt ein leeres `StringBuffer`-Objekt. `b` zeigt auf dieses leere Objekt.

```
b.append(x);
```

hängt `x` an `b` an. `x` kann vom Typ `char`, `int`, `long`, `float`, `double`, `boolean`, `String` und `char[]` sein.

```
i = b.length();
```

liefert die Anzahl der Zeichen in `b`.

```
b.insert(pos, x);
```

fügt `x` an der Stelle `pos` in `b` ein. Für die Typen von `x` gilt das Gleiche wie bei `append`.

```
b.delete(from, to);
```

löscht in `b` die Zeichen von Position `from` (inklusive) bis Position `to` (exklusive).

```
b.replace(from, to, "abc");
```

ersetzt in `b` die Zeichen von Position `from` (inklusive) bis Position `to` (exklusive) durch „abc“.

```
s = b.substring(from, to);
```

liefert den Teil von `b` von Position `from` (inklusive) bis Position `to` (exklusive) als `String`.

```
ch = b.charAt(i);
```

liefert das Zeichen mit Index `i` aus `b`.

```
b.setChar(i, 'x');
```

ersetzt das Zeichen mit Index `i` aus `b` durch „x“.

```
s = b.toString();
```

liefert den Inhalt von `b` als `String`.

Stringkonversionen

Für die Umwandlung von Zahlen in Strings und umgekehrt stellt die Java-Bibliothek geeignete Methoden bereit.

```
int i = Integer.parseInt("123");
```

erzeugt aus dem `String` „123“ die `int`-Variable 123.

```
long l = Long.parseLong("1234567890123");
```

erzeugt aus dem `String` „1234567890123“ die `long`-Variable 1234567890123.

```
float f = Float.parseFloat("3.14");
```

erzeugt aus dem `String` „3.14“ die `float`-Variable 3.14.

```
double d = Double.parseDouble("1.6e-19");
```

erzeugt aus dem `String` „1.6e-19“ die `float`-Variable 1.6E-19.

```
String s = String.valueOf(x);
```

erzeugt aus dem Wert von `x` (Typ `int`, `long`, `float`, `double`, `boolean` oder `char[]`) einen `String`. Für die Ausgabe von Werten ist aber meist der `+`-Operator bequemer:

```
Out.println("Der Wert von x ist " + x + ".");
```

Für die Umwandlung eines Strings in ein `char`-Array gibt es auch eine Methode:

```
char[] a = s.toCharArray();
```