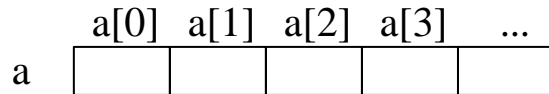


# Eindimensionale Arrays



## Array = Tabelle gleichartiger Elemente



- Name *a* bezeichnet das gesamte Array
- Elemente werden über Indizes angesprochen (z.B. *a[3]*)
- Indizierung beginnt bei 0
- Elemente sind "namenlose" Variablen

## Deklaration

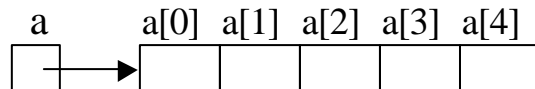
```
int[] a;  
float[] b;
```

- deklariert ein Array namens *a* (bzw. *b*)
- seine Elemente sind vom Typ *int* (bzw. *float*)
- seine Länge ist noch unbekannt

## Erzeugung

```
a = new int[5];  
b = new float[10];
```

- legt ein neues *int*-Array mit 5 Elementen an (aus dem Heap-Speicher)
- weist seine Adresse *a* zu



Array-Variablen enthalten Zeiger auf Arrays!  
(Zeiger = Speicheradresse)

## Zugriff auf Arrayelemente

```
a[3] = 0;  
a[2*i+1] = a[2];
```

- Arrayelemente werden wie Variablen benutzt
- Index kann ein ganzzahliger Ausdruck sein
- Laufzeitfehler, falls Array noch nicht erzeugt wurde
- Laufzeitfehler, falls Index  $< 0$  oder  $\geq$  Arraylänge

## Arraylänge abfragen

```
int len = a.length;
```

- *length* ist Standardoperator, der auf alle Arrays angewendet werden kann.
- Liefert Anzahl der Elemente (hier 5).

## Beispiele

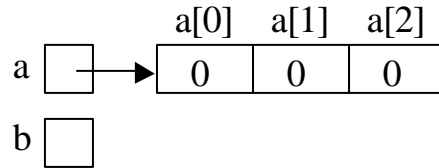
```
for (int i = 0; i < a.length; i++) // Array einlesen  
    a[i] = In.readInt();
```

```
int sum = 0; // Elemente aufaddieren  
for (int i = 0; i < a.length; i++)  
    sum += a[i];
```

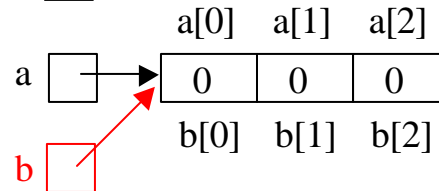
# Arrayzuweisung



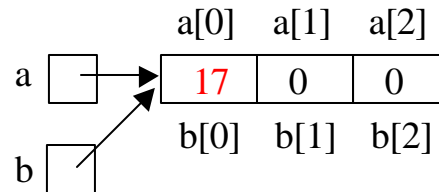
```
int[] a, b;  
a = new int[3];
```



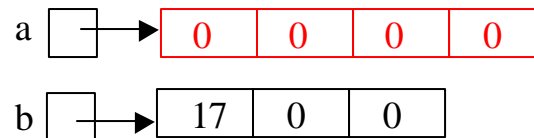
```
b = a;
```



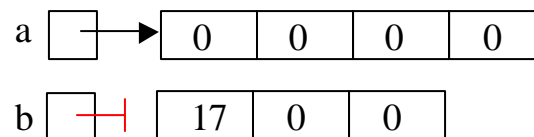
```
a[0] = 17;
```



```
a = new int[4];
```



```
b = null;
```



Arrayelemente werden in Java standardmäßig mit 0 initialisiert

`b` bekommt denselben Wert wie `a`.

Arrayzuweisung ist in Java **Zeigerzuweisung!**

ändert in diesem Fall auch `b[0]`

`a` zeigt jetzt auf neues Array.

`null`: Spezialwert, der auf kein Objekt zeigt; kann jeder Arrayvariablen zugewiesen werden

# Freigeben von Arrayspeicher



## Garbage Collection (Automatische Speicherbereinigung)

Objekte, auf die kein Zeiger mehr verweist, werden automatisch eingesammelt.  
Ihr Speicher steht für neue Objekte zur Verfügung

```
static void P() {
```

```
    int[] a = new int[3];
```

```
    int[] b = new int[4];
```

```
    int[] c = new int[2];
```

```
    ...
```

```
    ...
```

```
    b = a;
```

```
    ...
```

```
    ...
```

```
    c = null;
```

```
    ...
```

```
    ...
```

```
    ...
```

```
}
```



kein Zeiger mehr auf dieses Objekt  
⇒ wird eingesammelt



kein Zeiger mehr auf dieses Objekt  
⇒ wird eingesammelt

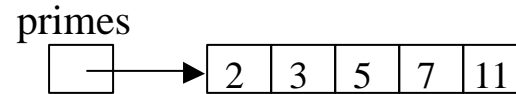


Am Methodenende werden lokale Variablen  
freigegeben ⇒ Zeiger *a*, *b*, *c* fallen weg  
⇒ Objekt wird eingesammelt

# Initialisieren von Arrays



```
int[] primes = {2, 3, 5, 7, 11};
```



identisch zu

```
int[] primes = new int[5];  
primes[0] = 2;  
primes[1] = 3;  
primes[2] = 5;  
primes[3] = 7;  
primes[4] = 11;
```

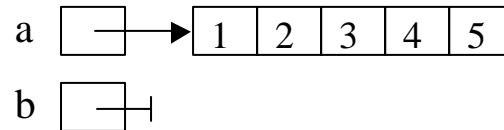
Initialisierung kann auch bei der Erzeugung erfolgen

```
primes = new int[] {2, 3, 5, 7, 11};
```

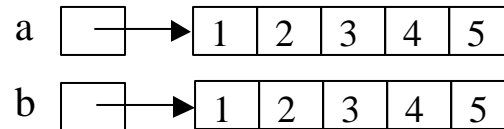
# Kopieren von Arrays



```
int[] a = {1, 2, 3, 4, 5};  
int[] b;
```



```
b = (int[]) a.clone();
```



Typumwandlung nötig, da *clone* etwas vom Typ *Object[]* liefert

# Kommandozeilenparameter



## Programmaufruf mit Parametern

```
java Programmname par1 par2 ... parn
```

## Parameter werden als String-Array an main-Methode übergeben

```
class Sample {  
    public static void main (String[] arg) {  
        for (int i = 0; i < arg.length; i++)  
            Out.println(arg[i]);  
        ...  
    }  
}
```

### Aufruf z.B.

```
java Sample Anton /a 10
```

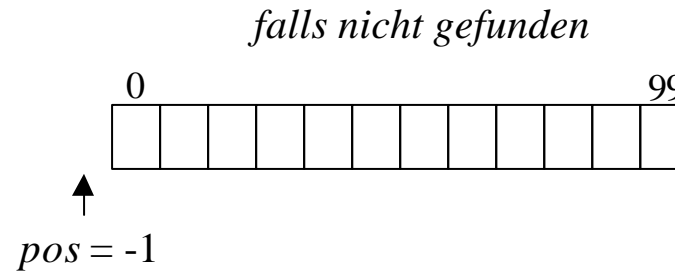
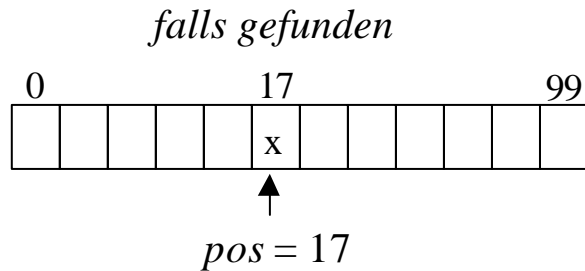
### Ausgabe:

```
Anton  
/a  
10
```

# Beispiel: sequentielles Suchen



Suchen eines Werts  $x$  in einem Array



```
static int search (int[] a, int x) {  
    int pos = a.length - 1;  
    while (pos >= 0 && a[pos] != x ) pos--;  
    // pos == -1 || a[pos] == x  
    return pos;  
}
```

← gewünschtes Ergebnis

**Achtung:** `int[] a` wird nur als Zeiger übergeben.

Würde `search` etwas in `a` ändern (z.B. `a[3] = 0;`), würde sich diese Änderung auch auf das Array im Rufer auswirken.

# Primzahlenberechnung: Sieb des Erathostenes



## 1. "Sieb" wird mit den natürlichen Zahlen ab 2 gefüllt

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, ...

## 2. Erste Zahl im Sieb ist Primzahl. Entferne sie und alle ihre Vielfachen

② 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, ...

3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, ...

## 3. Wiederhole Schritt 2

③ 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, ...

5, 7, 11, 13, 17, 19, 23, 25, ...

## ... Wiederhole Schritt 2

⑤ 7, 11, 13, 17, 19, 23, 25, ...

7, 11, 13, 17, 19, 23, ...

# Implementierung



Sieb = *boolean*-Array, Zahl  $i$  im Sieb  $\Leftrightarrow sieve[i] == true$

0	1	2	3	4	5	6	7	8	9	...
false	false	true	true	true	true	true	true	true	true	...

Zahl  $i$  entfernen:  $sieve[i] = false$

0	1	2	3	4	5	6	7	8	9	...
false	false	false	true	false	true	false	true	false	true	...

```
static void printPrimes (int max) {  
    boolean[] sieve = new boolean[max + 1];  
    for (int i = 2; i <= max; i++) sieve[i] = true;  
    for (int i = 2; i <= max; ) {  
        Out.print(i + " "); // i is prime  
        for (int j = i; j <= max; j = j + i) sieve[j] = false;  
        while (i <= max && !sieve[i]) i++;  
    }  
}
```

# *Beispiel: Monatstage berechnen*



## **Bisher mit Switch-Anweisung gelöst**

```
switch (month) {  
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:  
        days = 31; break;  
    case 4: case 6: case 9: case 11:  
        days = 30; break;  
    case 2:  
        days = 28;  
}
```

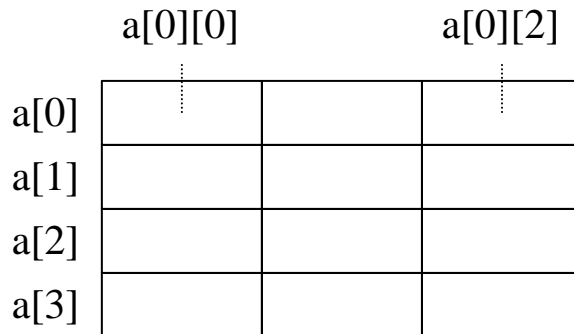
## **Besser mit Tabelle**

```
int[] days = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
...  
int d = days[month];
```

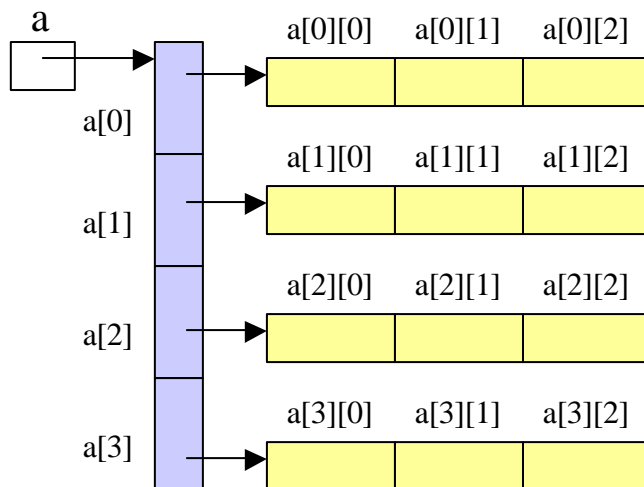
# Mehrdimensionale Arrays



**Zweidimensionales Array  $\equiv$  Matrix**



**In Java als Array von Arrays implementiert**



**Deklaration und Erzeugung**

```
int[][] a;  
a = new int[4][3];
```

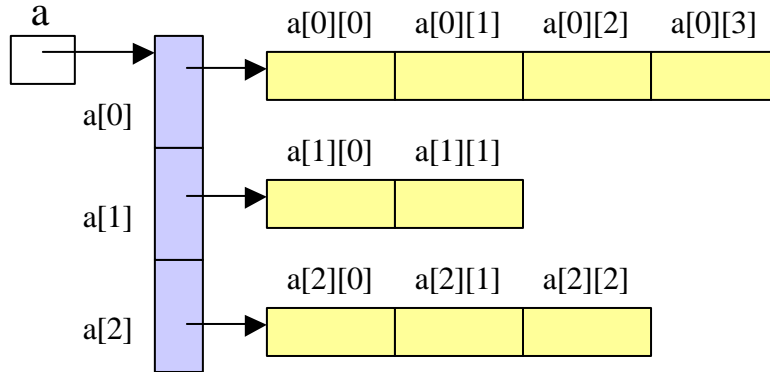
**Zugriff**

```
a[i][j] = a[i][j+1];
```

# Mehrdimensionale Arrays



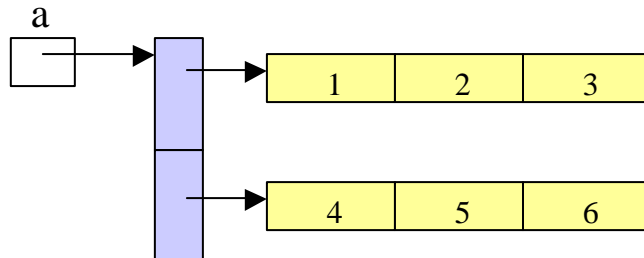
**Zeilen können unterschiedlich lang sein** (das ist aber selten sinnvoll)



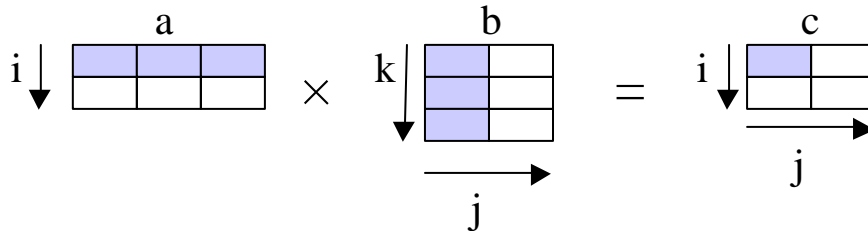
```
int[][] a = new int[3][];  
a[0] = new int[4];  
a[1] = new int[2];  
a[2] = new int[3];
```

## Initialisierung

```
int[][] a = {{1, 2, 3},{4, 5, 6}};
```

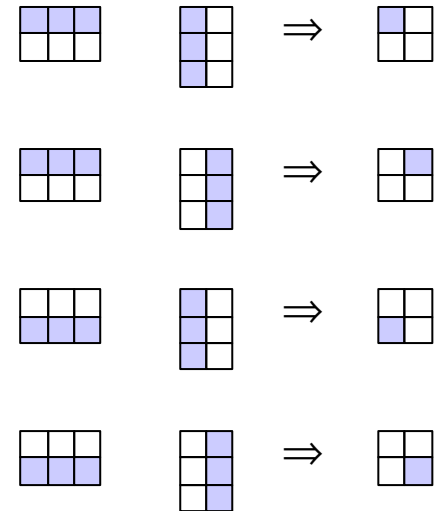


# Beispiel: Matrixmultiplikation



$$c_{0,0} = a_{0,0} * b_{0,0} + a_{0,1} * b_{1,0} + a_{0,2} * b_{2,0}$$

```
static float[][] matrixMult (float[][] a, float[][] b) {  
    float[][] c = new float[a.length][b[0].length];  
    for (int i = 0; i < a.length; i++)  
        for (int j = 0; j < b[0].length; j++) {  
            float sum = 0;  
            for (int k = 0; k < b.length; k++)  
                sum += a[i][k] * b[k][j];  
            c[i][j] = sum;  
        }  
    return c;  
}
```



# Datentyp *char*



```
char ch = 'x';
```

Zeichenvariable

Zeichenkonstante  
(unter einfachen Hochkommas)

Zeichen braucht man zur Verarbeitung von Texten, Namen, Bezeichnungen.

## Zeichencodes

**ASCII** (American Standard Code for Information Interchange)

- 1 Zeichen = 1 Byte (128 bzw. 256 Zeichen darstellbar)
- z.B. in Pascal oder C verwendet

**Unicode** ([www.unicode.org](http://www.unicode.org))

- 1 Zeichen = 2 Bytes (65536 Zeichen darstellbar)
- auch Umlaute, griechische, arabische Zeichen etc.
- z.B. in Java und C# verwendet
- ASCII ist Teilmenge von Unicode

# ASCII



0	NUL	16	DLE	32	space	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

## Wichtige Steuerzeichen

**BS** *backspace* löscht Zeichen vor Cursor

**HT** *horizontal tab* Tabulatorsprung

**ESC** *escape*

**CR** *carriage return* Zeilenvorschub

**LF** *line feed* folgt auf CR

**FF** *form feed* Seitenvorschub

# Unicode



0000 - 007F	ASCII-Zeichen
0080 - 024F	Umlaute, Akzente, Sonderzeichen
0370 - 03FF	griechische Zeichen
0400 - 04FF	cyrillische Zeichen
0530 - 058F	armenische Zeichen
0590 - 05FF	hebräische Zeichen
0600 - 06FF	arabische Zeichen
...	...

Details siehe  
<http://www.unicode.org>

## Deutsche Umlaute

00E4	ä	00C4	Ä	00DF	ß
00F6	ö	00D6	Ö		
00FC	ü	00DC	Ü		

# Unicode-Zeichenkonstanten



Alle Zeichen können auch mit ihrem Unicode-Wert angegeben werden:

`\udddd`

Beispiele:

<code>\u0041</code>	'A'
<code>\u000d</code>	CR (carriage return)
<code>\u0009</code>	TAB
<code>\u03c0</code>	$\pi$

## Spezielle Zeichen

<code>\n</code>	LF (line feed, newline)
<code>\r</code>	CR (carriage return)
<code>\t</code>	TAB
<code>\\</code>	\
<code>\'</code>	'
<code>\ddd</code>	Zeichenwert als Oktalzahl

# Zeichen-Operationen



## Zuweisungen

```
char ch1, ch2 = 'a';  
ch1 = ch2;          // ok, gleicher Typ  
int i = ch2;        // ok, char kann int zugewiesen werden  
ch1 = (char)i;      // Zuweisung nach Typumwandlung möglich
```

double  $\supseteq$  float  $\supseteq$  long  $\supseteq$  int  $\supseteq$  short  $\supseteq$  byte  
 $\supseteq$  char

## Vergleiche (==, !=, <, <=, >, >=)

Zeichen sind nach Unicode-Wert geordnet;  
Buchstaben und Ziffern liegen aufeinanderfolgend

```
if ('a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'Z') ...
```

## Arithmetische Operationen (+, -, \*, /, %)

```
10 + (ch - 'A')    // Ergebnistyp: int
```

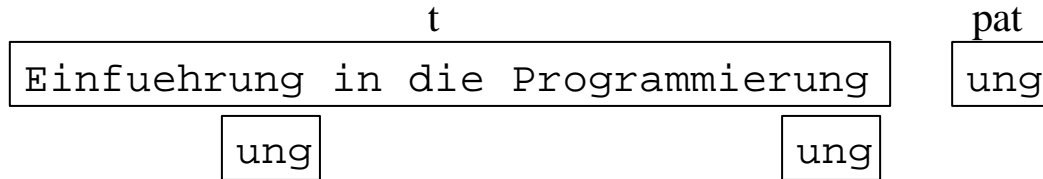
## Zeichenarrays

```
char[] s = new char[20]; // initialisiert alle Elemente mit '\u0000'  
char[] t = {'a', 'b', 'c'};
```

# Beispiel: Textsuche

geg.: Text  $t$ , Muster  $pat$

ges.: erstes Vorkommen von  $pat$  in  $t$



Ergebnis:  $pos \geq 0$ :  $pat$  in  $t$  an Stelle  $pos$

$pos < 0$ :  $pat$  kommt nicht in  $t$  vor

```
static int search (char[] t, char[] pat) {
    int last = t.length - pat.length;
    for (int i = 0; i <= last; i++) {
        if (t[i] == pat[0]) {
            int j = 1;
            while (j < pat.length && pat[j] == t[i+j]) j++;
            // j == pat.length || pat[j] != t[i+j]
            if (j == pat.length) return i;
        }
    }
    return -1;
}
```

# Beispiel: Nachbauen von readInt



```
static int readInt() {
    int val = 0;
    char ch = In.read(); // liest ein einzelnes Zeichen
    while (In.done() && '0' <= ch && ch <= '9') {
        val = 10 * val + (ch - '0');
        ch = In.read();
    }
    // ! In.done() || ch < '0' || ch > '9'
    return val;
}
```

Schreibtischttest: Eingabe 123+

val	ch	
<del>0</del>	<del>'1'</del> (49)	'0' = 48
<del>1</del>	<del>'2'</del> (50)	
<del>12</del>	<del>'3'</del> (51)	
123	'+' (43)	

"Horner-Schema"

# Standardfunktionen mit Zeichen



if (**Character.isLetter(ch)**) ...

true, wenn *ch* ein Unicode-Buchstabe ist

if (**Character.isDigit(ch)**) ...

true, wenn *ch* eine Ziffer ist

if (**Character.isLetterOrDigit(ch)**) ...

if (**Character.isJavaIdentifierStart(ch)**) ...

true, wenn ein Java-Name mit *ch* beginnen kann

if (**Character.isJavaIdentifierPart(ch)**) ...

true, wenn *ch* in einem Java-Namen vorkommen kann

if (**Character.isLowerCase(ch)**) ...

true, wenn *ch* ein Kleinbuchstabe ist

if (**Character.isUpperCase(ch)**) ...

true, wenn *ch* ein Großbuchstabe ist

ch1 = **Character.toUpperCase(ch2)**;

wandelt *ch2* in einen Großbuchstaben um

ch1 = **Character.toLowerCase(ch2)**;

wandelt *ch2* in einen Kleinbuchstaben um

# Datentyp String



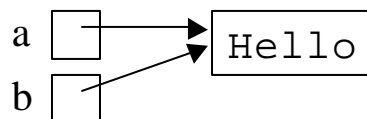
**String** a, b;

Bibliothekstyp für Zeichenarrays

a = "Hello";

Stringkonstante (unter doppelten Hochkommas)

b = a;



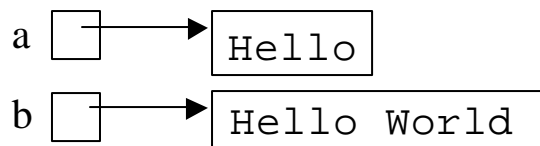
Stringvariablen sind Zeiger auf Stringobjekte

Stringzuweisung ist eine Zeigerzuweisung

Stringobjekte sind nicht als Arrays ansprechbar

Stringobjekte sind nicht veränderbar

b = a + " World";

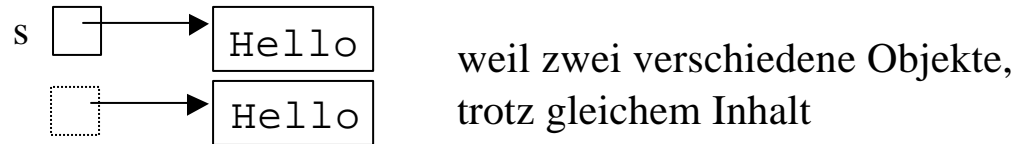


Verkettung mit "+" erzeugt neues Stringobjekt (relativ teure Operation)

# Stringvergleiche



```
String s = In.readWord(); // liest ein Wort, z.B. Hello  
if (s == "Hello") ... // liefert false! (Zeigervergleich)
```



```
if (s.equals("Hello")) ... // liefert true! (Wertvergleich)
```

aber:

```
String s = "Hello";  
if (s == "Hello") .. // liefert true, weil gleichlautende Stringkonstanten  
// nur einmal als Objekt abgespeichert werden.
```

trotzdem Wertvergleich immer mit *equals* durchführen

# Stringoperationen



```
String s = "a long string";
```

0	1	2	3	4	5	6	7	8	9	10	11	12
a		l	o	n	g		S	t	r	i	n	g

```
int len = s.length();
```

liefert Anzahl der Zeichen in *s*  
(im Gegensatz zu *arr.length* sind Klammern nötig!)

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

liefert das Zeichen mit Index 3 (hier 'o')

```
int i = s.indexOf("ng");
```

liefert Index des 1. Vorkommens von "ng" in *s* (hier 4) oder -1

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

liefert Index des 1. Vorkommens von "ng" ab Index 5 (hier 11)

```
i = s.indexOf('n');
```

geht auch mit *char*

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

liefert Index des letzten Vorkommens von "ng" (Varianten wie oben)

```
String x;
```

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

liefert Teilstring ab Index 2 (hier "long string")

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

liefert Teilstring *s*[2..5] (hier "long")

```
if (s.startsWith("abc")) ...
```

liefert true, falls *s* mit "abc" beginnt

```
if (s.endsWith("abc")) ...
```

liefert true, falls *s* mit "abc" ended

# Aufbauen von Strings



## aus Stringkonstante

```
String s = "very simple";
```

## aus char-Array

```
char[] a = {'a', 'b', 'c', 'd', 'e'};
```

```
String s1 = new String(a); // s1 enthält Kopie der Zeichen in a
```

```
String s2 = new String(a, 2, len); // s2 enthält Kopie von a[2..2+len-1]
```

# Aufbauen von Strings aus StringBuffer



**aus StringBuffer** (Bibliothekstyp wie *String*, aber modifizierbar)

```
StringBuffer b;  
b = new StringBuffer();           // erzeugt leeren StringBuffer der Länge 0  
  
len = b.length();  
b.append(x);                     // hängt x an b an. x kann beliebigen Typ haben:  
                                 // short, int, long, float, double, char, char[], String, boolean  
b.insert(pos, x);                // fügt x an der Stelle pos ein (Typ von x beliebig)  
b.delete(from, to);              // löscht [from..to[ aus b  
b.replace(from, to, "abc");      // ersetzt b[from, to[ durch "abc"  
  
ch = b.charAt(i);                // wie bei String  
s = b.substring(from, to);  
  
b.setCharAt(pos, 'x');           // setzt b[pos] auf 'x'  
  
s = b.toString();                // liefert Pufferinhalt als String
```

# Stringkonversionen



```
int i = new Integer("123").intValue(); // String ⇒ int
float f = new Float("3.14").floatValue(); // String ⇒ float
```

```
String s;
s = String.valueOf(123); // int ⇒ String
s = String.valueOf(3.14); // float ⇒ String
```

```
char[] a = s.toCharArray(); // String ⇒ char[]
String s = new String(a); // char[] ⇒ String
```

# Beispiel: Manipulation von Dateipfaden



`dir1\dir2\name.java`  $\Rightarrow$  `name.class`

- Verzeichnisse entfernen
- ".java" auf ".class" ändern (bzw. ".class" anhängen)

```
static String strip (String path) {  
    StringBuffer b = new StringBuffer(path); // erzeugt StringBuffer mit path als Inhalt  
    if (path.endsWith(".java")) {  
        int len = path.length();  
        b.delete(len-5, len);  
    }  
    b.append(".class");  
    int i = path.lastIndexOf("\\");  
    if (i >= 0) b.delete(0, i+1);  
    return b.toString();  
}
```

# Beispiel: Wörter aus einem Text lösen



Eingabe: "Ein Text aus Woertern ..."

Ausgabe:

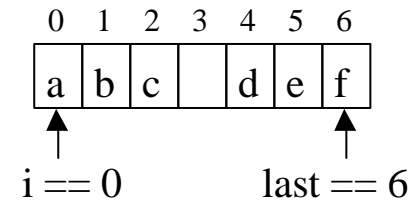
Ein

Text

aus

Woertern

```
static void printWords (String text) {  
    int i = 0, last = text.length() - 1;  
    while (i <= last) {  
        //--- skip nonletters  
        while (i <= last && !Character.isLetter(text.charAt(i))) i++;  
        // end of text or text[i] is a letter  
        //--- read word  
        int beg = i;  
        while (i <= last && Character.isLetter(text.charAt(i))) i++;  
        // end of text or text[i] is not a letter  
        //--- print word  
        if (i > beg) Out.println(text.substring(beg, i));  
    }  
}
```

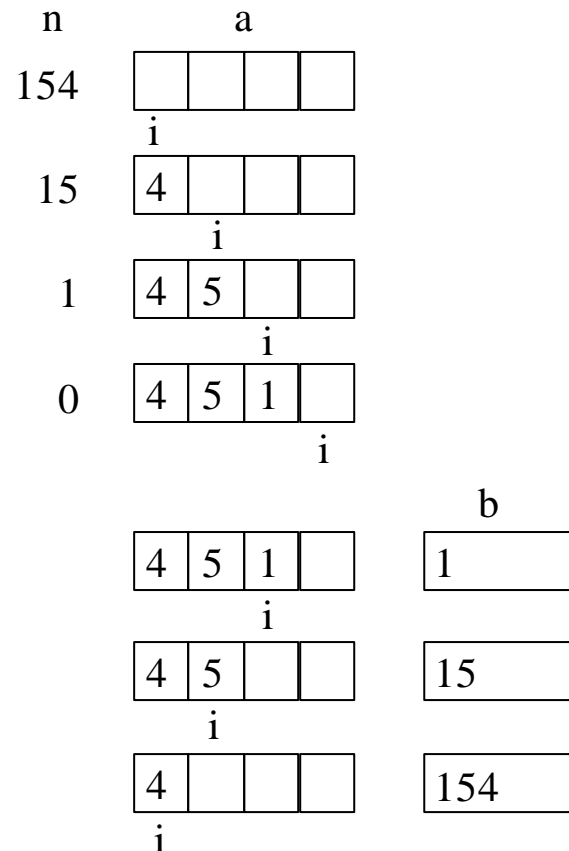


# Beispiel: Zahl in String konvertieren



Idee: Ziffern mit  $n \% 10$  abspalten und in *char*-Array sammeln

```
static String valueOf (int n) {  
    char[] a = new char[20];  
    int i = 0;  
    do {  
        a[i] = (char) (n % 10 + '0');  
        n = n / 10;  
        i++;  
    } while (n > 0);  
    StringBuffer b = new StringBuffer();  
    do {  
        i--;  
        b.append(a[i]);  
    } while (i > 0);  
    return b.toString();  
}
```



- Umdrehen von Domainnamen in Mailadressen
- `test@test1.test2.net` => `net.test2.test1`
- Funktion main erhält Adresse als Parameter
- Run length encoding: codiere Zeichen mit mehrfachem Vorkommen als Zeichen und Häufigkeit (benutzt in .tif und .pcx-Dateien)
- Beispiel: `AAACCBDDDD` => `A3C2B1D4`
- Funktion main erhält String als Parameter