

# Zeichen



## Datentyp char

```
char ch = 'c';
```

Zeichenvariable

Zeichenkonstante  
(unter einfachen Hochkommas)

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

### Zeichencodes

**ASCII** (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	

## Spezielle Zeichen

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





# Zeichen-Operationen

## 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 - '0') // Ergebnistyp: int
```



# 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;  
}
```

Schreibtestest: Eingabe 123+

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

"Horner-Schema"



# Standardfunktionen mit Zeichen

<code>if (Character.isLetter(ch)) ...</code>	true, wenn <i>ch</i> ein Unicode-Buchstabe ist
<code>if (Character.isDigit(ch)) ...</code>	true, wenn <i>ch</i> eine Ziffer ist
<code>if (Character.isLetterOrDigit(ch)) ...</code>	
<code>if (Character.isJavaIdentifierStart(ch)) ...</code>	true, wenn ein Java-Name mit <i>ch</i> beginnen kann
<code>if (Character.isJavaIdentifierPart(ch)) ...</code>	true, wenn <i>ch</i> in einem Java-Namen vorkommen kann
<code>if (Character.isLowerCase(ch)) ...</code>	true, wenn <i>ch</i> ein Kleinbuchstabe ist
<code>if (Character.isUpperCase(ch)) ...</code>	true, wenn <i>ch</i> ein Großbuchstabe ist
<code>ch1 = Character.toUpperCase(ch2);</code>	wandelt <i>ch2</i> in einen Großbuchstaben um
<code>ch1 = Character.toLowerCase(ch2);</code>	wandelt <i>ch2</i> in einen Kleinbuchstaben um

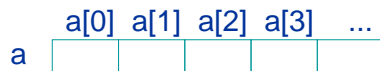


# Arrays



# 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

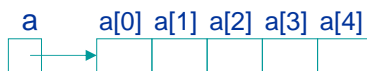
```
int[] a;  
float[] b;
```

- 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)



# Arbeiten mit Arrays

## 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

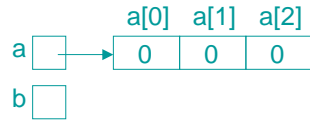
```
for (int i = 0; i < a.length; i++) // Array einlesen  
    a[i] = In.readInt();
```

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



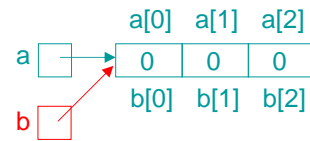
# Arrayzuweisung

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



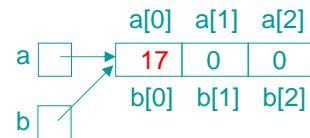
Arrayelemente werden in Java standardmäßig mit 0 initialisiert

```
b = a;
```



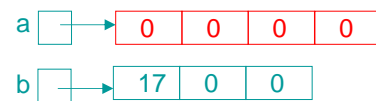
*b* bekommt denselben Wert wie *a*.  
Arrayzuweisung ist in Java **Zeigerzuweisung!**

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



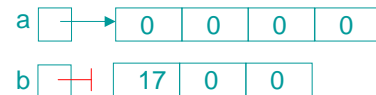
ändert in diesem Fall auch *b*[0]

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



*a* zeigt jetzt auf neues Array.

```
b = null;
```



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

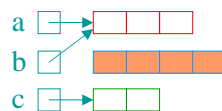
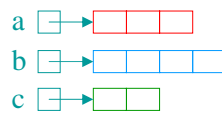


# Freigeben von Arrayspeicher

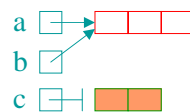
## Garbage Collection (Automatische Speicherbereinigung)

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};
```

primes 

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;
```

a   
b 

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

a   
b 

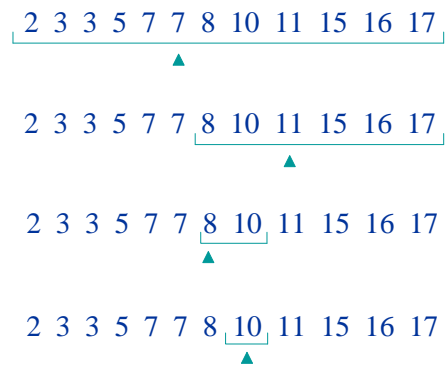
Typumwandlung nötig, da `clone` etwas vom Typ `Object[]` liefert





# Binäres Suchen

```
static int binarySearch (int[] a, int x) {
    int low = 0;
    int high = a.length - 1;
    while (low <= high) {
        int m = (low + high) / 2;
        if (a[m] == x) return m;
        else if (x > a[m]) low = m + 1;
        else /* x < a[m] */ high = m - 1;
    }
    /* low > high*/
    return -1;
}
```



- Suchraum wird in jedem Schritt halbiert
- bei  $n$  Arrayelementen sind höchstens  $\log_2(n)$  Schritte nötig, um jedes Element zu finden

n	seq.Suchen	bin.Suchen
10	10	4
100	100	7
1000	1000	10
10000	10000	14



# 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];
    int i, j;
    for (i = 2; i <= max; i++) sieve[i] = true;
    i = 2;
    while (i <= max) {
        Out.print(i + " "); // i is prime
        for (j = i; j <= max; j = j + i) sieve[j] = false;
        while (i <= max && !sieve[i]) i++;
        // i > max || sieve[i] == true
    }
}
```



# 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;
}
```

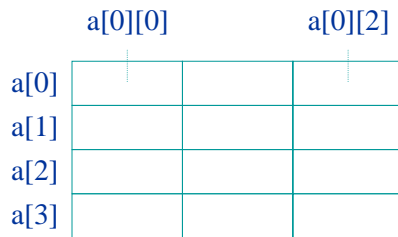
Besser mit Tabelle

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

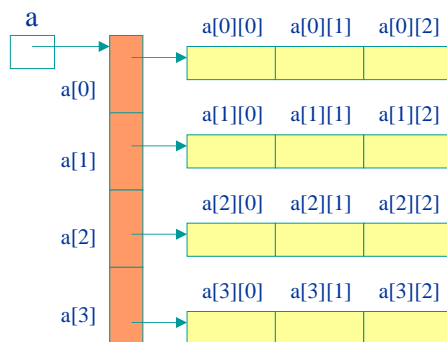


# Mehrdimensionale Arrays

Zweidimensionales Array  $\equiv$  Matrix



In Java als Array von Arrays implementiert



Deklaration und Erzeugung

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

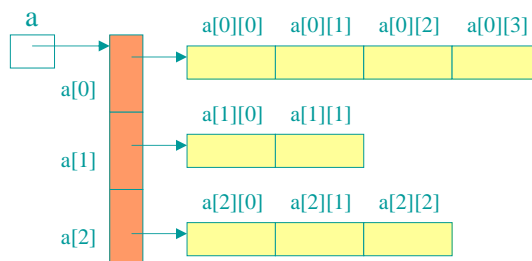
Zugriff

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



# Mehrdimensionale Arrays

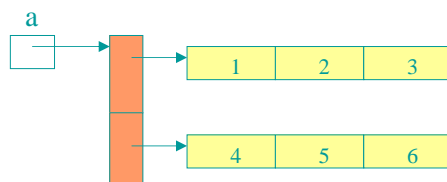
Zeilen können unterschiedlich lang sein (das ist aber selten sinnvoll)



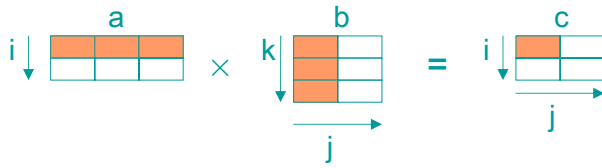
```
int[][] a = new int[3][];  
a[0] = new int[4];  
a[1] = new int[2];  
a[2] = new int[3];
```

Initialisierung

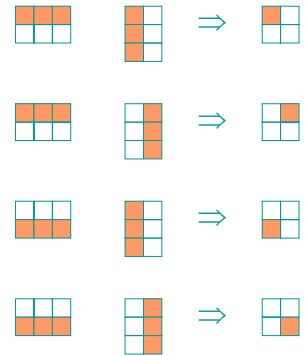
```
int[][] a = {{1, 2, 3}, {4, 5, 6}};
```



# Beispiel: Matrixmultiplikation



```
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[i].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;  
}
```



# Zeichen-Arrays

## Zeichen-Arrays

```
char[] s = new char[20]; // initialisiert alle Elemente mit '\u0000',
```

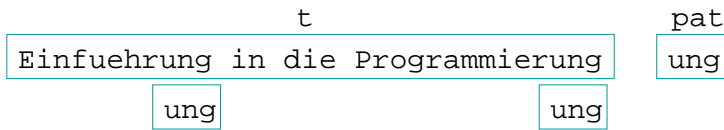
```
char[] t = {'a', 'b', 'c'};
```



## 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;
}
```



## Strings



# Datentyp String

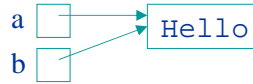
```
String a, b;
```

Bibliothekstyp für Zeichenarrays

```
a = "Hello";
```

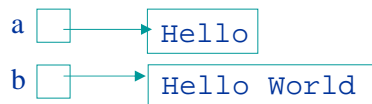
Stringkonstante (unter doppelten Hochkommas)

```
b = a;
```



Stringvariablen sind Zeiger auf Stringobjekte  
Stringzuweisung ist eine Zeigerzuweisung  
Stringobjekte sind nicht als Arrays ansprechbar  
Stringobjekte sind nicht veränderbar

```
b = a + " World";
```



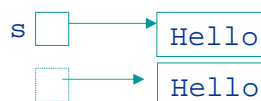
Verkettung mit "+" erzeugt neues Stringobjekt (relativ teure Operation)



# Stringvergleiche

```
String s = In.readName(); // liest einen Namen, z.B. Hello
```

```
if (s == "Hello") ... // liefert false! (Zeigervergleich)
```



weil zwei verschiedene Objekte,  
trotz gleichem Inhalt

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

liefert Index des 1. Vorkommens von "ng" in *s* (hier 4) oder -1  
liefert Index des 1. Vorkommens von "ng" ab Index 5 (hier 11)  
geht auch mit *char*  
liefert Index des letzten Vorkommens von "ng",  
(Varianten wie oben)

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

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

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

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



# Aufbauen von Strings

## aus Stringkonstante

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

## aus char-Array

```
char[] a = new char[10];  
for (int i = 0; i < a.length; i++)  
    a[i] = In.read();  
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();
}
```

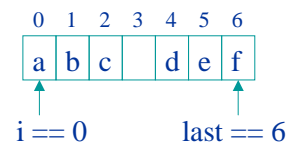


## Beispiel: Wörter aus einem Text lösen

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 of 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();
}
```

n	a				
154	<table border="1"><tr><td></td><td></td><td></td><td></td></tr></table>				
15	<table border="1"><tr><td>4</td><td></td><td></td><td></td></tr></table>	4			
4					
1	<table border="1"><tr><td>4</td><td>5</td><td></td><td></td></tr></table>	4	5		
4	5				
0	<table border="1"><tr><td>4</td><td>5</td><td>1</td><td></td></tr></table>	4	5	1	
4	5	1			



## 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
```

