Rahmen einer Anwendung

```
public class Anwendung {
  public static void main(String[] args) {
      ....
      ....
      ....
  }
}
```

muss in Datei **Anwendung.java** abgespeichert werden.

Unterscheidung zwischen Groß-/Kleinschreibung!!!

Dokumentationskommentar (Beispiel)

```
/**
 * <p>Kommentardemo </p>
 * <p>Mittels dieser Klasse soll javadoc demonstriert werden! </p>
 * @author Heinz Faßbender
 * @version 1.0
 * @see Anwendung1.java
 */
public class Kommentar {
 /**
  * Jetzt kommt eine Methode
  * @param param1 ist der erste Parameter
  * @param param2 ist der zweite Parameter
  * @return ist das Produkt
  */
   public int meth1(int param1, int param2) {
        return (param1 * param2);
    }
}
```

Schlüsselwörter in Java

abstract	default	if	private	throw
boolean	do	implements	protected	throws
break	double	import	public	transient
byte	else	instanceof	return	try
case	extends	int	short	void
catch	final	interface	static	volatile
char	finally	long	super	while
class	float	native	switch	
const	for	new	synchronized	
continue	goto	package	this	

Später: neue Schlüsselwörter ab Java 5.0

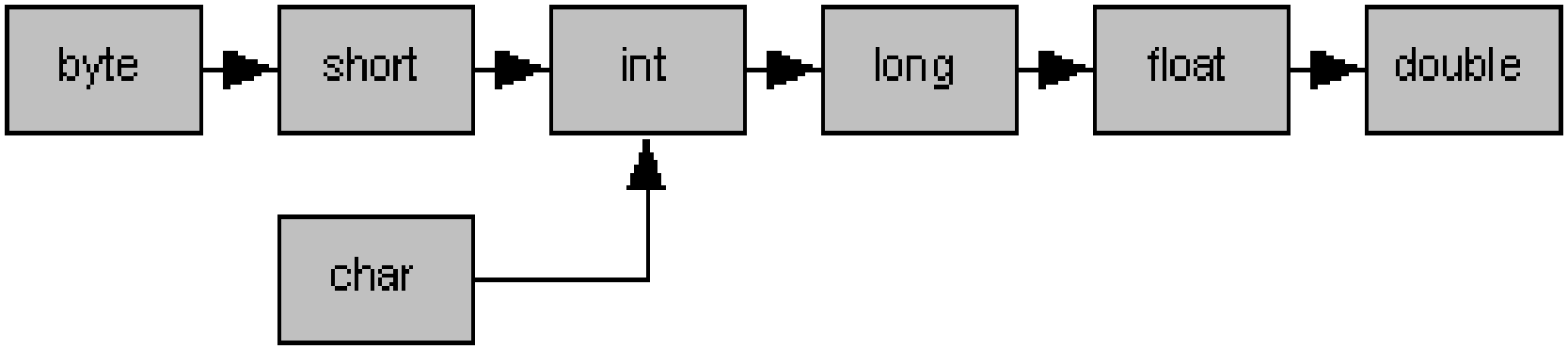
primitive Datentypen

Typname	Länge	Wertebereich	Default
boolean	1	true, false	false
char	2	Alle Unicode-Zeichen	\u0000
byte	1	$-2^7 \ldots 2^7-1$	0
short	2	$-2^{15} \ldots 2^{15}-1$	0
int	4	$-2^{31} \ldots 2^{31}-1$	0
long	8	$-2^{63} \ldots 2^{63}-1$	0
float	4	$+/-3.40282347 * 10^{38}$	0.0
double	8	$+/-1.79769313486231570 * 10^{308}$	0.0

Standard-Escape-Sequenzen

Zeichen	Bedeutung
\b	Rückschritt (Backspace)
\t	Horizontaler Tabulator
\n	Zeilenschaltung (Newline)
\f	Seitenumbruch (Formfeed)
\r	Wagenrücklauf (Carriage return)
\"	Doppeltes Anführungszeichen
\'	Einfaches Anführungszeichen
****	Backslash
\nnn	Oktalzahl nnn (kann auch kürzer als 3 Zeichen sein, darf nicht größer als oktal 377 sein)

Konvertierungen



- **erweiternd:** von links nach rechts
  Bsp:
  ```
  char c = 'h';
  int i = c;
  ```

- **einschränkend:** von rechts nach links
  Bsp:
  ```
  int i = 13;
  byte b = (byte) i;
  i = (int) 13.456
  // i wird 13 zugeordnet
  ```

arithmetische Operatoren

Op.	Bezeichnung	Bedeutung
+	Positives Vorzeichen	+n ist gleichbedeutend mit n
-	Negatives Vorzeichen	-n kehrt das Vorzeichen von n um
+	Summe	a + b -> die Summe von a und b
-	Differenz	a - b -> die Differenz von a und b
*	Produkt	a * b -> das Produkt aus a und b
/	Quotient	a / b -> den Quotienten von a und b
%	Restwert	a % b -> den Rest der ganzzahligen Division von a durch b auch auf Fließkommazahlen anwendbar
++	Präinkrement	++a -> a+1 und erhöht a um 1
++	Postinkrement	a++ -> a und erhöht a um 1
--	Prädekrement	--a -> a-1 und verringert a um 1
--	Postdekrement	a-- -> a und verringert a um 1

relationale Operatoren

Op.	Bezeichnung	Bedeutung
==	Gleich	a == b -> true, wenn a gleich b ist
!=	Ungleich	a != b -> true, wenn a ungleich b ist
<	Kleiner	a < b -> true, wenn a kleiner b ist
<=	Kleiner gleich	a <= b -> true, wenn a kleiner oder gleich b ist
>	Größer	a > b -> true, wenn a größer b ist
>=	Größer gleich	a >= b -> true, wenn a größer oder gleich b ist

logische Operatoren

Op.	Bezeichnung	Bedeutung
!	Logisches NICHT	!a -> false, wenn a wahr true, wenn a falsch
&&	UND mit SCE	a && b -> true, wenn a und b wahr false, wenn a falsch oder (a wahr & b falsch)
||	ODER mit SCE	a || b -> true, wenn a wahr oder (a falsch & b wahr) false, wenn a und b falsch
&	UND ohne SCE	a & b -> true, wenn a und b wahr false, wenn a oder b falsch (beide ausgewertet)
|	ODER ohne SCE	a | b -> true, wenn a oder b wahr (beide ausgewertet)
^	Exklusiv-ODER	a ^ b -> true, wenn a und b unterschiedlich

bitweise Operatoren

Op.	Bezeichnung	Bedeutung
~	Einerkomplement	alle Bits invertiert
|	Bitweises ODER	korrespondierende Bits ODER-verknüpft
&	Bitweises UND	korrespondierende Bits UND-verknüpft
^	Bitweises Exklusiv-ODER	korrespondierende Bits Exklusiv-ODER-verknüpft
>>	Rechtsschieben mit Vorz.	alle Bits von a um b Positionen nach rechts Falls höchstwertigste Bit gesetzt, auch höchstwertigste Bit des Resultats setzen
>>>	Rechtsschieben ohne Vorz.	alle Bits von a um b Positionen nach rechts. höchstwertigste Bit immer auf 0 gesetzt.
<<	Linksschieben	alle Bits von a um b Positionen nach links höchstwertigste Bit nicht gesondert behandelt

Zuweisungoperatoren

Op.	Bezeichnung	Bedeutung
=	Einfache Zuweisung	a = b weist a den Wert von b zu; Rückgabewert b
+=	Additionszuweisung	a += b weist a den Wert von a + b zu Rückgabewert a + b als.
-=	Subtraktionszuweisung	a -= b weist a den Wert von a - b zu Rückgabewert a - b
*=	Multiplikationszuweisung	a*= b weist a den Wert von a * b zu Rückgabewert a * b
/=	Divisionszuweisung	a /= b weist a den Wert von a / b zu Rückgabewert a / b
%=	Modulozuweisung	a %= b weist a den Wert von a % b zu Rückgabewert a % b
&=	UND-Zuweisung	a &= b weist a den Wert von a & b zu Rückgabewert a & b

Zuweisungoperatoren (Forts.)

Op.	Bezeichnung	Bedeutung
|=	ODER-Zuweisung	a |= b weist a den Wert von a | b zu Rückgabewert a | b
^=	Exklusiv-ODER-Zuweisung	a ^= b weist a den Wert von a ^ b zu Rückgabewert a ^ b
<<=	Linksschiebezuweisung	a <<= b weist a den Wert von a << b zu Rückgabewert a << b
>>=	Rechtsschiebezuweisung	a >>= b weist a den Wert von a >> b zu Rückgabewert a >> b
>>>=	Rechtsschiebezuweisung mit Nullexpansion	a >>>= b weist a den Wert von a >>> b Rückgabewert a >>> b

Typisierungen bei Vorrangregeln

- mögliche Operandentypen:
  - »N« numerische,
  - »I« integrale (also ganzzahlig numerische),
  - »L« logische,
  - »S« String-,
  - »R« Referenz-
  - »P« primitive Typen.
  - »A« alle Typen
  - »V« zeigt an, dass eine Variable erforderlich ist.

Vorrangregeln

Gr.	Operator	Typisierung	Ass.	Bezeichnung
1	++	N	R	Inkrement
	--	N	R	Dekrement
	+	N	R	Unäres Plus
	-	N	R	Unäres Minus
	~	I	R	Einerkomplement
	!	L	R	Logisches NICHT
	(type)	A	R	Type-Cast
2	*	N,N	L	Multiplikation
	/	N,N	L	Division
	%	N,N	L	Modulo
3	+	N,N	L	Addition
	-	N,N	L	Subtraktion
	+	S,A	L	String-Verkettung
4	<<	I,I	L	Linksschieben
	>>	I,I	L	Rechtsschieben
	>>>	I,I	L	Rechtsschieben mit Nullexp.

Vorrangregeln (Forts.)

Gr.	Operator	Typisierung	Ass.	Bezeichnung
5	<	N,N	L	Kleiner
	<=	N,N	L	Kleiner gleich
	>	N,N	L	Größer
	>=	N,N	L	Größer gleich
	instanceof	R,R	L	Klassenzugehörigkeit
6	==	P,P	L	Gleich
	!=	P,P	L	Ungleich
	==	R,R	L	Referenzgleichheit
	!=	R,R	L	Referenzungleichheit
7	&	I,I	L	Bitweises UND
	&	L,L	L	Logisches UND ohne SCE
8	^	I,I	L	Bitweises Exklusiv-ODER
	^	L,L	L	Logisches Exklusiv-ODER
9	|	I,I	L	Bitweises ODER
	|	L,L	L	Logisches ODER ohne SCE
10	&&	L,L	L	Logisches UND mit SCE

Vorrangregeln (Forts. 2)

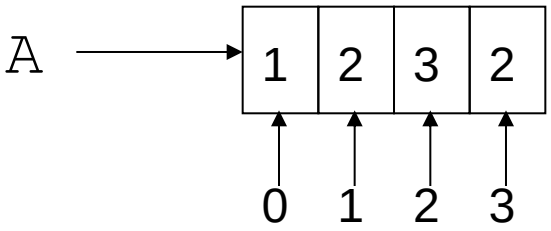
Gr.	Operator	Typisierung	Ass.	Bezeichnung
11	||	L,L	L	Logisches ODER mit SCE
12	?:	L,A,A	R	Bedingte Auswertung
13	=	V,A	R	Zuweisung
	+=	V,N	R	Additionszuweisung
	-=	V,N	R	Subtraktionszuweisung
	*=	V,N	R	Multiplikationszuweisung
	/=	V,N	R	Divisionszuweisung
	%=	V,N	R	Restwertzuweisung
	&=	N,N u. L,L	R	Bitweise (Logische) -UND-Zuw.
	|=	N,N u. L,L	R	Bitweise (Logische) -ODER-Zuw.
	^=	N,N u. L,L	R	Bitweise (Logische) -Exklusiv-ODER-Zuw.
	<<=	V,I	R	Linksschiebezuweisung
	>>=	V,I	R	Rechtsschiebezuweisung
	>>>=	V,I	R	Rechtsschiebezuweisung mit Nullexpansion

continue und break mit Label

```
loop1:
for (int i = 1; i <= 5; ++i)
{
  for (int j = 1; j <= 10; ++j)
  {
    System.out.println("Springen!");
    break loop1;
  }
}
```

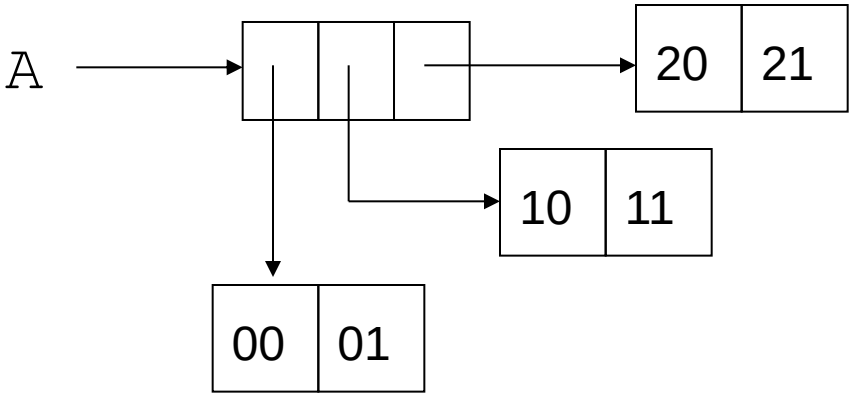
Felder

eindimensional




```
int i = A[1]*A[2];
-> i: 6
```

mehrdimensional



Deklaration und Initialisierung

1. Deklaration

```
int[] a;
```

a ⟶

2. Erzeugen eines Arrays und Zuweisung an Variable

```
a = new int[5];
```

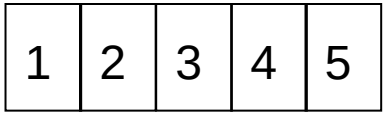
a ⟶ 

1. und 2. geht auch gemeinsam

```
int[] a = new int[5];
```

alternativ zu new kann man auch mit Literalen initialisieren

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

x ⟶ 

nicht rechteckiges Feld

```
public class NrFeld {

  public static void main(String[] args) {
    int[][] a = { {0},
                  {1,2},
                  {3,4,5},
                  {6,7,8,9}
                };

    for (int i=0; i < a.length; ++i) {
      for (int j=0; j < a[i].length; ++j) {
        System.out.print(a[i][j]);
      }
      System.out.println();
    }
  }
}
```

Wie herkömmliche Erzeugung und Initialisierung?

Erzeugung eines dreieckigen Feldes

```
// Erzeugung eines 1-dim. Feldes mit null-Zeigern
int[][] dreieck = new int[4][];
// Füllungszähler
int füllung = 0;

for (int i = 0; i < dreieck.length; i++) {
// Erzeugung der i-ten Zeile der Länge i
  dreieck[i] = new int[i+1];
// Belegung der Felder und Ausgabe
  for(int j = 0; j < i+1; j++) {
    dreieck[i][j] = füllung++;
    System.out.print(dreieck[i][j]);
  }
// Zeilenumbruch
  System.out.println();
}
```