

JAVA 02

Imperative Sprachelemente

JAVA 02 - Imperative Sprachelemente

1. Bezeichner

2. Variablen

3. Datentypen

4. Ausdrücke und Operatoren

5. Kontrollstrukturen

1. Bezeichner

- Was sind Bezeichner?

Kurz: *Wann immer der Programmierer selbst Namen vergibt, handelt es sich dabei um einen Bezeichner...*

- Die Regeln für Bezeichner:

- Beliebig viele **Unicode** (engl. Zeichensatz) Zeichen und Zahlen
- Erstes Zeichen = **Buchstabe**, **Unterstrich** oder **Dollarzeichen**
- **Keine Schlüsselwörter** (im Kurs durchgängig blau dargestellt)
- **CaseSensitive** = `anumber` ist etwas anderes als `aNumber` !

1. Bezeichner

Konventionen

- Es ist anzuraten, ausschließlich **englische** Bezeichner zu nutzen!
- Zusammengesetzte Wörter im „**CamelCase**“ – Format
- ***Variablen*** (und ***Methoden***) beginnen mit einem **Kleinbuchstaben**

„Gute Bezeichner erzählen was sie sind und was sie tun!“

2. Variablen

- Definition:

```
int aNumber;
```

- Initialisierung:

```
aNumber = 10;
```

- Explizite Initialisierung:

```
int anotherNumber = 20;
```

- Mehrere Variablen eines selben

Datentyps gleichzeitig definieren:

```
int age, size, weight;
```

- Und auch explizit:

```
int height = 10, width = 20;
```

3. Datentypen

3.1. Die einfachen („primitiven“) Datentypen

- Wahrheitswerte

```
boolean isReady = true;
```

- Ganzzahlen

```
int pos = 123, neg = -987;
```

- Fließkommazahlen (*englischer Punkt statt deutschem Komma!*)

```
float number = 12.34f;
```

- Zeichen (*nur genau EIN Zeichen in „einfachen“ Anführungszeichen*)

```
char letter = 'A';
```

3. Datentypen

3.2. Die 8 einfachen („primitiven“) Datentypen - Übersicht

Bezeichnung	Datentyp/Schlüsselwort	Wertebereich
Wahrheitswerte	boolean	false, true
Zeichen	char	a...z, A...Z, 0...9, !\"\$%&/ () ... Alle möglichen Zeichen
Ganze Zahlen	byte	-128 ... 127
	short	-32.768 ... 32.767
	int	-2.147.483.648 ... 2.147.483.647
	long	-9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807
Fließkommazahlen/ Gleitkommazahlen	float	$\sim 1,4 \cdot 10^{-45} \dots 3,4 \cdot 10^{38}$ Genauigkeit ca. 7 Stellen
	double	$\sim 4,9 \cdot 10^{-324} \dots 1,8 \cdot 10^{308}$ Genauigkeit ca. 15 Stellen

3. Datentypen

3.3. String und Ausgabe („Auf den Schirm!“)

String ist eine **Standard-Klasse**. Es ist KEIN „einfacher/primitiver“ Datentyp. Trotzdem wird sie so oft genutzt, dass Sie hier erwähnt wird.

```
String string1 = „Hello“;  
String string2 = „World“;  
String message = string1 + „ “ + string2;  
System.out.println(message); // Ausgabe!
```

WICHTIG: Vergleiche erfolgen nicht mit `==` sondern mit einer Methode Namens „equals“:

```
if (string1.equals(string2)) ...
```


4. Ausdrücke und Operatoren

4.1. Ausdrücke

- Ein **Ausdruck** ist eine Kombination von **Operatoren** und **Operanden**
- Das **Ergebnis** des Ausdrucks wird „nach links geschrieben“ (**zugewiesen**)

<code>int aNumber</code>	<code>=</code>	<code>2</code>	<code>+</code>	<code>8;</code>
<i>Ergebnis</i>	<i>=</i>	<i>Operand1</i>	<i>Operator</i>	<i>Operand2</i>

- Jeder Operator hat eine **Priorität**, wodurch die Reihenfolge mit der sie innerhalb eines Ausdrucks abgearbeitet werden, festgelegt ist.

4. Ausdrücke und Operatoren

4.2. Arithmetische Operatoren

Bezeichnung	Operator		Priorität
Positives Vorzeichen	+	10 (das Vorzeichen wird weggelassen)	1
Negatives Vorzeichen	-	-10	
Inkrementieren	++	<code>a++;</code> äquivalent zu: <code>a = a + 1;</code>	
Dekrementieren	--	<code>a--;</code> äquivalent zu: <code>a = a - 1;</code>	
Multiplikation	*	<code>int multiplied = a * b;</code>	2
Division	/	<code>float divided = a / b;</code>	
Rest	%	<code>int modulo = a % b;</code>	
Addition	+	<code>int addition = a + b;</code>	3
Subtraktion	-	<code>int subtraction = a - b;</code>	

4. Ausdrücke und Operatoren

4.3. Relationale Operatoren

Bezeichnung	Operator		Priorität
Kleiner	<	<code>boolean isSmaller</code> = <code>a < b;</code>	5
Kleiner oder gleich	<=	<code>boolean isSmallerOrEuqal</code> = <code>a <= b;</code>	
Größer	>	<code>boolean isGreater</code> = <code>a > b;</code>	
Größer oder gleich	>=	<code>boolean isGreaterOrEqual</code> = <code>a >= b;</code>	
Gleich	==	<code>boolean isEqual</code> = <code>a == b;</code>	6
Ungleich	!=	<code>boolean isUnequal</code> = <code>a != b;</code>	

4. Ausdrücke und Operatoren

4.4. Logische Operatoren

Bezeichnung	Operator		Priorität
NICHT	!	<code>boolean isTrue = !isFalse;</code>	1
UND („vollständige“ Ausw.)	&	<code>boolean isTrue = justTrue & trueToo;</code>	7
ODER („vollständige“ Ausw.)		<code>boolean isTrue = justTrue justFalse;</code>	9
UND („kurze“ Auswertung)	&&	<code>boolean isTrue = justTrue && trueTrue;</code>	10
ODER („kurze“ Auswertung)		<code>boolean isTrue = justTrue justFalse;</code>	11
exklusive ODER (XOR)	^	<code>boolean isTrue = justTrue ^ justFalse;</code> <code>boolean isFalse = justTrue ^ trueTrue;</code>	8

Der Unterschied von & zu && wird später erläutert

4. Ausdrücke und Operatoren

4.5. Bitoperatoren

Beispiele:

$8 \ll 2 = 32$

$8 \gg 2 = 2$

0000 1000 (also 8)

\ll

2

=

0010 0000 (also 32)

0000 1000 (also 8)

\gg

2

=

0000 0010 (also 2)

Was ist das binäre Zahlensystem?

0	0	0	0	0	0	0	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Beispiele:	Die Zahl	9	(= 8 + 1)	schreibt man binär	0000 1001
	Die Zahl	123	(= 64 + 32 + 16 + 8 + 2 + 1)	schreibt man binär	0111 1011

...eine Erwähnung der Vollständigkeit halber - Thema kommt nicht weiter vor...

4. Ausdrücke und Operatoren

4.6. Zuweisungsoperatoren

Bezeichnung	Operator		Priorität
Einfache Zuweisung	=	Das Ergebnis des Ausdrucks (rechts vom Zuweisungsoperator) wird einer Variablen (links vom Zuweisungsoperator) zugeordnet	13 (immer die letzte Aktion)
Kombinierte Zuweisung	op=	<code>int aNumber = aNumber + anotherNumber;</code> <i>Vom Ergebnis her dasselbe wie:</i> <code>int aNumber += anotherNumber;</code> Mögliche bei: * / % + - und Bitoperatoren...	

4. Ausdrücke und Operatoren

4.7. Cast-Operator

(Priorität = 1)

```
int x = 13;
int y = 5;
long bigResult = x * y;
int result = bigResult; // FEHLER!
int result = (int) bigResult; // OK
```

```
float f = 10.123
int i = f;           // FEHLER !!!
int i = (int) f;     // OK
```

*Problemlos, wenn Ergebnistyp „größer“
Absicht bei „kleineren“ Ergebnissen angeben*

Umwandlung der Fließkommazahl **10.123**
in die Ganzzahl **10** erfolgt NICHT automatisch.

5. Kontrollstrukturen

5.1. Das Ende eines Ausdrucks ;

- Mit einem **Semikolon** wird das Ende eines Ausdrucks definiert

5.2. Block {...}

- Die Grenzen von Klassen, Methoden und anderen Kontrollstrukturen
- Zusammenfassung mehrerer aufeinander folgenden Ausdrücke.
- Alle Variablen innerhalb eines Blocks, sind nur dort gültig.

5. Kontrollstrukturen

5.3. Verzweigungen mit Bedingung (1/2)

Einfachste Form der Bedingung – der Block wird nur ausgeführt, wenn Bedingung „wahr“:

```
if (a < b) {  
    System.out.println(„a is smaller than b“);  
}
```

Alternative – was soll ich „sonst“, wenn der erste Ausdruck „falsch“ ist:

```
if (a > b) {  
    System.out.println(„a is bigger than b“);  
} else {  
    System.out.println(„a is NOT bigger than b“);  
}
```

5. Kontrollstrukturen

5.3. Verzweigungen mit Bedingung (2/2)

```
if (a < b) {  
    System.out.println("a is smaller");  
} else if (a > b) {  
    System.out.println("a is bigger");  
} else {  
    System.out.println("a & b are equal");  
}
```

5. Kontrollstrukturen

5.4. Verzweigung mit dem Bedingungsoperator

(Priorität = 12)

Bedingung ? Ausdruck1 : Ausdruck2

Ist die **Bedingung wahr**, wird **Ausdruck1** ausgewertet.

Ansonsten wird **Ausdruck2** ausgewertet.

Verschachteln vermeiden!!!

b3 ? b2 ? b1 ? a1 : a2 : a3 : a4 ;

// ???????

5. Kontrollstrukturen

5.6. Verzweigungen abhängig von immer derselben Variable

Die Ausdrücke links und rechts sind äquivalent:

```
if (aNumber == 1) {  
    aNumber += 2;  
} else if (aNumber == 4) {  
    aNumber--;  
} else {  
    aNumber = 3;  
}
```

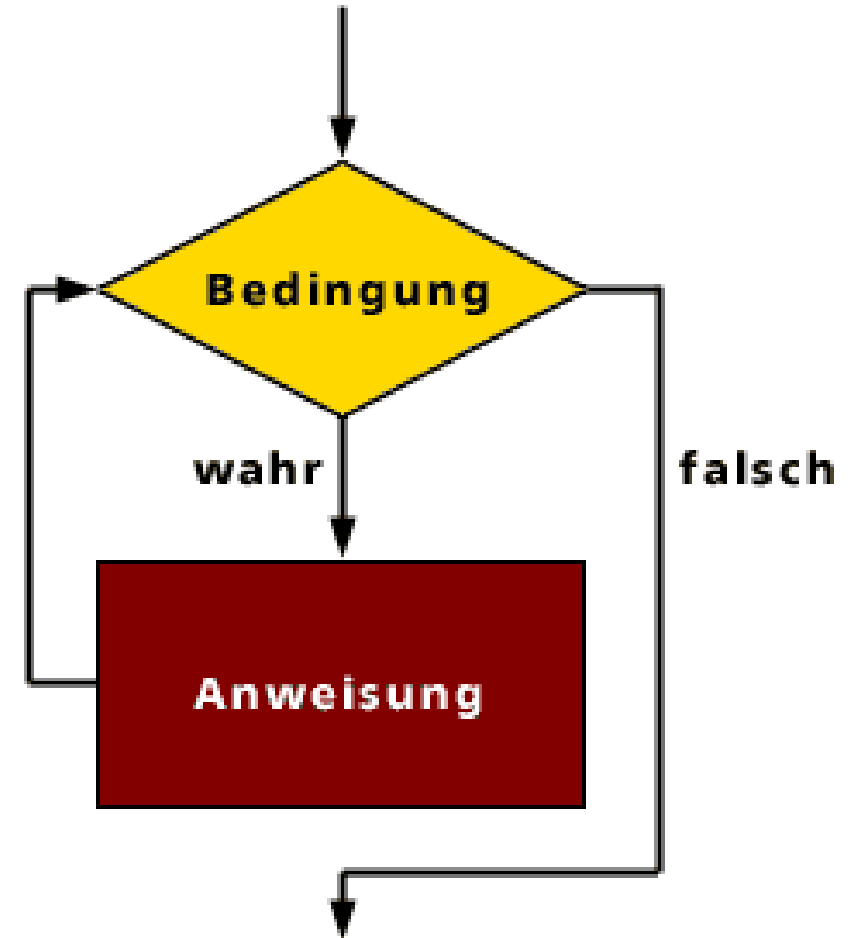
```
switch (aNumber) {  
    case 1:  
        aNumber += 2;  
        break;  
    case 4:  
        aNumber--;  
        break;  
    default:  
        aNumber = 3;  
        break;  
}
```

5. Kontrollstrukturen

5.7. Schleifen (1/4) „while“

```
int i = 1;  
while (i < 10) {  
    System.out.println(i++);  
}
```

```
while (Bedingung) {  
    Anweisung  
}
```

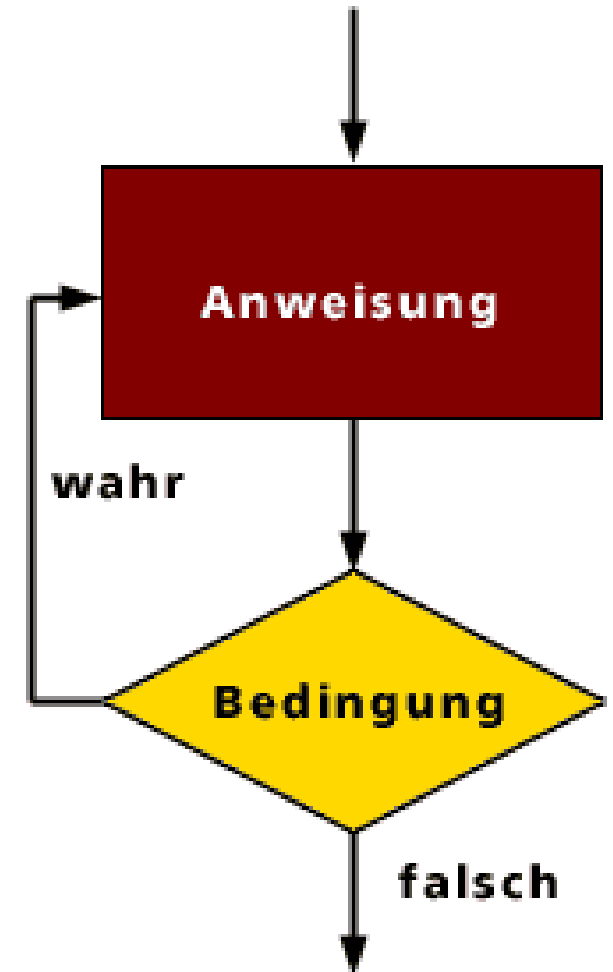


5. Kontrollstrukturen

5.7. Schleifen (2/4) „do“

```
int i = 1;  
do {  
    System.out.println(i++);  
} while (i < 10);
```

```
do {  
    Anweisung  
} while (Bedingung);
```

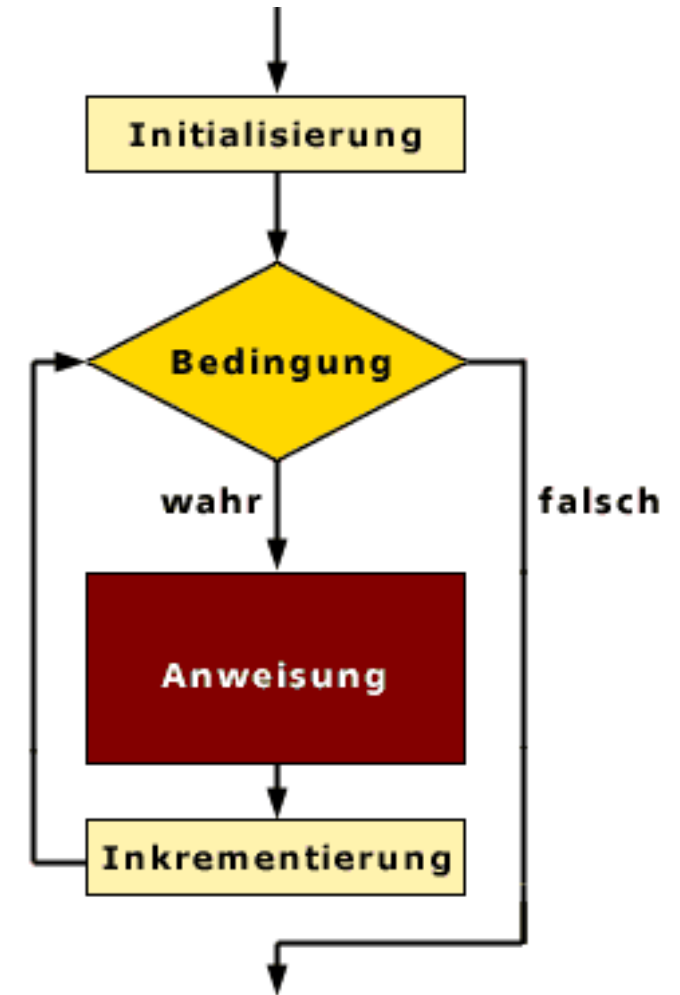


5. Kontrollstrukturen

5.7. Schleifen (3/4) „for“

```
for (int i = 1; i < 10; i++) {  
    System.out.println(i);  
}
```

```
for (Init; Bedingung; Update) {  
    Anweisung  
}
```



5. Kontrollstrukturen

5.7. Schleifen (4/4) *Sprunganweisungen*

```
boolean justOddNumbers;  
int externalMaxValue;  
// [...]  
for (int i = 1; i < 10; i++) {  
    if (justOddNumbers && i % 2 != 0) {  
        continue;           // naechster Schleifendurchlauf  
    }  
    if (i > externalMaxValue) {  
        break;               // Schleife komplett abbrechen!  
    }  
    System.out.println(i);  
}
```


Der Unterschied von & und &&

```
boolean x = trueOrFalse; // „trueOrFalse“ ist wahr ODER falsch...
boolean isUnsafe = x    &&    a > b++;
boolean isSafe   = x    &    a > b++;
```

&& = Ausdruck wird **beim ersten false abgebrochen**. **b** wird NICHT inkrementiert!

& = Ausdruck wird **vollständig ausgewertet** und **b** ist danach um **1** höher!

```
int x = 0;
while (x < 3) {
    if (false & x++) {
        System.out.println(x);
    }
}
```

```
int x = 0;
while (x < 3) {
    if (false && x++) {
        System.out.println(x);
    }
}
```

Die linke Schleife wird beendet, die rechte läuft endlos weiter...