BKO-Unterrichtsinhalte

JAVA – INFO

Quellen:

Java-Einfuehrung
Leitprogramm_Java
Java-Skript-V7

Datentypen

Datentyp	in JAVA	Beispiel
ganze Zahlen	***int***	1, -2, 5
Dezimalzahlen	***double***	4,12, -2,34
Wahrheitwerte	***boolean***	true, false
Zeichen	***char***	‚a', ‚+',
Text	***String***	„Dies ist ein Text"

Verkettung

Beim Datentyp String hat der Operator **+** eine andere Bedeutung. Er verbindet zwei Strings. Bsp.: „Hal" **+** „lo" wird zu „Hallo".

Vergleichsoperatoren

Um Bedingungen zu formulieren werden Vergleichoperatoren benötigt. Ein Vergleich liefert ***true*** oder ***false*** zurück.

< kleiner
<= kleiner oder gleich
> größer
>= größer oder gleich
== gleich
!= ungleich

UND / ODER

Vergleichsoperationen können durch ***und*** bzw. durch ***oder*** verknüpft werden.

oder **||** ist dann true, wenn mind. eine Bedingung true ist.

und **&&** ist dann true, wenn beide Bedingungen erfüllt sind.

Aufgaben

Schreibe die Aufgabestellung ab und löse:

AB_JAVA_Datentypen und Vergleichsoperatoren

Kontrollstrukturen

Der Programmcode wird normalerweise sequentiell ausgeführt. Jedoch kann durch Kontrollstrukturen bzw. durch die Formulierung von Bedingungen die Ausführung gesteuert werden. Dieses geschieht mittels Verzweigungen und unterschiedlichen Arten von Schleifen.

Art	Typ	Erläuterung	Syntax

Verzweigungen	if-Anweisung	Diese entscheidet, ob ein nachfolgender Anweisungsblock ausgeführt werden soll oder nicht.	if (Bedingung) {Anweisungen;
			
			}

	if-else-Verzweigung	Die if-Verzweigung kontrolliert nicht nur die Ausführung einer einzelnen Anweisung bzw. eines Anweisungsblocks; man kann auch alternativ einen Anweisungsblock A oder einen Anweisungsblock B ausführen lassen. Dazu wird eine if-Konstruktion um eine else-Konstruktion erweitert	if (Bedingung) {Anweisung(en);
			
			}
			
			else {
			
			Anweisung(en);
			
			}

	Bedingungsoperator ?:	If-else-Verzweigungen können mithilfe des Bedingungsoperators ?: ausgedrückt werden. Dieser erwartet als einziger Opera-	Bedingung ? Ausdruck1: Ausdruck2;

		tor drei Operanden: eine Bedingung und zwei Ausdrücke, von denen je nach Ergebnis der Bedingung einer ausgeführt wird.	
	switch-Verzweigung	Diese dient dazu, den Wert einer numerischen Variablen/ Ausdrucks/ Strings mit einer Reihe von Konstanten zu vergleichen.	switch (Ausdruck) {case CONST1: Anweisungen; break; case CONST2: Anweisungen; break; default: Anweisungen; break;

Schleifen	while-Schleife	Diese ist die allgemeinste Schleife. Jede Schleife besteht aus einem Schlüsselwort (in diesem Fall while), eine Bedingung und einem Anweisungsblock.	while (Bedingung) {Anweisungen;
			
			}

	do-while-Schleife	Diese ist mit der while-Schleife eng verwandt. Der einzige Unterschied besteht darin, dass bei der while-Schleife die Schleifenbedingung bereits vor dem ersten Ausführen des Anweisungsblocks getestet wird, während die do-while-Schleife den Anweisungsblock einmal ausführt und erst dann die Bedingung überprüft.	Initialisierung;do { Anweisung(en) inklusive Veränderung; } while (Bedingung);

	for-Schleife	Die drei Anweisungen zur Bearbeitung der Schleifenvariablen (Initialisierung, Bedingung und Änderung) werden im Schleifenkopf zusammengezogen. Das macht diese Schleife sehr übersichtlich.	for (Initialisierung; Bedingung; Veränderung) {Anweisungen;
			
			}

	for-each-Schleife	Diese ist eine Variante der for-Schleife. Mit ihr können	// gegeben ist ein Array ‘zahlen‘ mit 10 Integer-Elementen.// Werte der Elemente

		alle Elemente eines Arrays, einer enum-Aufzählung, einer Collection bzw. ganz allgemein jedes Objekts, das die Schnittstelle Iterable implementiert, durchlaufen werden.	mit ‘for-each‘-Schleife ausgeben
			
			for(int i : zahlen) {
			
			System.out.println(i);
			
			}

Tabelle 3: Übersicht über wichtige Kontrollstrukturen.

Schleifen

```
while (boolean expression) {statement(s)}
```

Die Bedingung (boolean expression) wird am Schleifenanfang geprüft. Die Anweisungen in der Schleife werden gegebenenfalls nicht ein einziges Mal ausgeführt.

```
do {statement(s)} while (boolrean expression);
```

Die Bedingung (boolean expression) wird am Schleifenende geprüft. Die Anweisungen in der Schleife werden immer mindestens ein Mal ausgeführt.

```
for (initialization; boolean expression; increment) { statement(s)}
```

Die Bedingung (boolean expression) wird am Schleifenanfang geprüft. Die Anweisungen in der Schleife werden gegebenenfalls nicht ein einziges Mal ausgeführt.

Bedingte Ausführung und Alternativen

```
if (boolean expression) {statement(s)}
```

Die Anweisungen (statement(s)) werden nur ausgeführt, wenn die Bedingung wahr ist.

```
if (boolean expression) {statement(s) 1}       else {statement(s) 2}
```

Abhängig von der Bedingung werden alternativ nur die ersten Anweisungen (statement(s) 1) oder die zweiten Anweisungen (statement(s) 2) ausgeführt.

```
if (boolean expression 1) {statement(s) 1}     else if (boolean
expression 2) {statement(s) 2} else if (boolean expression 3)
{statement(s) 3} else {statement(s) 4}
```

Abhängig von den Bedingungen (logical expression 1 to 3) werden alternativ nur die ersten An- weisungen (statement(s) 1), oder die zweiten Anweisungen (statement(s) 2), usw. aus- geführt.

```
switch (integer expression) {case integer expression 1: statement(s) 1
break;
case integer expression 2:statement(s) 2 break; ......
default:statement(s)break; }
```

Abhängig vom ganzzahligen Ausdruck (integer expression) wird zum zugehörigen Fall (case) ver- zweigt. Die Anweisungen werden ab dieser Stelle bis zum „break“ ausgeführt. Wenn keiner der Fälle zutrifft, wird der default-Zweig gewählt.

Weitere Kontrollanweisungen

continue: Die Anweisungen unterbricht die innerste Schleife und führt dazu, dass deren Schleifen- bedingung erneut ausgewertet wird.

break: Die Anweisung beendet die innerste „switch“, „for“, „while“ oder „do while“ Anweisung.

BEARBEITEN