Java Überblick 1/2

Datentypen

Typ	Beschreibung	Wertebereich / Beispiel
boolean	Boolescher Wert	true, false
char	einzelnes Zeichen (16 Bit)	alle Unicode-Zeichen
byte	ganze Zahl (8 Bit)	$-2^7 \dots 2^7 - 1$
short	ganze Zahl (16 Bit)	$-2^{15} \dots 2^{15} - 1$
int	ganze Zahl (32 Bit)	$-2^{31} \dots 2^{31} - 1$
long	ganze Zahl (64 Bit)	$-2^{63} \dots 2^{63} - 1$
float	Fließkommazahl (32 Bit)	3,14159f
double	Fließkommazahl (64 Bit)	$-1{,}79 * 10^{38}$
String	Zeichenkette	"Dies ist ein String."
int[]	ganzzahliges Feld (Array)	{3, 1, 4, 1, 5, 9}

Java kennt **primitive Typen** (boolean, char, …, double) und **Referenztypen** (Objekte, Strings und Arrays).

Variablendeklaration

```
(<Zugriffsart>) <Typ> <Bezeichner> (= <Wert>)
```

Beispiel	Erläuterung
`private int anzahl;`	Ganzzahlige Variable mit Namen *anzahl*
`int tage = 14;`	Ganzzahlige Variable mit Bezeichner *tage* und Zuweisung des Werts 14
`boolean gesund;`	Boolesche Variable mit Bezeichner *gesund*
`public String name;`	(Öffentlich zugängliche) Text-Variable mit dem Bezeichner *name*

Referenztypen (außer `String`) müssen mithilfe des `new`-Operators erzeugt werden:

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

Es wird ein leeres Feld mit dem Bezeichner *du* erzeugt, welches fünf ganzzahlige Werte aufnehmen kann.

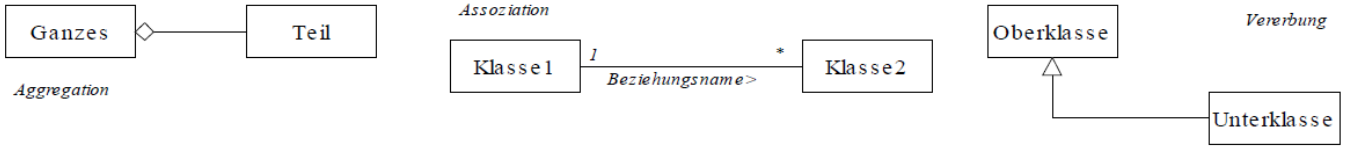
```
double[] messungen;
```

Deklaration eines Feldes mit Namen *messungen*. Bevor es jedoch Werte aufnehmen kann, muss es mit dem `new`-Operator erzeugt werden.

Operatoren

+	Addition		
-	Subtraktion		
*	Multiplikation		
/	Division	*	
%	Modulo	Divisonsrest	
++	Inkrement	i++ entspr. i = i+1	
--	Dekrement	i = i – 1	
==	Vergleich	=	Zuweisung**
<	Kleiner	<=	Kleiner gleich
>	Größer	>=	Größer gleich
!=	Ungleich	!	logisches NICHT
||	logisches ODER	&&	logisches UND

* x / y ergibt den Quotienten von x und y. Sind x und y ganzzahlig, so ist auch x / y ganzzahlig (z.B. 9 / 4 liefert 2).
** a == b liefert `true` oder `false`



Methodendefinition

```
(<Zugriffsart>) <Rückgabetyp> <Bezeichner> (<Parameter>) {…}
```

Beispiel

```
public void hello(String name){
       System.out.print("Hallo " + name);
}

public double umfang(double radius){
       return 2*radius*3,14159;
}

public void zubettgehen(){
       ausziehen();
       waschen();
       zaehneputzen();
       schlafenlegen();
}
```

Erläuterung

Die öffentliche Methode *hello* gibt auf dem Bildschirm „Hallo XYZ" aus, wenn ihr „XYZ" beim Aufruf übergeben wurde.

Die Methode gibt den Kreisumfang bei Übergabe des Radius zurück.

Die Methode *zubettgehen* hat keinen Rückgabewert, keine Parameter und ruft nacheinander die Methoden *ausziehen, waschen, zaehneputzen* und *schlafenlegen* auf.

Klassendefinition

```
(<Zugriffsart>) class <Bezeichner> (extends <Oberklasse>) {…}
```

Beispiel

```
public class Quadrat{

   //Attribute
   private int laenge;
   private String farbe;


   //Methoden
   Quadrat(int seitenl){
       laenge = seitenl ;
       farbe = "rot";
   }

   public double flaeche(){
       return laenge*laenge;
   }

} //Ende Quadrat
```

Erläuterung

Kopf: **bei Vererbung** class Unterkl **extends** Oberkl, z.B. `class Quadrat extends Figur`
Deklaration der Attribute
Es werden ein ganzzahliges Attribut für die Seitenlänge sowie eine Text-Variable für die Füllffarbe des Quadrates deklariert.

Methodendefinitionen
Der Konstruktor zur Erzeugung des Objekts hat den gleichen Namen wie die Klasse selbst.

Methode, die als Rückgabewert den Flächeninhalt des Quadrats berechnet.

Ende der Klassendefinition

Java Überblick 2/2

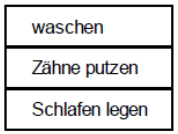
Sequenz

Jede Anweisung wird mit einem Semikolon abgeschlossen; Mehrere Anweisungen nacheinander ergeben eine Sequenz

Beispiel

```
waschen();
zaehneputzen();
schlafenlegen();
```

Struktogramm



Fallunterscheidung (bedingte Anweisung)

Die bedingte Anweisung gibt es mit oder ohne Alternative:

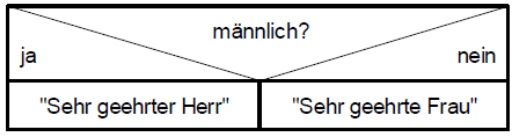
```
if (<Bedingung>) {
   <Anweisungen>
}
else {
   <Anweisungen>
}
```

```
if (<Bedingung>) {
   <Anweisungen>
}
```

Beispiel (mit Alternative):

```
if (geschl == 'm') {
   System.out.println("Sehr geehrter Herr");
}
else {
   System.out.println("Sehr geehrte Frau");
}
```

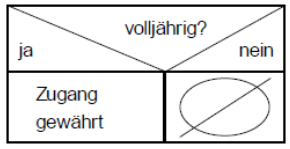
Struktogramm (mit Alternative):



Beispiel (ohne Alternative):

```
if (alter >= 18) {
   zugang = true;
}
```

Struktogramm (ohne Alternative):



Wiederholung mit fester Anzahl

```
for (<Init>; <Bedingung>; <Update>) {
      <Anweisungen>
}
```

`<Init>`	Deklaration einer ganzzahligen Zählvariablen und Zuweisung ihres Anfangswertes (z.B. `int i = 0`).
`<Bedingung>`	Solange die Bedingung (abhängig von der Zählvariablen) erfüllt ist, werden nachfolgende Anweisungen ausgeführt.
`<Update>`	Das Update erfolgt nach jedem Durchlauf und ändert die Laufvariable entsprechend der angegebenen Zuweisung (oft `i++`).

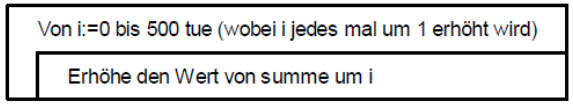
Beispiel

```
int summe = 0;

for (int i = 0; i <= 500; i++) {
      summe = summe + i;
}
```

Struktogramm und Erläuterung

Berechnet die Summe aller ganzen Zahlen von 0 bis 500. Als Zählvariable wird die ganze Zahl *i* deklariert, ihr Anfangswert ist 0. Die Anweisung wird solange wiederholt, bis *i* den Wert 500 erreicht, wobei *i* bei jedem Durchlauf um 1 erhöht wird.



Wiederholung mit Anfangsbedingung

```
while (<Bedingung>) {
      <Anweisungen>
}
```

Die Bedingung wird vor der Ausführung der Anweisungen getestet, so dass nachfolgende Anweisungen eventuell gar nicht ausgeführt werden.

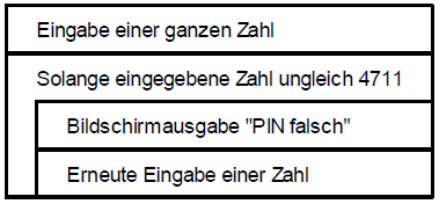
Beispiel

```
int pin = eingabe();

while (pin != 4711) {
      System.out.println("PIN falsch");
      a = eingabe();
}
```

Struktogramm und Erläuterung

Zuerst wird eine Variable pin vom Typ Integer deklariert. Die Zuweisung erfolgt über eine Methode eingabe(), welche ermöglicht, eine ganze Zahl über die Tastatur einzugeben und diese als Rückgabewert liefert. Solange pin nicht den Wert 4711 hat, wird der Fehler auf dem Bildschirm ausgegeben und zur erneuten Eingabe einer Zahl aufgefordert.



Datenstruktur Feld (Array)

- Die Datenstruktur Array speichert (beliebig) viele Elemente desselben Datentyps.

Deklaration + Initialisierung: `Datentyp [] Bezeichner = new Datentyp [Länge]`

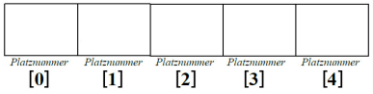
```
FIGUR[]gegners = new FIGUR[5];  int[]zahlen = new int[100];   String[]namen = new String [26];
```

Nach der Erstellung des Arrays sind die Einträge noch leer (null bzw. 0 bzw. 0.0). Über den **Index und die Länge(length)** können die einzelnen Elemente angesprochen werden. (Index 0 ≙ erstes und Index length-1 ≙ letztes Element)

```
gegners[0] = new FIGUR();   gegners[4] = new FIGUR();   gegners[0].Schritt();   zahlen[0] = 10;
namen[25] = "Zeus";
```

Ein Array zu **iterieren (= durchlaufen)** bedeutet, für alle Mitglieder dieser Sammlung eine (ähnliche) Aktion auszuführen.

```
for(int i = 0; i<5; i++){
      gegners[i] = new FIGUR(); gegners[i].setzteMittelpunkt(50, i*50);
}
```




```
for(int i = 0; i<5; i++){ \\ Elemente quadrieren
      zahlen[i] = zahlen[i]*zahlen[i];
}
```