Variablen und Datentypen

Typ	Deklaration	Wertebereich
Ganzzahl	byte b = - 1;	-128 < b < 127
Ganzzahl	short s=-255;	-32768 < s < 32767
Ganzzahl	integer i = 0;	-2147483648 < i < 2147483647
Ganzzah	long ℓ = 0;	-9223372036854775808<ℓ<9223372036854775807
Dezimalzahl	float f = 0.1;	f = -3.40282347e38 < f < 3.40282347e38
Dezimalzahl	double d = -0.1 ;	-1.79769313486231570e308 < d < 1.79769313486231570e308
Logiktyp (Boolescher Wert)	boolean= true;	‘‘true’’ oder „false“
Zeichen	char c = ‚A’;	Alle Zeichen des Uni-Codes (in JAVA 2 Byte Länge)
Zeichenkette	String s=“abc“;	„unbegrenzt“

Operatoren

	Arithmetische Operationen		**Logische Operatoren**
+	Addition	&&	logische UND
-	Subtraktion	||	logische ODER
*	Multiplikation	^	Exklusive ODER
/	Division	!	Negation
%	Modulo - Division		**Spezielle Operatoren**
		=	Wertzuweisung
	Vergleichsoperatoren	( )	Einklammern
==	gleich (nicht für Strings)	++	Inkrement um 1
!=	ungleich	- -	Dekrement um 1
<	kleiner	+ =	Inkrement um den Wert rechts vom Gleichheitszeichen
>	größer	- =	Dekrement um den Wert rechts vom Gleichheitszeichen
>=	größer gleich	* =	Multiplikation mit dem Wert rechts vom Gleichheitszeichen
<=	kleiner gleich	/ =	Division mit dem Wert rechts vom Gleichheitszeichen
equals ()	vergleicht zwei Objekte, z.B. zwei Strings	+	Verkettung von Strings

Einfache Ausgabe am Bildschirm

System.out.println(„*irgendein text*“ + *variablen*); System.out.print(„*Antwort*: \t “ + *a* +*b*);

Einfache Eingabe über Tastatur

String *s* = JOptionPane.showInputDialog(„*Wert?*“);
y = Console.readInt(„*Bitte Zahl eingeben!*“); *z* = Console.read.Double(„*Dezimalzahl?*“);

Einfache Programmstruktur

```
public class flaechenberechnung
{  public static void main()
    {  int laenge, breite;  double flaeche;
       laenge = 10; breite = 5;
       flaeche = laenge * breite;
      System.out.print("Der Flächeninhalt beträgt: " + flaeche));
     }
 }
```

Datentypumwandlung

double d = (double) i; // wandelt die ganze Zahl **i** in die Dezimalzahl d um
i = (int) ’A’ ; // macht aus dem Großbuchstabe **A** den zugehörigen Dezimalwert **65**
int *x* = Integer.parseInt(*s*); doubel *y* = Double.parseDouble(*s*); char *z* = Char.parseChar (*s*); // macht aus der Zeichenkette **s** den jeweils angegebenen Datentyp.
Bei **char** darf **s** natürlich nur ein Zeichen lang sein!

Kommentare und Blöcke

```
// in einer Zeile                  {  anweisung_1 ...
 /* über mehrere                      anweisung_n
  Zeilen */                        }
```

Dateiarbeit (Beispiele)

```
// Die folgenden Anweisungen lesen eine Textzeile aus der Datei „baumarchiv.txt“
File datei  = new File („baumarchiv.txt“);
BufferedReader lesen = new BufferedReader ( new FileReader(datei) );
String zeile = lesen.readLine();
// und schreiben eine Zeile in die Datei:
BufferedWriter schreiben = new BufferedWriter ( new FileWriter(datei) );
String zeile = schreiben.writeLine();
-----------------------------------------------------------------------------------------------------
// Die Klasse kopiert die Datei „testbild.bmp“ Byte-weise
import java.io.*.;
public class dateiarbeit
{ public static void main() throws Exception
  { kopiere(new FileInputStream("testbild.bmp"),new FileOutputStream
("testbildkopie.bmp"));
  }
  static void kopiere(InputStream fis, OutputStream fos )
  { try
    { byte  puffer[ ] = new byte[0x0001];  // 1 Byte Puffer
      int  nbytes;
      while ( (nbytes = fis.read(puffer)) != -1 ) fos.write( buffer, 0, nbytes );
    }
    catch( IOException e ) System.err.println( e );
   }
}
```

Modifizierer

static	Als static-deklarierte Elemente gehören keinem Objekt an, sondern der Klasse, in der sie definiert sind. Beispiel: static void *zeit* (...) { ... // auf die Methode „zeit“ kann von einer anderen Klasse / Objekt nicht zugriffen werden ... }
abstract	abstract class *fische* { abstract void *schwimmen*(); } class *fische_taetigkeiten* extends *fische* { void *schwimmen*() { System.out.println("*Wir schwimmen.*"); } } In der Basisklasse „fische“ wird angegeben, dass die Methode „schwimmen“ existiert, aber nicht wie sie implementiert ist. Damit wird vereinbart, dass jedes Objekt des Typs „fische“ - und damit auch jedes Objekt der davon abgeleiteten Klasse „fische_taetigkeiten“- die Methode „schwimmen“ besitzen soll. In der Klasse „fische_taetigkeiten“ muss die abstrakte Methode „schwimmen“ daher implementiert werden, ansonsten gibt es einen Compilerfehler.
void	Die Methode „wunschlos_gluecklich“ hat keine Daten zur (leeren) Rückgabe public void *wunschlos_gluecklich* ( )

private	Die so vereinbarte Variable oder Methode ist nur innerhalb der Klasse „sterne_zaehlen“ verwendbar. Beispiele:
	class *sterne_zaehlen* {
	private int *zaehler1*, *zaehler2*;
	private void *unsichtbar* ( ) { ... }
	}

public	Klasse, Methode oder Variable, die immer in anderen Klassen oder Methode verwendet werden können. Beipiel:
	public class *dackel* { ... }
	public void *fressen* ( ) { ...}
	public boolean *fressnapf_voll*;
	dackel *hunger* = new *hunger*( ); // Objekt „*hunger*“ kann überall, auch in anderen Klassen vereinbart und benutzt werden.
	hunger.fressen();

Vererbung

```
class Unterklasse extends Oberklasse
{ ...
}
```

Die sichtbaren Eigenschaften der „Oberklasse“ werden auf die Unterklasse durch das Schlüsselwort *extends* vererbt. Wenn sich die Implementierung einer Methode der Oberklasse ändert, so wird die Unterklasse diese Änderung mitbekommen.

Nebenläufigkeit / Multitasking (Threads)

Eine Klasse, deren Exemplare Programmcode parallel ausführen sollen (Multitasking), muss die Schnittstelle Runnable implementieren. Sie schreibt eine run()-Methode vor.

```
class DateThread implements Runnable
{ public void run()  // Methode gibt 1000-mal das Datum aus
  { for ( int i = 0; i < 1000; i++ )
      System.out.println( new Date() );
  }
}
class CounterThread implements Runnable
{  public void run()  // Methode gibt „parallel“ zur anderen Methode 1000-mal den Zähler aus
  { for ( int i = 0; i <1000; i++ )
      System.out.println( i );
  }
}
```

Weitere Thread-Zustände:
MeinThread.sleep(2000) – „MeinThread“ wartet 2 Sekunden in der Ausführung
suspend() - Thread wir gestoppt bis er über ..resume wieder gestartet wird
resume() - Thread wird wieder gestartet, nachdem er über .suspend gestoppt wurde
destroy() – Thread wir sofort beendet und das Thread-Objekt gelöscht

Ausnahmen (Exeptions / Fehlerbehandlung)

Eine Exeption wird in Java erzeugt wenn während der Laufzeit des Programmes ein Fehler aufgetreten ist. Bei systemnahen Routinen / Dateiarbeit ist die Fehlerbehandlung zwingend erforderlich. Beispiel:

```
try        // die Anweisungen im try-Block werden zuerst ausgeführt
{ meinSystem.exec(„notepad.exe“)
}
catch (IOExecption e) // die Anweisungen des catch-Blockes werden nur im Fehlerfalle des
                                  // try-Blockes ausgeführt
{ System.out.println(„Fehlenummer: „ + e);
}
finally
{   ...../ abschließend können noch die Anweisungen im finally-Block ausgeführt werden
}
```

Mathematische Methoden

```
double d = Math.random() // liefert Zufallszahl als Dezimalzahl  0<=d<1
double  wurzel = Math.sqrt(wurzelradikand);  // zieht Quadratwurzel
double p = Math.pow(basis,exponent); // potenziert x hoch y
double gerundete_zahl = Math.round( d * 100 ) / 100.0;
                        // rundet d auf zwei Stellen nach dem Komma
double gerundete_zahl = Math.round( d * 10000 ) / 10000.0;
                        // rundet d auf vier Stellen nach dem Komma
double p = Math.PI    // die Zahl π
```

Arbeiten mit Applets

Ein einfaches Beispiel *Welt.java* (hier wird nur paint() überschrieben um gezielt ab der Stelle 100,50 den Text ausgeben zu können):

```
import java.awt.*;
 import java.applet.Applet;
 public class Welt extends Applet
 { public void paint( Graphics g )
   { g.drawString( "Mein Java-Applet!",100,50 );    // 100 Pixel nach rechts und 50 nach unten
   }
 }
```

Die Datei Welt.java muss compiliert werden. Die entstandene Datei *Welt.class* muss im HTML-Code angegeben werden. Die dazugehörige HTML-Seite *applettest.html* könnte so aussehen. Die Größe des Applets ist mit 400x400 Pixeln angegeben:

```
 <html>
   <head>
     <title>Mein Java-Applet testen!</title>
   </head>
   <body>
     <applet code="Welt.class" width="400" height="400">
     </applet>
   </body>
 </html>
```

Kontrollstrukturen

Auswahl
if (*bedingung*) { *anweisungsblock*} else {*anweisungsblock*}
Mehrfachauswahl
switch (*auszuwertende_variable*) {
case 1 : {*anweisungsblock*} break;
case 2: {*anweisungsblock*} break;
default: {*anweisungsblock*} }
Zählschleifen
for (*zählvariable=anfangswert*; *wiederholungsbedingung*; *iterationsanweisung*)
{*anweisungsblock*}
Abweisende Wiederholung (Wiederholung mit Anfangsbedingung)
while (*wiederholungsbedingung*)
{*anweisungsblock*}
Annehmende Wiederholung (Wiederholung mit Endebedingung)
do {*anweisungsblock*}
while (*wiederholungsbedingung*) ; // Semikolon nicht vergessen!

Felder

int [ ] *zahlenfeld* =new int [4]; // Feld mit 4 Ganzzahlen vereinbaren
zahlenfeld[0] = *9*; *zahlenfeld*[1] = 8; *zahlenfeld*[2] = *7*; *zahlenfeld*[3] = *6*; // Wertzuweisung
int [ ] *zahlenfeld* = { *9,8,7,6* } ; // Feld erhält Werte aus einer Liste zugewiesen
for (int *k* = 0 ;*k* < *zahlenfeld*.length; *k*++) { ... } // durchläuft alle Elemente des Feldes

Methoden / Funktionen

double *a* = *methoden_name*(*x* , *y*);
public double *methoden_name*(int *x*, double *y*)
{ double z;
z = *x* + *y*;
return *z*; }

Programmstruktur mit Objekten

```
class Hello
{    public void Go()
     { System.out.println("Hallo Hansi!");
     }
     public static void main()
     { Hello irgendeinobjektname = new Hello();
       Hello objekt2 = new Hello ();
       irgendeinobjektname.Go();      objekt2.Go();
     }
}
```

Konstruktoren

```
public class hansi
{          private int x;        // Instanzvariable
           public hansi()        // Konstruktor für Objekte der Klasse hansi, heißen genauso
                                 //  wie die Klasse
           {          x = 0;     // Konstruktoren werden beim Vereinbaren eines Objektes
                                 // aus einer Klasse automatisch abgearbeitet
           }
           public int  Beispielmethode ( int y )
           {          return x + y;
           }
}
```

import-Anweisungen

```
import java.awt.*;                  fügt Klassen für Grafikprogrammierung hinzu
import javax.swing.*;       fügt Klassen für Ein-Ausgabeanweisungen hinzu
import java.math.*;         fügt Klassen für mathematische Funktionen hinzu
import java.util.*;         fügt Klassen für Zeit / Datum / Kalender hinzu
import java.io.*;           fügt Klassen für die Dateiarbeit und E/A hinzu
import java.applet.*;       fügt Klassen für Applet-Arbeit hinzu
```

Schlüsselwörter

abstract	Kennzeichnung von Klassen die später implementiert werden
boolean	Datentyp für Wahrheitswerte
break	Unterbrecher des Kontrollflusses (z.B. in Schleifen)
byte	8-bit breiter Ganzzahldatentyp
case	Einleitung eines konkreten Falls (switch-Anweisung)
catch	Blockanfang beim Abfangen von Ausnahmen
char	Zeichendatentyp
class	Leitet Klassendeklaration ein
continue	Stoppt nur den aktuellen Schleifendurchlauf nicht die Schleife
default	Standardfall in einer Fallunterscheidung
do	Leitet Schleife ein (Bedingungsprüfung am Ende des ersten Durchlaufs)
double	Gleitkommadatentyp
else	Einleitung einer Alternative (if.. else..)
extends	Vererbung - Klasse ist Unterklasse von ...
final	so deklarierte Klassen, Methoden und Variable sind fix
finally	Blockanfang, der im Normal- und Ausnahmefall ausgefürt wird
float	Gleitkommadatentyp
for	Leitet Zählschleife ein
goto	noch ohne Funktion- reserviert für spätere Java Versionen
if	Leitet eine Entscheidung ein
implements	Deklaration der implementierten Schnittstelle
import	Einbindung anderer Klassen
instanceof	Operator, der prüft, ob ein Objekt vom Typ einer Klasse ist
int	32-bit breiter Ganzzahldatentyp
interface	Leitet Schnittstellendeklaration ein
long	64-Bit breiter Ganzzahldatentyp
native	Methodenkopf von in einer anderen Programmiersprache geschriebenen Methode
new	Initialisierungsoperator für Referenzdatentypen
package	Paketzuordnungsanweisung
private	Zugriffsbeschränkung für Eigenschaften, Methoden und Konstruktoren
protected	Zugriffsbeschränkung für Eigenschaften, Methoden und Konstruktoren
public	keine Zugriffsbeschränkung für Klassen, Schnittstellen, Eigenschaften, Methoden und Konstruktoren
return	Sprunganweisung für das Verlassen einer Methode
short	16-bit breiter Ganzzahldatentyp
static	Ordnet eine Eigenschaft bzw. Methode einer Klasse zu (nicht direkt einem Objekt)
super	Zeiger auf Eigenschaften und Methoden der Oberklasse
switch	Leitet Fallunterscheidung ein
synchronized	synchonisiert Anweisungen und Methoden (Nebenläufigkeit)
this	Zeiger auf aktuelle Klasse
throw	Löst eine Ausnahme aus
throws	Gibt an das eine Klasse oder Methode eine Ausnahme auslösen kann
transient	Kennzeichnet Variable, deren Werte beim Speichern des Objekts verloren gehen sollen
try	Leitet einen Anweisungsblock ein, der Ausnahmen auslösen kann
void	Methode hat keinen Rückgabewert
voatile	Markiert Variablen als durch nicht-synchronisierte threads veränderbar
while	Leitet Schleife ein (Bedingungsprüfung am Anfang)

Arbeiten mit Zeichenketten

String *zk* = „XYZ123“	Vereinbart zk als Zeichenkette und weist „XYZ123“ zu
zk.charAt(0)	Ermittelt 1. Zeichen aus zk
zk.setChar(0,’A’)	Ersetzt 1. Buchstabe in zk durch A
zk.delete(0,2)	Löscht die ersten drei Zeichen in zk
zk.replace(0,2,“ABC“)	Ersetzt die ersten drei Zeichen in zk durch „ABC“
zk.insert(0,“ABC“)	Fügt nach dem 1. Zeichen die Zeichen „ABC“ ein
zk.length()	Bestimmt die Länge von zk
zk.toLowerCase().toUpperCase()	Konvertierung von zk von Klein- nach Großschreibung
if (*zk1*.equals(*zk2*)) ...	Vergleich zweier Zeichenketten auf Gleichheit
if ! (*zk1*.equals(*zk2*)) ...	Vergleich zweier Zeichenketten auf Ungleichheit

Kurzreferenz

Programmieren mit JAVA in der Schule
Volkmar Heinig

Verlag Holland +Josenhans

1. Basis-Modul
ISBN 37782-60-7

2. Master Modul
ab I / 2008