**Algorithmus** (Def. Ist mit einem Rezept aus einem Kochbuch zu vergleichen☺ dort steht Bsp. Nimm 500 Gramm Mehl und 1 Ei und mische es zusammen)

**Programm** (Ist eigentlich der Koch der alles kocht und ein Produkt=Ergebnis ausgibt)

Prozedur 🡪 Punkt für Punkt mache erst Schritt 1 dann Schritt 2 usw.

Zu 98% mit

Variablen und

Methoden (sind

in allen

Sprachen gleich)

Objektorientierte Programmierung 🡪 es sprechen verschiedene

Methoden miteinander

**Variablen** werden definiert in einer **Deklaration** und erhalten einen Wert durch eine **Zuweisung**.

Wert

Name

Typ

1. Variablen bestehen aus

Parameter

1. Methoden bestehen aus
2. Syntax = immer am Ende mit **;**(Semikolon) dies zeigt an das die Zeile zu Ende ist.
3. (Klassen) folgt später
4. Kompilieren 🡪 Compiler Es wird durch ein Programm (Compiler) in Maschinensprache

gewandelt. NULL EINS

1. Programmstruktur

**Zu 1 Variable**

Eine Variable ist eine BOX zb. Streichholzschachtel dies braucht einen

Namen (wie heiße ich) , einen Typ (was bin ich) und einen Inhalt (Wert)

**Zu 1.1 Typ**

native Datentypen

String (Zeichenkette)

Char (1 Zeichen)

Int (Ganzzahl von -2.147.483.648 bis +2.147.483.648 =32bit) long = 64bit

float oder double (Kommazahlen)

**var** ist ein Joker (es können beliebige Werte eingetragen werden ohne

vom Compiler geprüft zu werden. Der Entwickler muss sich merken was er damit tut.

eigene Datentypen

Muschel oder Lesezeichen sind Beispiele

**Zu 1.2 Name / Deklaration**

 Der Name ist freiwählbar aber bitte mit **SINN** zur Not mit Kommentaren☺

z.Bsp. rechneMittelwertVonTemparatur 🡨 Camelcase

Der Name sollte kleinGROßßkleinGROßß geschrieben werden.

Int zahl; 🡨 Deklaration

zahl = 5; 🡨 Wertzuweisung

Man kann diese Anweisung aber vereinfachen.

Datentyp

Int Zahl = 5;

Name

Wert

**ÜBUNG**

Int zahl1 = 3; 🡨 als Ganzzahl wird der Zahl1 ein Wert von 3 zugewiesen

Char buchstabe = a; 🡨 dem Datentyp (Char) wird dem Buchstaben(Name) das Zeichen a

zugewiesen

Int zahl2; 🡨 Zahl 2 hat nur eine Zuweisung, dass es eine Ganzzahl ist, aber kein

Wert

System.out.print( ) oder system.out.println( ) 🡨 Anweisung den Wert in den Klammern auszugeben

System.out.print(zahl1) = 3

System.out.print(zahl2) = 0 (gibt 0 aus da es keine Wertzuweisung gibt)

zahl2 = zahl1; = 3 (von rechts nach links also gib Zahl2 den Wert von 1)

zahl1 = 10; = neue Wertzuweisung von 3 auf 10

zahl1 = buchstabe; = Fehler (Zahl1 wurde mit int deklariert, kein char möglich)

System.out.print(zahl1) = 10

System.out.print(zahl2) = 3

Möglichkeiten zur Berechnung von neuen Werten:

zahl1 = (zahl2 + 10)\*5; = 3+10 =13 \*5 = 65 daher zahl1 hat neuen Wert von 65

zahl1 = zahl1\*10; = 65\*10=650 neuer Wert von zahl1 ist 650

kurzgeschrieben auch zahl1 \*= 10; = Bedeutet auch 65\*10 also auch 650

zahl2 +=1; = 1+ 3 = 4 Neuer Wert zahl2 ist 4 weil von rechts nach links

System.out.print(zahl1++) = 650 da die Plus erst von rechts gezählt werden

System.out.print(++zahl1) = 1 + 650 = 651

**Zu 6 Programmstruktur**

Schleifen (while, for)

Verzweigen (if, else)

Schleifen ( while, for) Unterschied:

Der Unterschied der while-Schleife zur for-Schleife besteht darin, dass die while-Schleife keinen Initialisierungs- und Aktualisierungsteil besitzt und du bei deinem Beispiel bei der Whileschleife mehr Code hast.  
Grundsätzlich nimmt man die For-Schleife hauptsächlich, wenn man weiß wie oft eine Schleife durchlaufen werden soll.  
Und die While-Schleife als Gegenteil zu der For-Schleife ... wenn die Anzahl der Durchläufe unbekannt ist.

while (Bedingung) {

Anweisung1;

Anweisung2;

}

Bedingung Vergleichsoperatoren

|  |  |
| --- | --- |
| Operator | Funktion |
| == | Testet zwei Variablen auf Gleichheit. |
| != | Testet zwei Variablen auf Ungleichheit. |
| < | Prüft, ob der linke Operand kleiner ist. |
| > | Prüft, ob der linke Operand größer ist. |
| <= | Prüft, ob der rechte Operand größer oder gleich ist. |
| >= | Prüft, ob der rechte Operand kleiner oder gleich ist. |
|  |  |
|  |  |

**While- schleife**

Ein Beispiel:

public class FakultaetWhile {

public static void main (String[] args) {

int inkrement = 4;

int ende = 1001;

int i = 1;

int summe = 0;

**while (i<=ende)**

{

summe += i;

System.out.println(i);

i += inkrement;

}

System.out.println(summe);

}

}

Bedingungen 🡨 werden werden immer in Klammern geschrieben.

**for- schleife**

for (Zählvariable; Bedingung; Erhöhung der Zählvariablen)

{ Code im Rumpf }

|  |  |
| --- | --- |
| Was macht nun dieses Stück Code? In der for-Schleife deklarieren wir die Variable in den runden Klammern und weisen ihr den Wert 0 zu. Nach dem Semikolon folgt dann die Bedingung, die die gleiche ist wie bei der while-Schleife. Ein Semikolon weiter schreiben wir das rein, was mit unserer Zählvariablen i gemacht werden soll – sie wird um eins erhöht.  Der Code im Rumpf der Schleife wird so lange wiederholt, bis i den Wert von drei übersteigt. |  |
|  |  |

public class FakultaetFor{

public static void main (String[] args) {

int inkrement = 4;

int ende = 1001;

int summe = 0;

for (int i=1; i<=ende; i += inkrement;)

{

System.out.println(i);

summe += i;

}

System.out.println(summe);

}

}

**foreach- schleife**

Foreach ( für jedes Element aus der Liste)

{ Code im Rumpf }

|  |  |
| --- | --- |
|  |  |
|  |  |

public class FakultaetFor{

public static void main (String[] args) {

foreach (String name: Liste)

{

System.out.println(name);

}

}

Bedingtes Und vom Typ (boolean)

Nur wenn beide Ausdrücke „true“ haben wird der gesamt Ausdruck „true“. Sollte einer oder beide Ausdrücke den Wert „false“ haben, dann nimmt auch der Gesamtausdruck den Wert „false“ an.

}

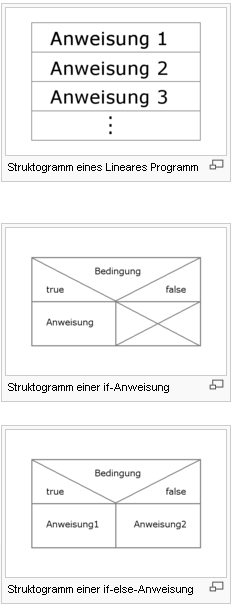
|  |  |  |
| --- | --- | --- |
| & (AND) | true | false |
| true  Operator   |  |  | | --- | --- | |  |  | |  |  | | true | false |
| true  1. Operand   |  |  | | --- | --- | |  |  | |  |  | | true | false |

Logische Und vom Typ (boolean)

Sollte einer oder beide Ausdrücke den Wert „true“ haben, dann nimmt auch der Gesamtausdruck den Wert „true“ an.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  (OR)  2. Operand   |  |  | | --- | --- | |  |  | |  |  | | true | false |
| true | true | true |
| true | true | false |

|  |  |  |
| --- | --- | --- |
| ^ (ex oder) | true | false  Exklusiv Oder vom Typ (boolean)  Nimmt nur dann den Wert „true“ an, wenn der andere Operand, nicht aber beide, den Wert „true“ haben. |
| true | false | true |
| true | true | false |

Verzwiegen (if, else)

Die Bedingte Anweisung und Verzweigung, im folgenden If-Anweisung genannt, ermöglicht es Anweisungen unter Voraussetzung einer bestimmten Bedingung ausführen zu lassen. Dazu wird das Schlüsselwort „if“ verwendet. Dahinter folgt ein Paar von runden Klammern, in dem die Bedingung definiert werden kann. Die Bedingung ist ein Ausdruck der als Ergebnis einen booleschen Wert (true oder false) liefern muss.

Das Schlüsselwort „else“ kann verwendet werden um eine Anweisung im Falle des Nichtzutreffens der Bedingung auszuführen. Es ist optional und in diesem Fall spricht man von einer Verzweigung, da entweder die erste oder die zweite Anweisung ausgeführt wird. Alternativ kann anstatt „else“ ein „else if“ verwendet werden, das wie ersteres „if“ eine weitere Bedingung erwartet. Dadurch ist möglich Kaskaden von Verzweigungen aufzubauen. Ohne Strukturierung durch Blöcke (geschweifte Klammern)

**Allgemeine Syntax**

if (<Bedingung>)

<Anweisung>;

else

<Anweisung>;

Beispiel für eine einfache If-Anweisung mit einem Block von zwei Ausgaben

if (i < 0) {

System.out.print("Die Zahl ist negativ"); // Bedingte Ausgabe

System.out.println(); // Zeilenumbruch auf der Konsole

}

**Komplexeres Beispiel**

if (i < 0) {

System.out.println("Die Zahl ist negativ");}

else if (i == 0) {

System.out.println("Die Zahl ist Null");}

else if (i < 5) {

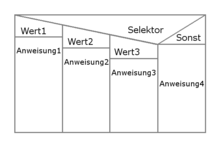
System.out.println("Die Zahl ist kleiner als 5");}

else {

System.out.print("Die Zahl ist größer oder gleich 5");

**Switch-Anweisung**

Struktogramm einer Switch-Anweisung

[](http://commons.wikimedia.org/wiki/File:Mehrseitige_Auswahl.png)Eine Fallunterscheidung, wobei der Ausdruck nur ein logischer Wert, ein Zeichen oder eine ganze Zahl sein kann.

switch (''Ausdruck'') {

case ''Konstante1'': ''Anweisung1''; break;

case ''Konstante2'': ''Anweisung2''; break;

default: ''Anweisung3'';

}

noch ein Beispiel:

switch (anfang){

case `x`: ausgabe = „x“;

case `y`: ausgabe = „y“;

case `z`: ausgabe = „z“;

default.ausgabe = „Meh!“;

sout(ausgabe);

**Methoden**

Zugriff

Typ

Name

Parameter in ()

**Signatur**

Die Signatur einer Methode (also auch des Konstruktors) ist der Name der Methode und die Parameter (welche Datentypen in welcher Reihenfolge). Anhand der Signatur weiß der Compiler, welche Methode er aufrufen muss, wenn irgendwo eine Methode aufgerufen wird, deshalb muss die Signatur eindeutig sein.

Public void summiere ()

Input

|  |  |
| --- | --- |
|  |  |
|  |  |

Gibt etwas aus

|  |  |
| --- | --- |
|  |  |
|  |  |

Zugriff R-Typ Name (Parameter) {

Verb Substantiv

Signatur

(int a, int b)

Bsp.:

Public int summiere ( ) {

System.out.println(“zahl1:”+a);

System.out.println(“zahl1:”+a);

Int summe = a+b;

Return summe;

}

**Array**

String [ **5** ] namensliste; [ ]

Louise

Matthias

Jörg

Ulrike

Thomas

**ACHTUNG: Alle Elemente sind nun von Typ String!!!!!!!!!!**

For ( i = 0; I < n-1; i++); Erklärung: i ist 0 da der Index der Array bei 0 anfängt.

n (Anzahl der Elemente in der Liste)

n -1 weil es ja 5 Elemente gibt aber der Index von 0-4 geht.

temp

Jörg

String [ **5** ] namensliste; [ ]

Louise

Matthias

Ulrike

Thomas

Aus Jörg wird Louise und Louise wird Jörg wobei Jörg kurz in einer Temp gespeichert wird.

Auslesen eines Wertes. Bsp. Element 1:

String einName = namensliste[1];

Ausgabe wäre Jörg oder Louise im 2 Beispiel.

Man kann dem Element 1 auch ein neuen Wert zuweisen:

Namensliste[2] = „Alex“; macht aus Ulrike nun den Alex.

Es gibt auch noch das ganze mit int erstellen.

Int [3] altersliste;

**ACHTUNG: Alle Elemente sind nun von Typ intger!!!!!!!!!!**

Übung

hier die neue Aufgabe (Text stammt aus meinen alten Unterlagen ;) ) Es geht um ein Sortierverfahren namens SelectionSort, das benutzt wird, um Elemente in eine bestimmte Reihenfolge zu bringen.

SelectionSort

Wenn ich Dich bitte, die Zahlen [3,9,5,2,3] der Größe nach zu sortieren, wirst Du vermutlich so vorgehen:

Du suchst das kleinste Element der Liste (2) und nennst es mir. Dann streichst Du in Gedanken diesen Wert,

erhältst die restliche Liste [3,9,5,3] und machst dasselbe wieder. Du nennst also 3 (Restliste [9,5,3]), dann

nochmals 3 (Restliste [9,5]), dann 5 (Restliste [9]) und schließlich 9, womit die Liste endlich leer geworden

ist.

Deine Strategie war also

1.) finde das kleinste Element in der Liste.

2.) nenne es

3.) entferne es aus der Liste

4.) wenn die Liste noch nicht leer ist, gehe zu Schritt 1) zurück.

Wie können wir diesen Algorithmus dem Computer mitteilen?

Der Code:

public class sortieren{

public static void main(String[] args){

int z[] ={3,9,5,2,3};

System.out.println("z sieht vor dem sortieren so aus : " + zToString(z));

zahlenSort(z);

System.out.println("z sieht nach dem sortieren so aus : " + zToString(z));}

public static void zahlenSort(int[] z){

int n = z.length;

for(int i = 0; i< n; i++){

int min = z[i];

int minIndex = i;

for(int j = i+1; j< z.length; j++){

if (min > z[j]){

min = z[j];

minIndex = j;

}}

z[minIndex] = z[i];

z[i] = min;

}}

public static String zToString(int[] z){

String s="";

for(int i=0;i<z.length;i++){

s=s+z[i]+","; }

s=s.substring(0,s.length()-1);

s=s+"\n";

return s; }}