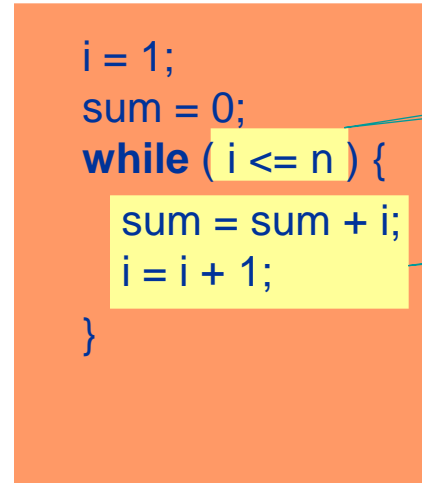
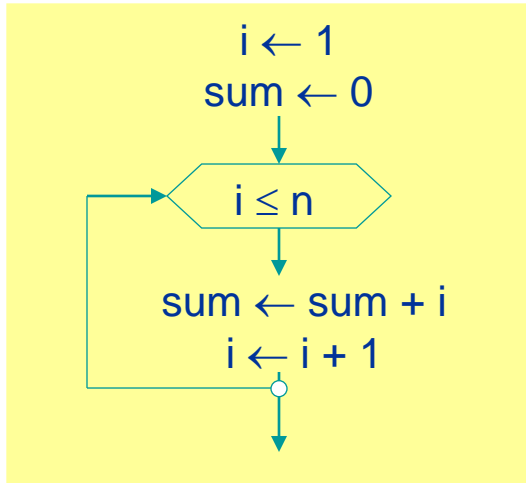


# Schleifen



# While-Schleife

Führt eine Anweisungsfolge aus, solange eine bestimmte Bedingung gilt



Schleifenbedingung

g

Schleifenrumpf

## Syntax

Statement = Assignment | IfStatement | SwitchStatement | WhileStatement | ... | Block.

WhileStatement = **while** "(" Expression ")" Statement .

Wenn Schleifenrumpf aus mehreren Anweisungen besteht, muß er mit {...} geklammert werden.



## Aufgabe: Zahlenfolge lesen und Histogramm ausgeben

Eingabe: 3 2 5

Ausgabe: \*\*\*  
\*\*  
\*\*\*\*\*

```
class Histogram {  
  
    public static void main (String[] arg) {  
        int i = In.readInt();  
        while (In.done()) {  
            int j = 1;  
            while (j <= i ) {Out.print("*"); j++;}  
            Out.println();  
            i = In.readInt();  
        }  
    }  
}
```

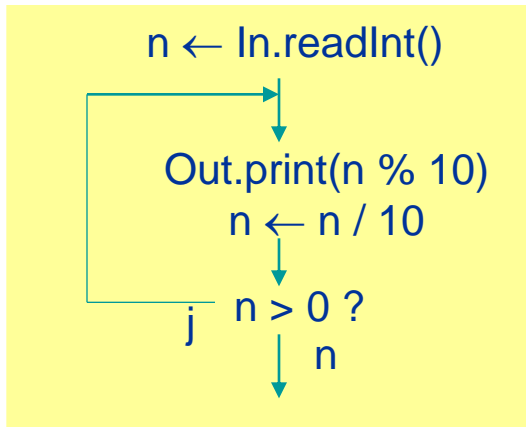
liest die Zahlenfolge

gibt i Sterne aus

# Do-While-Schleife

Abbruchbedingung wird am Ende der Schleife geprüft

Beispiel: gibt die Ziffern einer Zahl in umgekehrter Reihenfolge aus



```
int n = In.readInt();
do {
    Out.print(n % 10);
    n = n / 10;
} while ( n > 0 );
```

Schreibtischttest

n	n % 10
<del>123</del>	3
<del>12</del>	2
<del>1</del>	1
0	

## Syntax

Statement = Assignment | IfStatement | WhileStatement | DoWhileStatement | ... | Block.

DoWhileStatement = "**do**" Statement "**while**" "(" Expression ")" ";".

Wenn Schleifenrumpf aus mehreren Anweisungen besteht, muß er mit {...} geklammert werden.



Warum kann man dieses Beispiel nicht mit einer While-Schleife lösen?

```
int n = In.readInt();  
while (n > 0) {  
    Out.print(n % 10);  
    n = n / 10;  
}
```

"Abweisschleife"

Weil das für  $n == 0$  die falsche Ausgabe liefern würde.  
Die Schleife muß mindestens einmal durchlaufen werden, daher :

```
int n = In.readInt();  
do {  
    Out.print(n % 10);  
    n = n / 10;  
} while (n > 0);
```

"Durchlaufschleife"



Falls die Anzahl der Schleifendurchläufe im voraus bekannt ist

```
sum = 0;  
for ( i = 1 ; i <= n ; i++ )  
    sum = sum + i;
```

- 1) Initialisierung der Laufvariablen
- 2) Abbruchbedingung
- 3) Ändern der Laufvariablen

Kurzform für

```
sum = 0;  
i = 1;  
while ( i <= n ) {  
    sum = sum + i;  
    i++;  
}
```

# Syntax der For-Schleife

```
ForStatement = "for" "(" [ForInit] ";" [Expression] ";" [ForUpdate] ")" Statement.  
ForInit      = Assignment {"," Assignment}  
             | Type VarDecl {"," VarDecl}.  
ForUpdate    = Assignment {"," Assignment}.
```

## Beispiele

```
for (i = 0; i < n; i++) ...
```

```
for (i = 10; i > 0; i--) ...
```

```
for (int i = 0; i <= n; i = i + 1) ...
```

```
for (int i = 0, j = 0; i < n && j < m; i = i + 1, j = j + 2) ...
```

```
for (;;) ...
```

```
for (int n = In.readInt(); i < 1000; n = In.readInt()) {...
```



# Beispiel: Multiplikationstabelle drucken

```
class PrintMulTab {  
  
    public static void main (String[] arg) {  
        int n = In.readInt();  
        for (int i = 1; i <= n; i++) {  
            for (int j = 1; j <= n; j++) {  
                Out.print(" " + i * j);  
            }  
            Out.println();  
        }  
    }  
}
```

*Schreibtischtest für n == 3*

i	j			
1	1	1	2	3
	2	2	4	6
	3	3	6	9
	4			
2	1			
	2			
	3			
	4			
3	1			
	2			
	3			
	4			
4				



## Beispiel: Summieren mit Fehlerabbruch

```
int sum = 0;
int x = In.readInt();
while (In.done()) {
    sum = sum + x;
    if (sum > 1000) {Out.println("zu gross"); break;}
    x = In.readInt();
}
```

break verläßt die Schleife, in der es enthalten ist (while, do, for)

Schleifenabbruch mit `break` möglichst vermeiden: schwer zu verifizieren. Meist läßt sich dasselbe mit *while* ebenfalls ausdrücken:

```
int sum = 0;
int x = In.readInt();
while (In.done() && sum <= 1000) {
    sum = sum + x;
    if (sum <= 1000) x = In.readInt();
}
// ! In.done() || sum > 1000
```



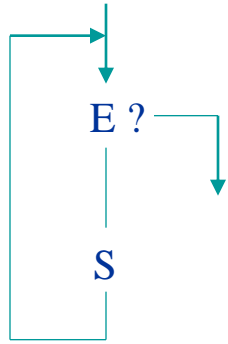
## Beispiel

```
outer:                // Marke!  
for (;;) {          // Endlosschleife!  
    for (;;) {  
        ...  
        if (...) break; // verläßt innere Schleife  
        else break outer; // verläßt äußere Schleife  
        ...  
    }  
}
```

## Wann ist ein Schleifenabbruch mit `break` vertretbar?

- bei Abbruch wegen Fehlern
- bei mehreren Ausprägungen an verschiedenen Stellen der Schleife
- bei echten Endlosschleifen (z.B. in Echtzeitsystemen)

# Vergleich der Schleifenarten



**Abweisschleife**

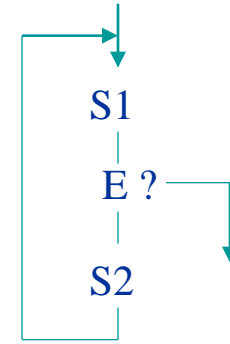
```
while (E)  
  S
```

```
for (I; E; U)  
  S
```



**Durchlaufschleife**

```
do  
  S  
while (E)
```



**allgemeine Schleife**

```
for (;;) {  
  S1;  
  if (E) break;  
  S2;  
}
```



## Wie denken Programmierexperten?

- nicht in einzelnen Anweisungen
- sondern in größeren Programm-Mustern

Muster = Schema zur Lösung häufiger Aufgaben

## Experten

- benutzen Muster intuitiv
  - z.B. Stellungen im Schachspiel
  - z.B. Redewendungen in Geschäftsbriefen
- lösen Aufgaben in Analogie zu bekannten Aufgaben

Frage: Was sind typische Muster in der Programmierung?



# Zusammensetzen von Programmen aus Mustern

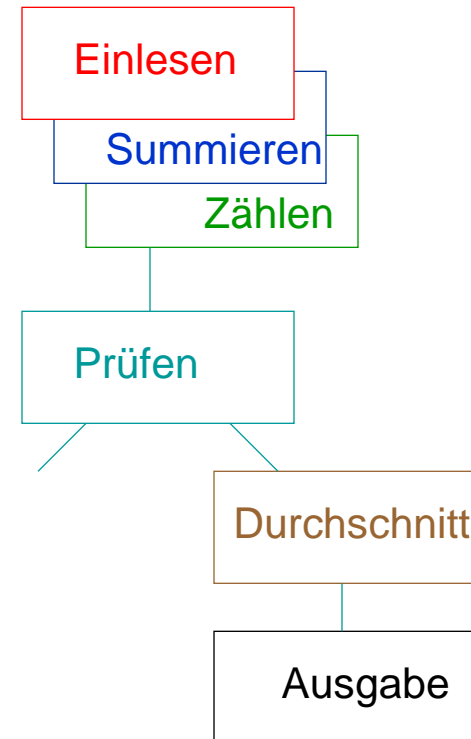
Beispiel: Berechnung des Durchschnitts einer Zahlenfolge

Einlese-Muster	Zähl-Muster
<pre>int x = In.readInt(); while (In.done()) {     ...     x = In.readInt(); }</pre>	<pre>int n = 0; while (...) {     n++;     ... }</pre>
Summierungs-Muster	Prüf-Muster
<pre>int sum = 0; while (...) {     sum = sum + x;     ... }</pre>	<pre>if (n != 0)     ... else     Out.println("error");</pre>
Durchschnitts-Berechn.	Ausgabe
<pre>float avg = (float)sum / n;</pre>	<pre>Out.println("avg = " + avg);</pre>



# Zusammensetzen der Muster

```
int x = In.readInt();
int sum = 0, n = 0;
while (In.done()) {
    sum = sum + x;
    n++;
    x = In.readInt();
}
if (n != 0) {
    float avg = (float)sum / n;
    Out.println("avg = " + avg);
} else
    Out.println("error");
```



## Und-Verknüpfung von true- und false-Termen

true true true  $\Rightarrow$  true

true false true  $\Rightarrow$  false

Einlese-Muster	Und-Muster
<pre>boolean b = In.readBoolean(); while (In.done()) {     ...     b = In.readBoolean(); }</pre>	<pre>boolean ok = true; while (...) {     ...     ok = ok &amp;&amp; b; }</pre>
Ausgabe-Muster	
<pre>Out.println(ok);</pre>	

```
boolean b = In.readBoolean();
boolean ok = true;
while (In.done()) {
    ok = ok && b;
    b = In.readBoolean();
}
Out.println(ok);
```

# Schrittweise Eingabe von Ablaufstrukturen

## Bei Schleifen

```
while (In.done()) {  
}
```

```
while (In.done()) {  
    x = In.readInt();  
}
```

```
while (In.done()) {  
    ...  
    x = In.readInt();  
}
```

## Bei Abfragen

```
if (n != 0) {  
} else {  
}
```

```
if (n != 0) {  
    avg = (float)sum / n;  
    Out.println("avg = " + avg);  
} else {  
}
```

```
if (n != 0) {  
    avg = (float)sum / n;  
    Out.println("avg = " + avg);  
} else {  
    Out.println("error");  
}
```



# Entwurf, Implementierung und Testen von Programmen



- Problem erfassen und beschreiben (Aufgabenstellung)
- Lösungsidee erarbeiten und niederschreiben
- Lösungsidee in einen schrittweisen Ablauf überführen (= Algorithmus)
- Algorithmus so weit verfeinern und konkretisieren, dass Umsetzung in Java direkt möglich ist
- Programmierung in Java
- Überlegen von Testfällen (dabei alle möglichen Sonderfälle, Grenzfälle betrachten)
- Testen und Verbessern
- Dokumentieren der Ergebnisse



## ■ Aufgabenstellung

Entwickeln Sie einen Algorithmus, der folgendes leistet: Es sollen beliebig viele nicht-negative Zahlen von einem Eingabemedium gelesen werden. Die Eingabe ist beendet, wenn eine abschließende negative Zahl gelesen wird. Von den eingelesenen Zahlen sollen die zwei größten Werte berechnet und ausgegeben werden. Wenn keine oder nur eine Zahl eingegeben wurde, soll eine entsprechende Fehlermeldung ausgegeben werden.

Beispiele:

Eingabe: 1.5 2 0 4.25 1 -1

Ausgabe: Groesste Zahlen: 4.25 und 2

Eingabe: 5.25 -3.2

Ausgabe: Zuwenig Zahlen eingegeben!

## ■ Lösungsidee

Es werden nacheinander Werte eingelesen und die Anzahl der Zahlen mitgezählt. Ist eine Zahl negativ, so wird das Einlesen beendet. Ist die Anzahl der eingelesenen Werte (d.h. der nicht-negativen Werte) jedoch kleiner als 2, so wird eine Fehlermeldung ausgegeben, ansonsten werden die beiden bisher größten Werte ausgegeben.

Einlesen der Werte: Immer wenn ein gültiger Wert eingelesen wurde, wird er mit dem aktuellen größten und zweitgrößten Wert verglichen. Stellt man fest, dass man einen neuen größten oder zweitgrößten Wert, so werden größter und zweitgrößter Wert nachgeführt.

Da alle Zahlen größer gleich 0 sind, können die beiden größten Werte auf 0 initialisiert werden (minimale Maxima). Kommen später größere Zahlen vor, so sind sie auf jeden Fall größer gleich den Startwerten.

Beispiele:

größter: 20; zweitgrößter: 10; neuer: 15 => größter: 20; zweitgrößter: 15

größter: 20; zweitgrößter: 15; neuer: 10 => größter: 20; zweitgrößter: 15

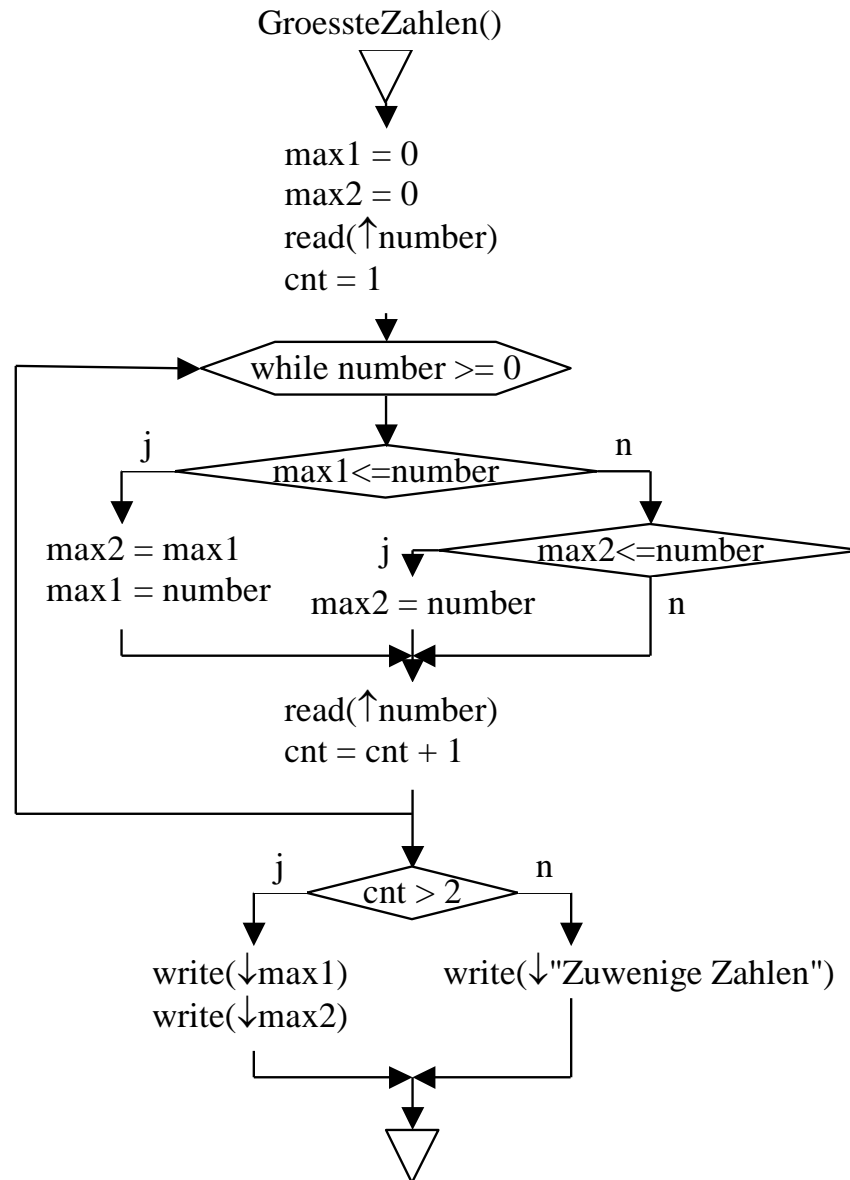
größter: 20; zweitgrößter: 10; neuer: 30 => größter: 30; zweitgrößter: 20

- Interaktiv entwickeln



# Beispiel "Die 2 größten Zahlen" (3)

- Ablaufdiagramm



# Beispiel "Die 2 größten Zahlen" (4)

- Programm: interaktiv entwickeln



- Was will man testen
  - Normale Fälle
    - testen normale Funktion des Algorithmus
    - dabei an alle unterschiedlichen Möglichkeiten denken
  - Grenzfälle
    - sind jene Fälle bei dem der Algorithmus noch funktionieren soll
    - Grenzfälle sind sehr wichtige Testfälle
  - Sonderfälle
    - sind jene Fälle, für die der Algorithmus eigentlich nicht funktioniert, die aber vorkommen können
    - benötigen spezielle Behandlung
- Wie will man testen
  - Eingabewerte
  - Testpositionen
  - erwartete Ergebnisse



## ■ Testfälle

### **Testfälle/Grenzen / Sonderfälle**

Keine Zahl eingelesen => Fehlerfall, zuwenige Werte (minimale Eingabe)

Nur eine Zahl eingelesen => Fehlerfall, (gerade noch) zuwenige Werte

Zwei Zahlen eingelesen (minimaler Wert) => Grenzfall gerade genügend Werte

Eingabe einer Buchstabenfolge => Illegale Eingabe

Eingabe von ganzen und Fließkomma-Zahlen => Mögliche legale Eingaben

Eingabe einer Nullfolge => Minimale Maxima

Größter und zweitgrößter Wert gleich => Mehrfach vorkommendes Maximum

Größter und zweitgrößter Wert ungleich => Einfaches Maximum

Grosse Werte => Oberer Wertebereich

Viele Werte => Obere Werteanzahl

...

# Beispiel "Die 2 größten Zahlen" (6)

## ■ Testplan

Nr	Eingabewerte	Erwartete Ausgabe	Zweck
1	-1	Zuwenige Zahlen	Fehlerfall, zuwenige Werte
2	1 -2.5	Zuwenige Zahlen	Fehlerfall, zuwenige Werte (gerade noch)
3	5 7 -1	7 5	Grenzfall, gerade genügend Werte
4	Das ist ein Test	<undefiniert>	Illegale Eingabe
5	3 4.5 12.2 10 -2	12.2 10	Mögliche legale Eingaben
6	0 0 0 0 0 -1	0 0	Minimale Maxima
7	3 7.5 3.2 0 7.5 2 1.2 7 -1	7.5 7.5	Mehrfach vorkommendes Maximum
8	7 4 3 10 -1	10 7	Einfaches Maximum
9	1E100 1.53E123 13 -1	1.53E123 1E100	Oberer Wertebereich (?)
10	5 3 6 3 2 6 3 5 5.5 2 10 9 12 4 1.5 0.23 7.9 3.3 2.1 4 2 11 23 1.2 -1	23 12	Obere Werteanzahl (?)
11	0 2 3 4 23 123 -1	123 23	Aufsteigende Werte
12	123 23 4 3 2 0 -1	123 23	Absteigende Werte
13	4 23 2 2 123 2 0 3 -1	123 23	Allgemeiner Fall
14	...	...	...

# Beispiel "Die 2 größten Zahlen" (7)

- Testen interaktiv entwickeln

