

Schleifen

Eine häufige Situation beim Programmieren ist, dass bestimmte Anweisungen mehrfach ausgeführt werden sollen. Hier kommen Schleifen zum Einsatz. Im Prinzip würde eine einzige Schleifenvariante ausreichen, um alle Anwendungsfälle abzudecken. Die Programme wären aber dabei je nach Situation etwas unübersichtlich. Daher gibt es drei verschiedene Schleifenvarianten, die sie für jeweils bestimmte Situationen prädestinieren.

for-Schleife

Die for-Schleife ist dann optimal, wenn bereits vorher feststeht, wie oft die Schleife durchlaufen werden soll.

Die grundsätzliche Struktur einer for-Schleife sieht wie folgt aus:

```
for ( ; ; )  
{  
  
}
```

In den runden Klammern müssen noch alle Angaben ergänzt werden, die den Ablauf der for-Schleife steuern. In den durch die geschweiften Klammern gebildeten Block kommen alle Anweisungen, die durch die Schleife mehrfach ausgeführt werden sollen.

Eine vollständige for-Schleife sieht z.B. wie folgt aus:

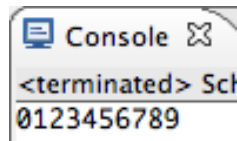
```
int i;  
  
for (i= 0 ; i < 10; i++)  
{  
    System.out.print(i);  
}
```

Die Steueranweisungen für die for-Schleife sind nun vollständig eingetragen:

1. Die erste Angabe gibt an, mit Hilfe welcher Variablen die Anzahl der Schleifendurchläufe mitgezählt werden soll (hier *i*) und bei welchem Wert das Zählen begonnen werden soll (hier 0). (Die zum Mitzählen verwendete Variable muss natürlich weiter oben deklariert worden sein, sonst gibt es eine Fehlermeldung; daher steht weiter oben die Variablendeklaration für *i*.)
2. Die zweite Angabe ist eine Aussage; solange diese Aussage wahr ist, wird die Schleife ein weiteres Mal ausgeführt. In diesem Beispiel wird die Schleife weiter ausgeführt, solange $i < 10$ ist.
3. Die dritte Angabe gibt an, wie die Zählvariable *i* bei jedem Schleifendurchlauf erhöht werden soll; $i++$ ist eine Kurzschreibweise die zu *i* den Wert 1 addiert.

Der Inhalt der Schleife ist eine Ausgabeanweisung, die den Wert der Variablen *i* ausgibt – so können wir beobachten, wie dieser Wert von Durchlauf zu Durchlauf wächst.

Die Ausgabe sieht wie folgt aus:

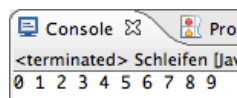


Das sieht nicht besonders schön aus, weil zwischen den Zahlen kein Leerraum ist. Am schnellsten schaffen wir hier Abhilfe über eine weitere Print-Anweisung:

```
public static void main(String[] args)
{
    // TODO Auto-generated method stub
    int i;

    for (i= 0 ; i < 10; i++)
    {
        System.out.print(i);
        System.out.print(" ");
    }
}
```

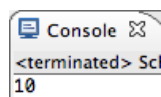
Wir geben damit einfach nach jeder Ausgabe des Wertes der Variablen *i* ein Leerzeichen aus und erhalten dann als Ausgabe:



Ein bei der for-Schleife gerne gemachter Fehler ist ein Strichpunkt zu viel:

```
for (i= 0 ; i < 10; i++);|
{
    System.out.print(i);
    System.out.print(" ");
}
```

Das Ergebnis ist dann:



Eine kleine Änderung hat eine große Wirkung. Das Programm ist immer noch syntaktisch korrekt, d.h. die Regeln der Sprache Java sind weiterhin eingehalten. Die Semantik hat sich aber komplett geändert.

Der Strichpunkt am Ende der for-Zeile bedeutet, dass an genau dieser Stelle die for-Schleife zu Ende ist. Sie läuft also weiterhin wie angegeben ab, hat aber keinen Inhalt mehr. Der durch die geschweiften Klammern gebildete Block wird erst ausgeführt, nachdem die for-Schleife einmal komplett abgearbeitet wurde. Dabei wird *i* solange erhöht, bis die Bedingung $i < 10$ nicht mehr erfüllt ist. Das ist das erste Mal der Fall, wenn die Variable *i* den Wert 10 angenommen hat. Dann wird die for-Schleife verlassen und der Wert der Variablen *i* ausgegeben.

Die for-Schleife wird u.a. häufig dafür benötigt, mit allen Elementen eines Array etwas anzustellen. Der folgende Programmcode gibt z.B. alle Elemente eines zehn Elemente großen Arrays zahlen aus:

```
for (i= 0; i < 10; i++)
{
    System.out.print(zahlen[i]);
    System.out.print(" ");
}
```

while-Schleife

Die while-Schleife wird verwendet, wenn die Anzahl der erforderlichen Durchläufe nicht schon im voraus feststeht, sondern von einer zu erfüllenden Bedingung abhängt. Es werden dann soviele Durchläufe gemacht wie die Bedingung vorgibt, wobei man natürlich darauf achten muss, keine Endlosschleife zu programmieren.

Die Grundstruktur einer while-Schleife ist:

```
while ()
{
}
```

In den runden Klammern muss noch eine Aussage ergänzt werden; solange diese Aussage wahr ist, wird die while-Schleife weiterhin ausgeführt. In den durch die geschweiften Klammern gebildeten Block kommen die Anweisungen, die durch die Schleife mehrfach ausgeführt werden sollen. Ein vollständiges Beispiel sieht wie folgt aus:

```
21     i= 0;
22     while (i*i*i < 100)
23     {
24         System.out.print(i);
25         System.out.print(" ");
26         i++;
27     }
```

In Zeile 22 wurde jetzt eine Bedingung ergänzt: Solange $i*i*i < 100$ ist wird die Schleife weiterhin ausgeführt. Die geschweiften Klammern in den Zeilen 23 und 27 bilden einen Block. Die darin enthaltenen Zeilen werden durch die Schleife mehrfach ausgeführt, z.B. die beiden print-Anweisungen in den Zeilen 24 und 25. Um die Steuerung der Schleife muss man sich selbst kümmern – dazu gehört insbesondere dafür zu sorgen, dass die Bedingung irgendwann falsch wird, sonst hat man eine Endlosschleife. In diesem Beispiel geschieht dies dadurch, dass die Variable i in der Zeile 21 den Wert 0 erhält und in der Zeile 26 der Wert von i jedesmal um 1 erhöht wird; dadurch ist die Bedingung in Zeile 22 irgendwann einmal falsch und die Schleife endet.

Auch hier gibt es eine „nette“ Fehlermöglichkeit:

```
i= 0;
while (i*i*i < 100);
{
    System.out.print(i);
    System.out.print(" ");
    i++;
}
```

Ein Strichpunkt zu viel erzeugt eine Endlosschleife, weil die Bedingung immer wahr bleibt.

do-Schleife

Für die do-Schleife gilt wie für die while-Schleife dass sie dann geschickt ist, wenn die Anzahl der Durchläufe nicht bereits im voraus feststeht.

Das grundsätzliche Aussehen einer do-Schleife ist wie folgt:

```
do
{
    } while ();
```

In die runden Klammern kommt eine Aussage; solange diese Aussage wahr ist, wird die Schleife weiterhin ausgeführt. Die Anweisungen, die durch die Schleife mehrfach ausgeführt werden sollen kommen zwischen die geschweiften Klammern.

Ein vollständiges Beispiel kann wie folgt aussehen:

```
31     i= 0;
32     do
33     {
34         System.out.print(i);
35         System.out.print(" ");
36         i++;
37     } while (Math.sqrt(i) < 20);
```

Die Schleife wird ausgeführt, solange die Wurzel von i (sqrt steht für square root) kleiner als 20 ist – das ist die Bedingung in Zeile 37. Zur Steuerung der Schleife wird in Zeile 31 die Variable i mit dem Wert 0 bestückt und innerhalb der Schleife in Zeile 36 ihr Wert jedesmal um 1 erhöht – das sorgt dafür, dass die Aussage in Zeile 37 irgendwann nicht mehr wahr ist und die Schleife abbricht – sonst hätte man eine Endlosschleife.

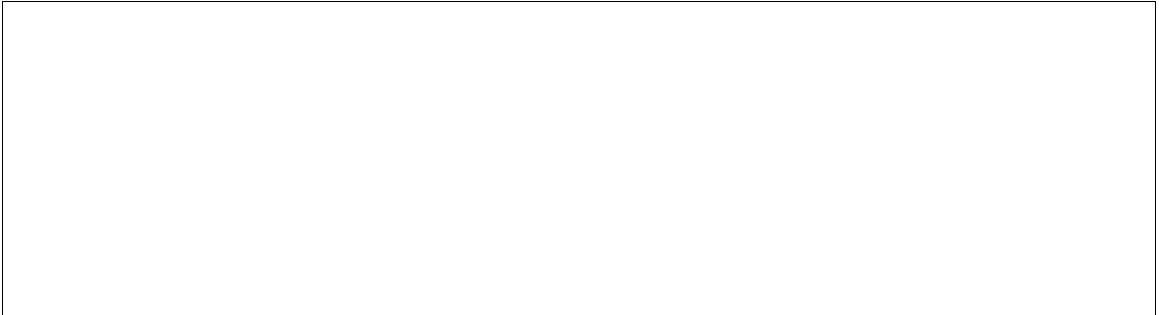
Das Programm gibt natürliche Zahlen bis 399 aus, da die Wurzel von 400 ja 20 ist und die Bedingung damit falsch.

Man mag sich nun fragen, warum es sowohl die while-Schleife als auch die do-Schleife gibt – wo doch beide dann eingesetzt werden, wenn die Anzahl der Schleifendurchläufe nicht bereits im voraus feststeht.

Der Unterschied liegt in der Position der Bedingung. Programmierere dazu das obige Beispiel für die do-Schleife mit der Wurzel in der Abbruchbedingung neu und verwende dazu eine while-Schleife.

Schleifen

Halte hier fest, was die unterschiedlichen Position der Schleifenbedingung bei der while- und der do-Schleife für eine Auswirkung haben:



Zur Entscheidung für einen bestimmten Schleifentyp mag das folgende Diagramm helfen:

