Anweisungen (statements)

Sven Eric Panitz

Unterschied zu Ausdrücken

- Anweisungen werten nicht zu einem Ergebnis aus
- Anweisungen steuern oft den Kontrollfluss:
  - In welcher Reihenfolge wir das Programm abgearbeitet
- In Anweisungen finden sich oft viele Ausdrücke als Teil der Anweisung
- Es gibt Programmiersprachen ohne Anweisungen, die nur Ausdrücke kennen.

- Ausdrücke betonen, **was** berechnet werden soll:

```
return
     hour < that.hour
  || (hour==that.hour && minute < that.minute)
  || (hour==that.hour && minute==that.minute
                      && sec<that.sec);
```

- Anweisungen betonen, **wie** gerechnet werden soll:

```
if (hour > that.hour) return false;
if (hour < that.hour) return true;
if (minute > that.minute) return false;
if (minute < that.minute) return true;
if (sec < that.sec) return true;
return false;
```

Anweisungen Überblick

- Aufruf von void-Methoden.
- Zuweisung
- return-Anweisung
- Bedingungen mit if
- Schleifen (mit break, continue)
- Fallunterscheidungen mit switch
- Ausnahmen
  - werfen
  - abfangen

return

- Die Return-Anweisung beendet eine Methode komplett und gibt das Ergebnis zurück.
- Damit kann nach einer return-Anweisung keine weitere Anweisung stehen.
- Der Compiler verbietet dieses.

Return beendet Methode

```
int f(int x){
  return x*x;
  System.out.println(“fertig“);
}
```

```
jshell> int f(int x){
   ...>     return x*x;
   ...>     System.out.println("fertig");
   ...>   }
|  Error:
|  unreachable statement
|      System.out.println("fertig");
|      ^---------------------------^
```

If-Anweisung

```
int f(int x){

  if (boolean-expression){
     statements
  } else {
     statements
  }

}
```

If-Anweisung

```
int f(int x){
  int result = 0;
  if (      x  > 0          ){
    result = 1;
  } else {
    result = -1;
  }
  return result;
}
```

If-Anweisung

```
int f(int x){

  if (boolean-expression){
      statements
  } else {
      statements
  }

}
```

If vs ? :

- Der Bedingungsoperator ? : hat immer einen Wert und benötigt daher zwei Alternativen

```
String y = x > 0 ? “Alt 1“ : “Alt 2“;
```

- Die if-Anweisung ist eine Verzweigung. Der else-Zweig ist optional:

```
String y = ““;
if (x > 0){
  y = “Einzige Alternative“;
}
```

Achtung: Mehrdeutigkeit

Gestern wurde eine Mutter von fünf Kindern ermordet.

```
int x=42;
int result = 0;
if (x<0)
  if (x<=42) result=5;
 else result = 42;
System.out.println(result);
```

Wozu gehört das else?

Achtung: Mehrdeutigkeit

Gestern wurde eine Mutter von fünf Kindern ermordet.

```
int x=42;
int result = 0;
if (x<0){
   if (x<=42){
      result=5;
   }else{
      result = 42;
   }
}
System.out.println(result);
```

Achtung: Mehrdeutigkeit

Gestern wurde eine Mutter von fünf Kindern ermordet.

```
int x=42;
int result = 0;
if (x<0){
  if (x<=42){
    result=5;
  }
}else{
  result = 42;
}
System.out.println(result);
```

Achtung: Unnötige If-Vermeiden

- statt

```
if (bedingung) return true;
 else return false;
```

  direkt schreiben:

```
return bedingung;
```

- statt

```
if(bedingung) return false;
 else return true;
```

  direkt schreiben:

```
return !bedingung;
```

Schleifen

- While-Schleife
- Do-While-Schleife
- For-Schleife
- For-each-Schleife (später)

While-Schleife

```
int f(int x){

  while(boolean-expression){
    statements

  }

}
```

While-Schleife

```
int f(int x){
  int result = 1;
  while(         x>0         ){
    result = result * x;
    x = x – 1;
  }
  return result;
}
```

Anweisungen nachvollziehen

- f(5)
- Tabellarisch schrittweise nachvollziehen
-

x	result	(x < 0)
5	1	true
4	5	true
3	20	true
2	60	true
1	120	true
0	120	false

- => 120

Do-While-Schleife

```
int f(int x){

   do {
      statements

   }while(boolean-expression)

}
```

Do-While-Schleife

```
int f(int x){
  int result = 0;
  do {
    result = result * x;
    x = x - 1;
  }while(     x > 0      )
  return x;
}
```

Do-While-Schleife

```
int f(int x){

   do {
      statements

   }while(boolean-expression)

}
```

While-Schleifen

Bei den zwei Varianten der While-Schleife spricht man auch von:

- vorgeprüfter Schleife
- und nachgeprüfter Schleife

For-Schleife

- Ziemliche häufig haben while-Schleifen die folgende Form:

```
int x = 10;
while (x > 0){
   System.out.println(x);
   x = x-1;
}
```

Initialisierung der Schleifenvariable

Test für Schleifendurchlauf

Weiterschalten der Schleifenvariable

Schleifenrumpf

Die 3 Teile, die steuern, wie oft die Schleife durchlaufen wird, sind auf drei Stellen verteilt:

- vor der Schleife
- in der Schleifenbedingung
- am Ende des Schleifenrumpfs

For-Schleife

- Dafür wurde die for-Schleife erfunden:

```
int x = 10;
while (             x > 0            ){
   System.out.println(x);
   x = x-1;
}
```

Initialisierung der Schleifenvariable

Test für Schleifendurchlauf

Weiterschalten der Schleifenvariable

Schleifenrumpf

For-Schleife

- Dafür wurde die for-Schleife erfunden:

```
int x = 10;
for   (           ;x > 0;           ){
   System.out.println(x);
   x = x-1;
}
```

Initialisierung der Schleifenvariable

Test für Schleifendurchlauf

Weiterschalten der Schleifenvariable

Schleifenrumpf

For-Schleife

- Dafür wurde die for-Schleife erfunden:

```
for ( int x = 10; x > 0;                  ){
   System.out.println(x);
   x = x-1;
}
```

Initialisierung der Schleifenvariable

Test für Schleifendurchlauf

Weiterschalten der Schleifenvariable

Schleifenrumpf

For-Schleife

- Dafür wurde die for-Schleife erfunden:

```
for ( int x = 10; x > 0; x = x -1){
   System.out.println(x);

}
```

Initialisierung der Schleifenvariable

Test für Schleifendurchlauf

Weiterschalten der Schleifenvariable

Schleifenrumpf

For-Schleife

- Dafür wurde die for-Schleife erfunden:

```
for ( int x = 10; x > 0; x = x -1){
   System.out.println(x);

}
```

Initialisierung der Schleifenvariable

Test für Schleifendurchlauf

Weiterschalten der Schleifenvariable

Schleifenrumpf

Jetzt befinden sich alle Informationen, die angeben, wie oft die Schleife durchlaufen wird, im Kopf der Schleife. Im Rumpf steht nur noch, was mehrfach ausgeführt werden soll.

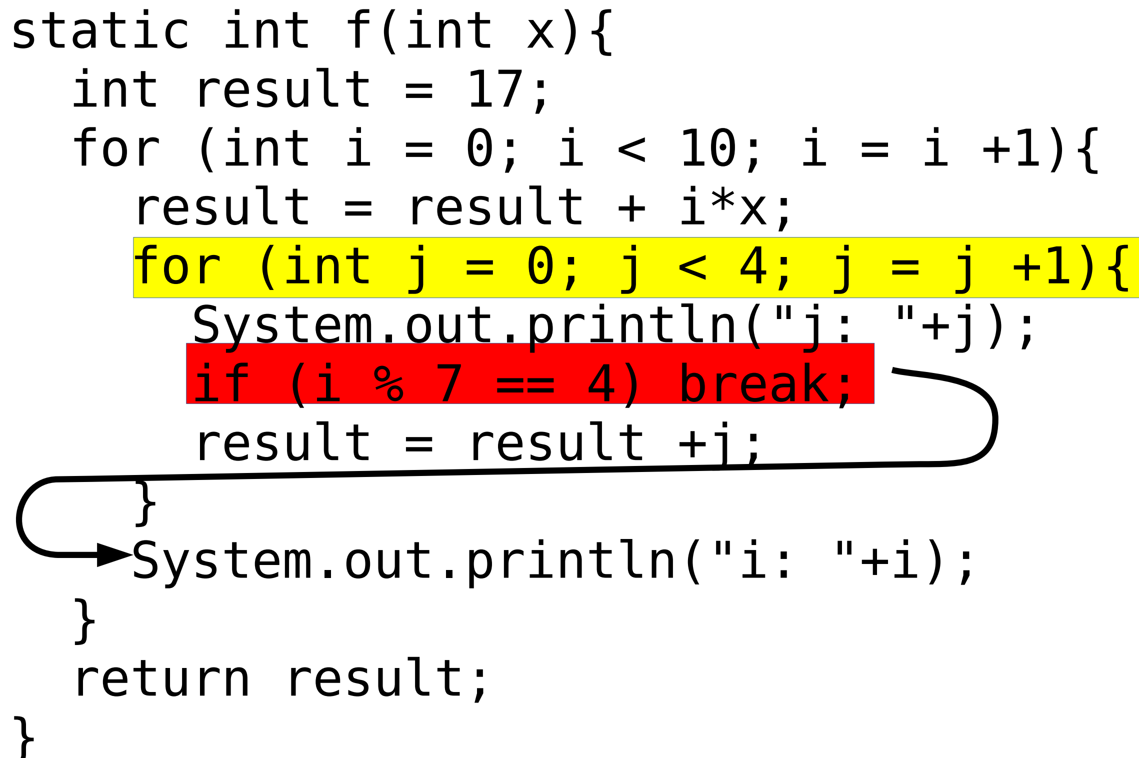
Schleifen verlassen mit `break`

- Mit dem Schlüsselwort `break` lassen sich alle Schleifen auch außerplanmäßig direkt verlassen.

```
int f(int x){
  int result = 17;
  for (int i = 0; i < 10; i = i +1){
    result = result + i*x;
    if (i % 7 == 4) break;
    System.out.println("i: "+i);
  }
  return result;
}
```

Schleifen verlassen mit `break`

- Das break bezieht sich, wenn mehrere Schleifen ineinander stecken auf die innerste Schleife:



```
static int f(int x){
  int result = 17;
  for (int i = 0; i < 10; i = i +1){
    result = result + i*x;
    for (int j = 0; j < 4; j = j +1){
      System.out.println("j: "+j);
      if (i % 7 == 4) break;
      result = result +j;
    }
    System.out.println("i: "+i);
  }
  return result;
}
```

Schleifen verlassen mit `break`

- Schleifen können ein Label bekommen und explizit beim break angegeben werden, welche Schleife verlassen werden soll:

```
static int f(int x){
  int result = 17;
  außen: for (int i = 0; i < 10; i = i +1){
    result = result + i*x;
    for (int j = 0; j < 4; j = j +1){
      System.out.println("j: "+j);
      if (i % 7 == 4) break außen;
      result = result +j;
    }
    System.out.println("i: "+i);
  }
  return result;
}
```

Zum nächsten Schleifendurchgang mit `continue`

- Mit dem Schlüsselwort `continue` lässt sich direkt zum nächsten Schleifedurchgang springen.

```
int f(int x){
  int r = 17;
  for (int i = 0; i < 10; i = i +1){
    r = r + i*x;
    if (i % 7 == 4) continue;
    r = r + 1;
    System.out.println("i: "+i+" r: "+r);
  }
  return r;
}
```

Zum nächsten Schleifendurchgang mit `continue`

- Der Aufruf von `System.out.println(f(5));` führt zu folgender Ausgabe:

```
i: 0 r: 18
i: 1 r: 24
i: 2 r: 35
i: 3 r: 51
i: 5 r: 97
i: 6 r: 128
i: 7 r: 164
i: 8 r: 205
i: 9 r: 251
251
```

Spring in ein case mit switch

Eine Fallunterscheidung auf konstante Werte kann mit der switch-Anweisung gemacht werden.

```
void f(int x){
  switch (4 * x){
    case 42 : System.out.println(42);
    case 52 : System.out.println(52);
    case 32 : System.out.println(32);
    case 22 : System.out.println(22);
    case 12 : System.out.println(12);
    default : System.out.println("0");
  }
}
f(13);
```

f(13) bedeutet:
Springe zum Fall für 4*13 also zum Fall 52.
Arbeite diesen und alle folgenden Fälle ab.
Ausgabe:

52
32
22
12
0

Springe in ein case mit switch

Soll nur der Fall ausgeführt werden, der angesprungen wurde, muss dieser mit einem break enden:

```
void f(int x){
   switch (4 * x){
     case 42 : System.out.println(42);break;
     case 52 : System.out.println(52);break;
     case 32 : System.out.println(32);break;
     case 22 : System.out.println(22);break;
     case 12 : System.out.println(12);break;
     default : System.out.println("0");
  }
}
f(13);
```

f(13) springt jetzt zum Fall 52 und verlässt beim break die Anweisung.

Ausgabe:

52

Switch-case-Anweisung

- Ein case kann nur eine Zahlenkonstante,
- eine Stringkonstante,
- ein Aufzählungswert (kommt später)
- oder ein Klassenpattern (kommt später) sein.
- Es gibt auch einen switch-case-Ausdruck. Dabei steht nach den Fällen statt : ein -> und es muss immer einen default-Fall geben.

Zusammenfassung

- Anweisungen kontrollieren den Kontrollfluss
- oder verändern Werte im Speicher (Zuweisung).
- return verlässt Methoden, egal wo.
- if und switch dienen der Fallunterscheidung
- Schleifen steuern wiederholte Ausführung
- Schleifen können mit break verlassen werden
- mit continue kann zum nächsten Schleifendurchgang gesprungen werden.