

Grundlagen der Programmierung

VL05: Anweisungen und Ablaufsteuerung - Teil II

Prof. Dr. Samuel Kounev,
M.Sc. Norbert Schmitt



Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

(Martin Fowler)

izquotes.com

Danksagung

- Vorlesungsmaterialien von Prof. Dr. Detlef Seese wurden als Basis verwendet
- Unterstützung bei der technischen und inhaltlichen Gestaltung des Vorlesungsmaterials leisteten:

Jóakim v. Kistowski

Dietmar Ratz, Joachim Melcher, Roland Küstermann, Jana Weiner, Hagen Buchwald, Matthes Elstermann, Oliver Schöll, Niklas Kühl, Tobias Diederich

Inhalt und Ziele

- Warum benötigen wir Schleifen?
- Welche Arten von Schleifen gibt es und worin unterscheiden sie sich?
- **while**-, **do**-, **for**-Schleifen
- Endlosschleifen
- Sprungbefehle **break** und **continue**
- Markierte Anweisungen

Syntax while-Anweisung

```
while (<Bedingung>) {  
    <Anweisungsfolge>  
}
```

```
while (<Bedingung>)  
    <Anweisung>
```

Schleifenkörper



Die <Bedingung> ist ein Ausdruck vom Typ **boolean**

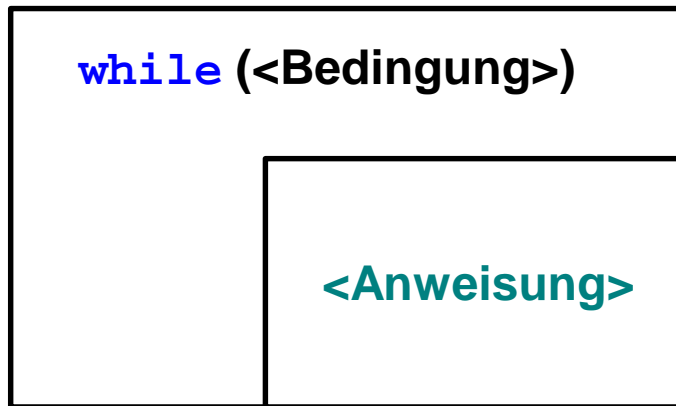
Semantik:

- prüfe die Bedingung
- ist diese wahr, so wird der Körper ausgeführt
- ist die Bedingung falsch, so wird die Schleife verlassen, d.h. die Anweisungen nach dem Schleifenkörper werden bearbeitet

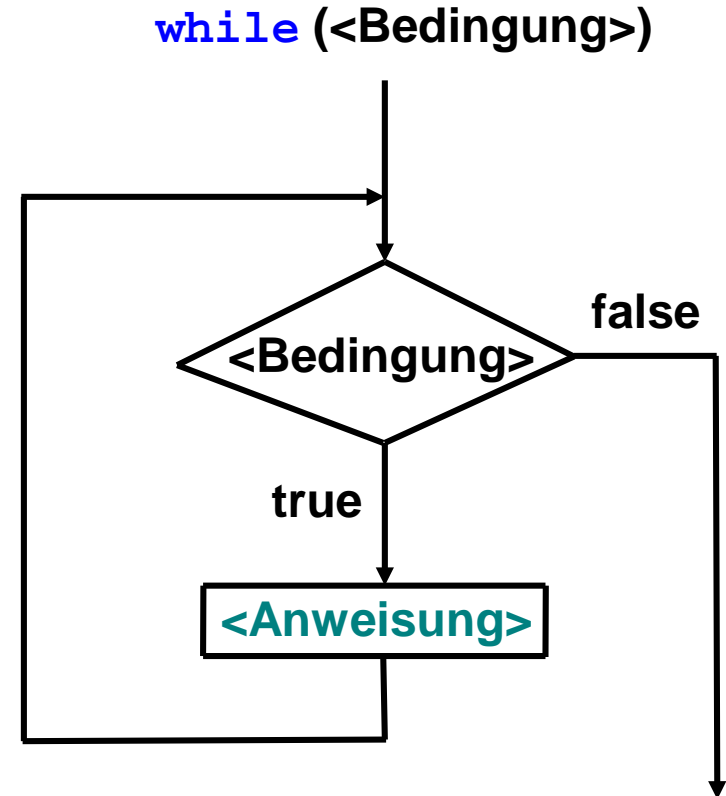
Die **while**-Anweisung bezeichnet man auch als **abweisende** Schleife

Synonyme: **while**-Anweisung / **while**-Schleife / **Schleife mit Eingangsprüfung**

```
while (<Bedingung>)  
    <Anweisung>
```



Struktogramm



Programmablaufplan (PAP)

```
// (a) Berechnung der Quersumme
```

```
//      (z.B.: Quersumme von 1234 ist 10)
```

```
Scanner input = new Scanner(System.in);
```

```
System.out.print("Positive ganze Zahl: ");
```

```
long a = input.nextLong();
```

```
System.out.print("Die Quersumme von " + a + " ist ");
```

```
    0  
long qs = ;
```

```
    a > 0
```

```
while ( ) {
```

```
    qs = qs + a % 10;
```

```
    a = a / 10;
```

```
}
```

```
System.out.println(qs);
```

`%10` liefert die letzte Dezimalstelle
`/10` hängt die letzte Dezimalstelle ab

```
// (b) Berechnung der Fakultät:  
//       $k! = k * (k-1) * (k-2) * \dots * 3 * 2 * 1$   
Scanner input = new Scanner(System.in);  
System.out.print("Positive ganze Zahl: ");  
long k = input.nextLong();  
System.out.print(k + "! = ");  
long f = 1;  
while (k > 1) {  
    f = f * k;  
    k = k - 1;  
}  
System.out.println(f);
```


Beispiele: `while`-Schleife

(c) Größter Gemeinsamer Teiler (GGT):

Gesucht ist der größte gemeinsame Teiler zweier `positiver ganzer` Zahlen

Lösungsansatz: Euklidischer Algorithmus (Erinnerung an V01)

mathematische Beobachtung:

Seien $a > b > 0$ ganze Zahlen und $g = \text{GGT}(a,b)$,
dann ist g auch $\text{GGT}(a - b, b)$.

Iterationsschritt: ersetze a durch $a - b$

In irgendeinem Schritt sind die aktuell betrachteten Zahlen
gleich. Sie gleichen dann dem GGT der Ausgangszahlen.

Kern des Programms:

```
Scanner input = new Scanner(System.in);
System.out.print("Ganze Zahl a = ");
int a = input.nextInt();
System.out.print("Ganze Zahl b = ");
int b = input.nextInt();
System.out.print("Der GGT von " + a + " und " + b + " ist ");
```

```
while(a != b) {
    if (a > b)
        a -= b;
    else
        b -= a;
}
```

```
System.out.println(a);
```

Vorsicht Falle!

```
// (d)
    double x = 15;           // Endlosschleife!
    while (x > 10);          // Rumpf der while-Schleife besteht
        x = x - 10;          // nur aus leerer Anweisung (wg. ;)

// (e)
    while (x > 0)
        x = Math.sqrt(x);    // Nur die erste Anweisung gehört
        x = x - 10;          // zum Rumpf der while-Schleife

// (f)
    while (true)             // Endlosschleife
        x = 5;

// (g)
    while (false)            // Compilierungs-Fehler,
        x = 5;              // da Anweisung unerreichbar
```

Wie stoppt man eine Endlosschleife?

```
while (true) {  
  
    // Anweisungen  
  
    // hier sollte ein break  
  
    // oder ein return stehen,  
  
    // return wird später behandelt  
  
}
```

Beispiel:

```
int x = 1;
while (true) {
    System.out.println(x);
    if (x == 10) break;
    // unterbricht die aktuelle Schleife
    x = x + 1;
}
```

do-Anweisung - Schleife mit Ausgangsprüfung

Syntax: **do**-Anweisung

do

<Anweisung>

while (<Bedingung>);

do {

<Anweisungsfolge>

} **while** (<Bedingung>);

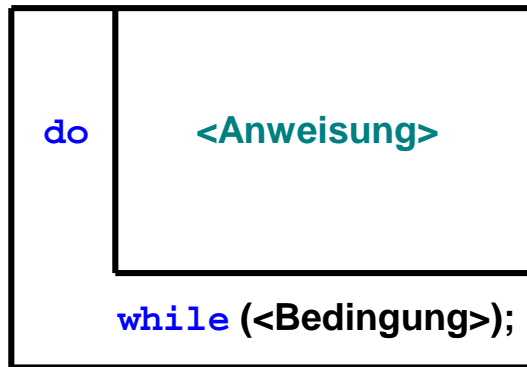
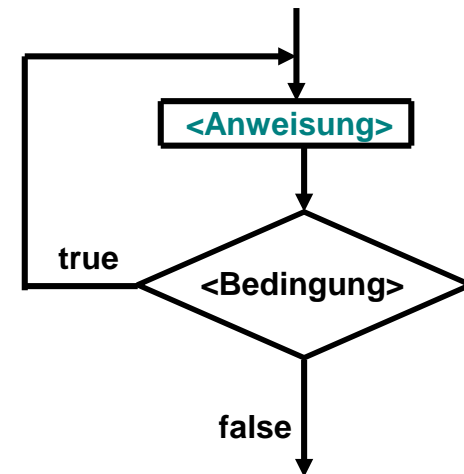
Anweisungen des
Schleifenkörpers



Die <Bedingung> ist ein Ausdruck
vom Typ **boolean**

Semantik:

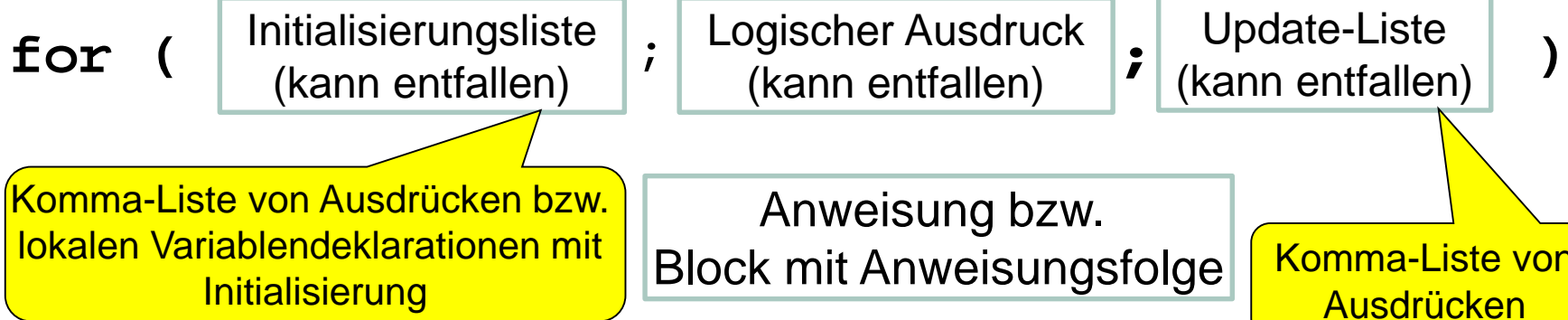
Nach der Initialisierung startet die `do`-Schleife mit der Abarbeitung des Schleifenkörpers. Erst danach wird die Bedingung geprüft. Bei positivem Ausgang des Tests wird wieder der Schleifenkörper durchlaufen usw. Bei negativem Ausgang wird die Schleife beendet.

**Struktogramm****Programmablaufplan (PAP)**

Beispiel:

```
int i = 1;
do {      // Schleifenkörper wird mindestens einmal ausgeführt,
          // auch dann, wenn die Bedingung nicht erfüllt ist
    System.out.println(i);
    i = i + 1;
} while (i < 6);
```


Syntax:

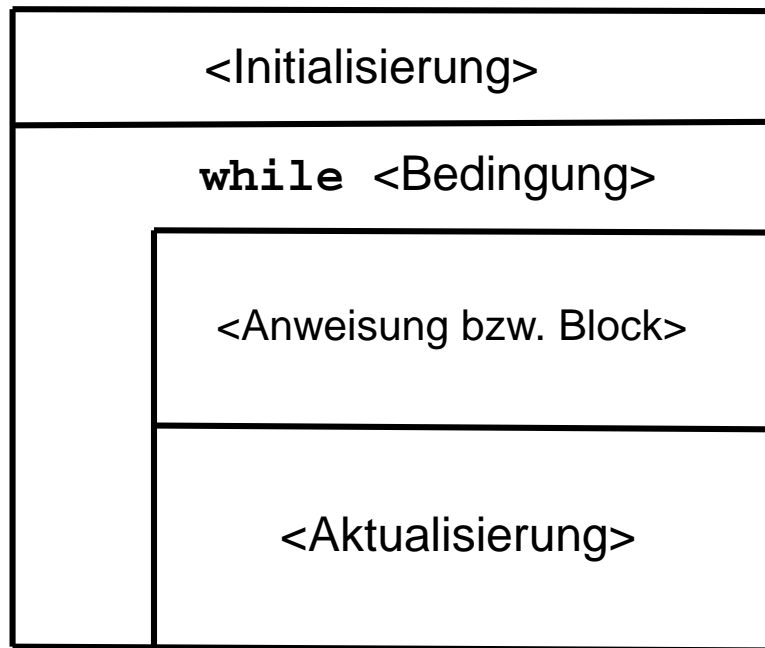


Wirkung:

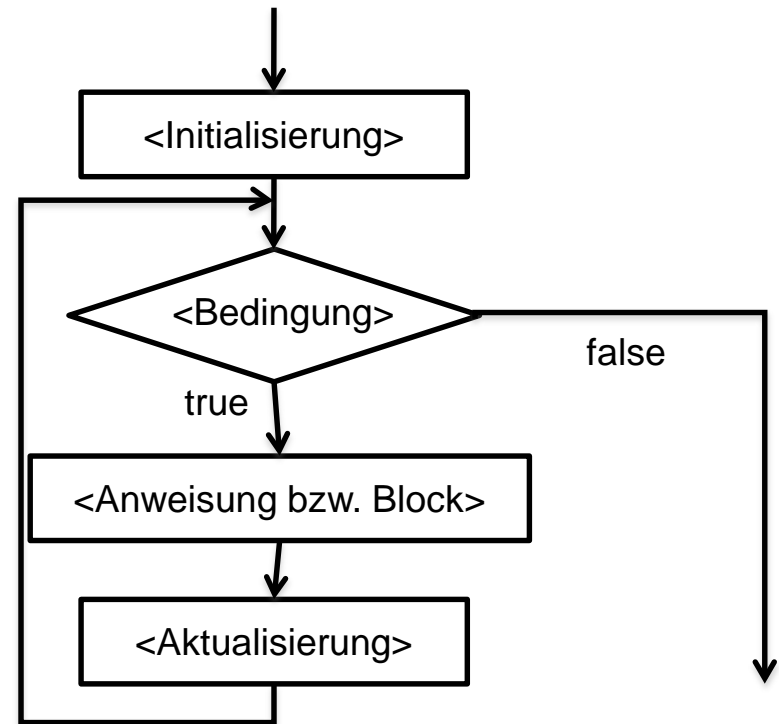
1. Die Initialisierungsliste wird ausgeführt.
2. Der logische Ausdruck wird ausgewertet.
3. Ist der Wert **true**, dann wird die Anweisung bzw. der Block (Anweisungsfolge) ausgeführt, danach die Update-Liste ausgeführt und wieder zu Schritt 2 gesprungen.
4. Ist der Wert jedoch **false**, so ist die **for**-Anweisung beendet.

Weitere
Nebenbedingungen
folgen am Beispiel

for (<Initialisierung>_{opt}; <Bedingung>_{opt}; <Aktualisierung>_{opt})
<Anweisung bzw. Block>



Struktogramm



Programmablaufplan (PAP)

Semantik der **for**-Schleife:

Die Schleife ist äquivalent zu folgender **while**-Schleife:

```
{<Initialisierung>;  
  while (<Bedingung>) {  
    <Anweisung bzw. Block>      // des Schleifenkörpers  
    <Aktualisierung>;  
  }  
}
```

Übung: Zeigen Sie, dass sich **alle** Arten von Schleifen
äquivalent durcheinander ersetzen lassen.

```
// (c) Einmaleins
long j, k, jMax, kMax;
Scanner input = new Scanner(System.in);
System.out.println("Obere Schranke fuer j: ");
jMax = input.nextLong();
System.out.println("Obere Schranke fuer k: ");
kMax = input.nextLong();
System.out.println("Die Produkte des Einmaleins");
```

Ausgaben der Form:

```
Obere Schranke fuer j: 5
Obere Schranke fuer k: 3
Die Produkte des Einmaleins
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
3 * 1 = 3
3 * 2 = 6
...
```

Ausgaben:
20
17
14
11
8
5
2

```
// (c) Einmaleins
long j, k, jMax, kMax;
Scanner input = new Scanner(System.in);
System.out.println("Obere Schranke fuer j: ");
jMax = input.nextLong();
System.out.println("Obere Schranke fuer k: ");
kMax = input.nextLong();
System.out.println("Die Produkte des Einmaleins");
for (j = 1; j <= jMax; j++)
    for (k = 1; k <= kMax; k++)
        System.out.println(j + " * " + k + " = " + j*k);
```

Ausgaben der Form:

```
Obere Schranke fuer j: 5
Obere Schranke fuer k: 3
Die Produkte des Einmaleins
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
3 * 1 = 3
3 * 2 = 6
...
```

Vorsicht Falle!

```
// (d)
    for (;;) {                                // Endlosschleife!
        System.out.println("Netzstecker ziehen!");
    }

// (e)
    for (int a = 0; a < 5; a++) {
        System.out.println(a + " " + a*a*a);
    }
    System.out.println(a); // unzulässig!
                           // Variable a ist nur in
                           // der Schleife bekannt
```

Vorsicht Falle!

```
// (f)
    for (int p=0, long q=7; p<q; p++, q-- )
        ;
                                     // unzulässige 2. Typangabe

// (g)
    int p;
    long q;
    for (p=0, q=7; p<q; p++, q-- )      // ok
        . . .;

// (h)
    int u;                               // u hier bereits dekl.
    for (int u=0, v=7; u<v; u++, v--)
        . . .;
```

break-Anweisung

Syntax: break-Anweisung

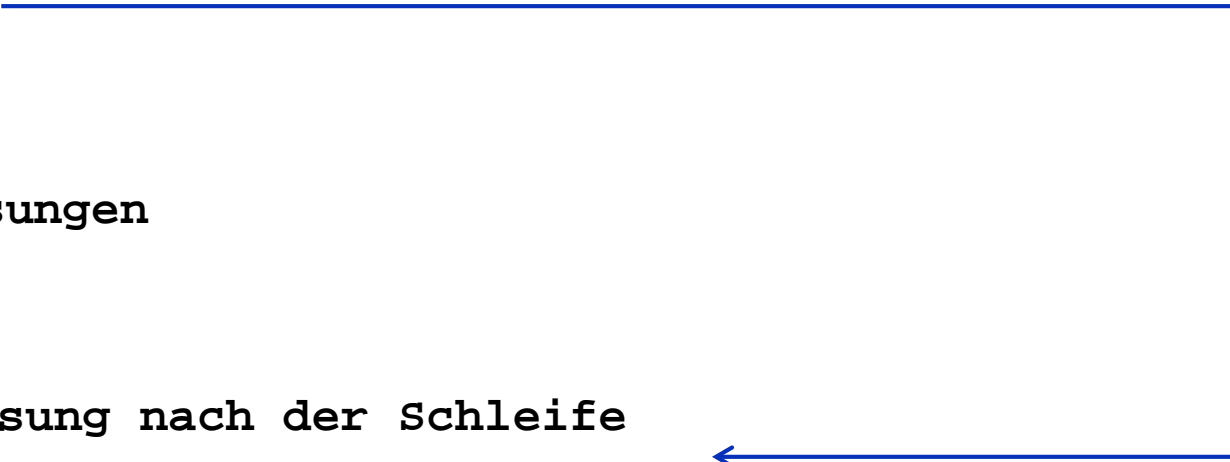
```
break <Bezeichner>opt ;
```

- Eine **break**-Anweisung sollte nur in **switch**-, **while**-, **do** oder **for**-Anweisungen stehen.
- Der <Bezeichner> **muss** Marke einer markierten **switch**-, **while**-, **do**- oder **for**-Anweisung sein, welche das **break** enthält

Semantik (ohne Marke): Die **break**-Anweisung beendet die Ausführung der **innersten**, das **break** umschließenden Schleife bzw. **switch**-Anweisung


Beispiele:

```
while ( Bedingung ) {  
  
    // Anweisungen  
  
    if (...) {  
        break;  
    }  
  
    // Anweisungen  
  
}  
// 1. Anweisung nach der Schleife
```



search:

```
for (i = 0; i < arrayOfInts.length; i++) {  
    for (j = 0; j < arrayOfInts[i].length; j++) {  
        if (arrayOfInts[i][j] == searchfor) {  
            foundIt = true;  
            break search;  
        }  
    }  
}  
  
// 1. Anweisung nach der Schleife
```



continue-Anweisung

Syntax: `continue`-Anweisung

`continue` <Bezeichner>_{opt} ;

- darf nur in einer `while`-, `do`- oder `for`-Anweisung vorkommen
- der <Bezeichner> muss Marke einer markierten Wiederholungsanweisung (`while`-, `do`- oder `for`-Anweisung) sein, welche das `continue` enthält

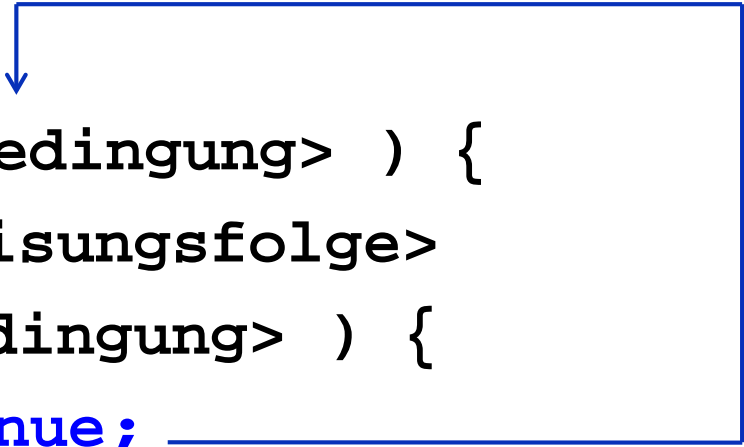
Semantik (hier nur ohne Marke):

Bewirkt einen Sprung an das Ende der innersten, das `continue` umschließenden Schleife und beginnt gegebenenfalls eine erneute Iteration.

Bei einer `while`- oder `do`-Schleife wird also die Abbruchbedingung (boolescher Ausdruck) ausgewertet, bei einer `for`-Schleife werden vorher noch die Aktualisierungs-Ausdrücke ausgewertet.

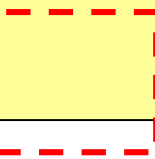
Je nach Ausgang wird der Schleifenkörper erneut ausgewertet oder die Schleife verlassen.

Beispiel:




```
while ( <Bedingung> ) {  
    // <Anweisungsfolge>  
    if ( <Bedingung> ) {  
        continue;  
    }  
    // <Anweisungsfolge>  
}  
  
// erste <Anweisung> nach Schleife
```

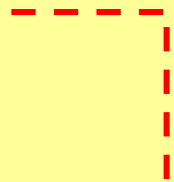
```
while (...) {  
    ...  
    if (...)  
        break;  
}
```



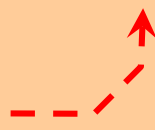
```
while (...) {  
    if (...)  
        continue;  
    ...  
}
```



```
for (i = 0; i < 100; i++) {  
    if (...)  
        break;  
    ...  
}
```



```
for (i = 0; i < 10; i++) {  
    if (...)  
        continue;  
    ...  
}
```



- Die Anweisung **break** beendet die Ausführung der innersten (das **break** umschließenden) Schleife.
- Die Anweisung **continue** beendet den aktuellen Durchlauf der innersten (das **continue** umfassenden) Schleife.

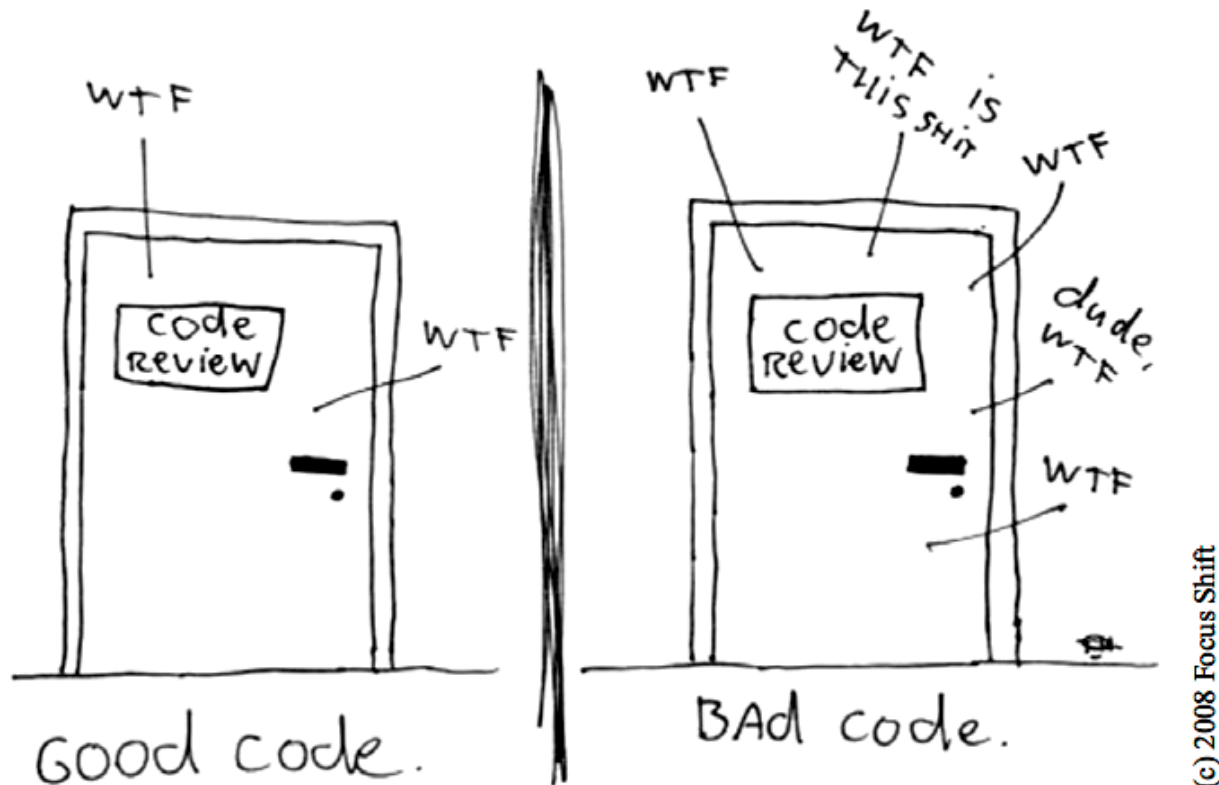
```
for (int i = 0; i < 50; i++) {  
    if (i == 26)  
        break;  
    if (i % 9 != 0) // i ist nicht durch 9 teilbar  
        continue;  
    System.out.println(i);  
}  
int i = 0;  
while (true) {  
    i++;  
    int j = i * 30;  
    if (j == 990)  
        break;  
    if (i % 10 != 0) // i ist nicht durch 10 teilbar  
        continue;  
    System.out.println(i);  
}
```

Ausgaben:

0
9
18
10
20
30

Fragen?

The ONLY valid measurement
OF code QUALITY: WTFs/minute



(c) 2008 Focus Shift