

Anweisungstypen in Java

❖ In Java gibt es verschiedenen Typen von Anweisungen:

✓ **Deklarationsanweisung (declaration statement)**

✓ **Zuweisung (assignment)**

✓ **Konditionale Anweisung (if statement)**

✓ **Return Anweisung (return statement)**

▢ **Schleifenanweisungen (iteration statements)**

Schleifenstrukturen

- ❖ **Definition Schleifenkörper (loop body):** Eine Anzahl von Anweisungen, die innerhalb der Schleifenanweisung durchlaufen wird.
- ❖ **Definition Schleifeneintrittsbedingung (loop entry condition):** Muss wahr sein, damit der Schleifenkörper (erneut) exekutiert wird.
- ❖ In der strukturierten Programmierung gibt es 3 Typen von Schleifenstrukturen:
 - **Zählschleife:** Eine Repetition, in der bereits vor Beginn der Schleifenausführung klar ist, wie oft der Schleifenkörper insgesamt durchlaufen werden muss.
 - **while-Schleife:** Eine Repetition, in der die Schleifenbedingung *vor jedem Eintritt in den Schleifenkörper* abgefragt wird.
 - **do-while-Schleife:** Eine Repetition, in der die Schleifenbedingung *nach jeder Exekution des Schleifenkörpers* abgefragt wird.

Zählschleife

- ❖ Allgemeine Struktur:

```
for (Zählerinitialisierung;  
     Schleifeneintrittsbedingung;  
     Weiterschaltung)
```

Schleifenkörper

- ❖ Beispiel:

```
for ( int k = 0; k < 100; k = k+1 )  
    System.out.print("hello ");
```

- ❖ Dies ist ein Beispiel einer **Null-Indizierten** Schleife:
Die Schleifenvariable k läuft von 0 bis 99.

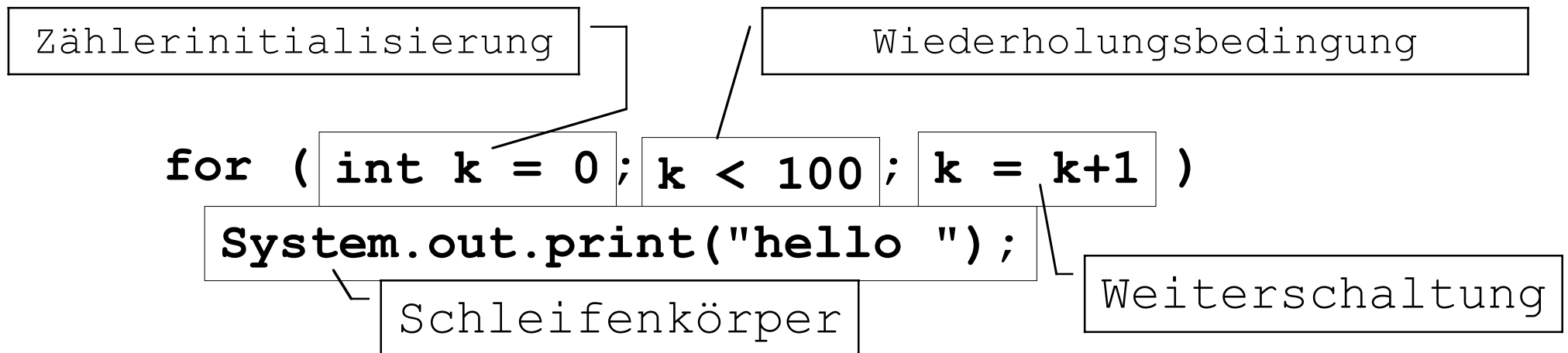
Zählschleife

- ❖ Allgemeine Struktur:

```
for (Zählerinitialisierung;  
     Schleifeneintrittsbedingung;  
     Weiterschaltung)
```

Schleifenkörper

- ❖ Beispiel:



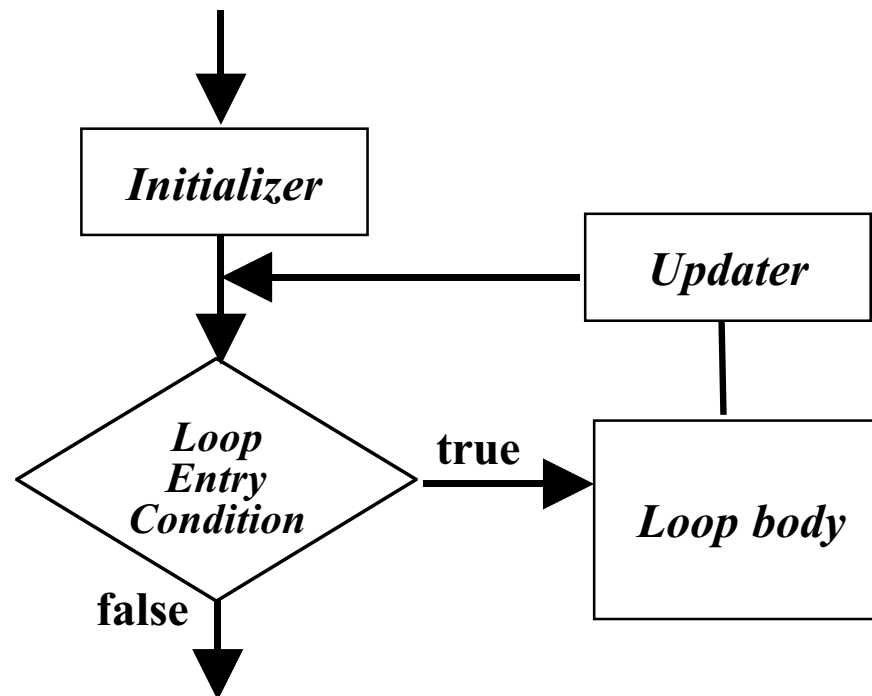
- ❖ Dies ist ein Beispiel einer **Null-Indizierten** Schleife:
Die Schleifenvariable `k` läuft von 0 bis 99.

Die Zählschleife (*for*-Anweisung)

❖ Syntax:

```
for ( Initializer ; Loop Entry Condition ; Updater )  
  Loop body
```

❖ Semantik:



Der Schleifenkörper kann eine einzelne Anweisung sein oder ein Block von Anweisungen (sog. Verbundanweisung)

Gültigkeitsbereich für Schleifenvariablen

- ❖ Eine Schleifenvariable ist eine lokale Variable.
- ❖ Wenn k innerhalb der **for**-Anweisung deklariert ist, kann es nicht außerhalb des Schleifenkörpers verwendet werden:

```
for (int k = 0; k < 100; k++)  
    System.out.println("Hello");  
System.out.println("k = " + k); // Syntaxfehler: k nicht deklariert
```

k++ ist identisch mit k = k+1

- ❖ Wenn k außerhalb der **for**-Anweisung deklariert ist, kann es auch außerhalb benutzt werden:

```
int k = 0; // Deklaration der Schleifenvariable  
for (k = 0; k < 100; k++)  
    System.out.println("Hello");  
System.out.println("k = " + k); // Jetzt kann k außerhalb des  
// Schleifenkörpers benutzt werden
```

Zählschleifen können nach oben oder unten zählen

- ❖ Eine Zählschleife initialisiert den **Zähler** mit einem Anfangswert und zählt 0 oder mehr Iterationen, bis die Grenze der Schleife (*loop bound*) erreicht ist.
- ❖ Die **Schleifeneintrittsbedingung** testet, ob die Grenze der Schleife erreicht worden ist.

`k--` ist identisch mit `k = k-1`

```
public void countdown() {  
    for (int k = 10; k > 0; k--)  
        System.out.print(k + " ");  
    System.out.println("BLASTOFF");  
} // countdown()
```

- ❖ Bei der **Weiterschaltung** muss ein Fortschritt in Richtung Schleifengrenze erzielt werden, sonst gibt es eine unendliche Schleife.
- ❖ **Definition Unendliche Schleife (Infinite loop):** Eine Schleife, die niemals ihre Grenze erreicht.

Unendliche Zählschleifen

❖ Beispiele

`k+=2` ist identisch mit `k = k+2`

```
for (int k = 0; k < 100 ; k--) // k bekommt die Werte 0, -1, -2, ...
    System.out.println("Hello");

for (int k = 1; k != 100 ; k += 2) // k bekommt die Werte 1, 3, ..., 99, 101, ...
    System.out.println("Hello");

for (int k = 98; k < 100 ; k = k / 2) // k bekommt die Werte 98, 49, ..., 0, 0, ...
    System.out.println("Hello");
```

- ❖ In jedem dieser Fälle macht die *Weiterschaltung* überhaupt keinen Fortschritt in Richtung Schleifengrenze und die Schleifeneintrittsbedingung wird deshalb nie **false**.

Lesbarkeit von Programmen

- ❖ Einrückung (Indentierung) erhöht die Lesbarkeit von Programmen
- ❖ Aber: Die Bedeutung einer Anweisung wird durch zusätzliche Leer- und Tabulator-Zeichen bzw. Zeilenumbrüche **nicht** verändert.
- ❖ Beispiel: äquivalente Zählschleifen

```
for (int k = 10 ; k > 0 ; k--)  
    System.out.print (k + " ");  
System.out.println( "BLASTOFF" ); // Nach der Schleife  
  
for (int k = 10 ; k > 0 ; k--)  
System.out.print (k + " ");  
System.out.println("BLASTOFF");  
  
for (int k = 10 ; k > 0 ; k--) System.out.print(k + " ");  
System.out.println("BLASTOFF");  
  
for  
(int k = 10 ; k > 0 ; k--)  
System.out.print(k + " ");  
System.out.println("BLASTOFF");
```

Blöcke als Schleifenkörper

- ❖ **Verbundanweisung** (*Compound statement*) oder **Block**: Eine Sequenz von Anweisungen, die in {}-Klammern eingeschlossen ist.
- ❖ **Schleifenkörper**: Kann eine einfache Anweisung oder eine Verbundanweisung sein.

```
for (int k = 0; k < 100; k++)           // Drucke 0 5 10 15 ... 95
    if (k % 5 == 0)                    // Schleifenkörper ist bedingte Anweisung
        System.out.println("k= " + k);
```

```
for (char k = 'a' ; k <= 'z'; k++)     // Drucke 'a' 'b' 'c' ... 'z'
    System.out.print(k + " ");         // Einfache print() Anweisung
```

```
for (int k = 1 ; k <= 10; k++) {       // Drucke 5 10 15 20 ... 50
    int m = k * 5;
    System.out.print(m + " ");
}
```

Verbundanweisung

```
for (int k = 1 ; k <= 10; k++)
    int m = k * 5;
    System.out.print (m + " ");
```

// Schleifenkörper
// Fehler: Zugriff auf m außerhalb der Schleife!

Tipp: Immer Klammern schreiben!

Geschachtelte Zählschleifen

- ❖ Nehmen wir mal an, Sie sollen folgende Tabelle mit 4 Reihen und 9 Spalten drucken:

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36

- ❖ Hier kann man eine **geschachtelte** Zählschleife nehmen:
Die **äußere Schleife** (*outer loop*) druckt die 4 Reihen.
Die **innere Schleife** (*inner loop*) druckt die 9 Spalten.

```
for (int row = 1; row <= 4; row++) {           // Für jede der 4 Reihen
    for (int col = 1; col <= 9; col++) {       // Für jede der 9 Spalten
        System.out.print(col * row + "\t");
    }                                           // innere Schleife: Druckt insgesamt 36 Zahlen
    System.out.println();                       // Starte eine neue Reihe
}                                               // äußere Schleife
```

Geschachtelte Zählschleifen (cont.)

- ❖ Drucke das folgende Dreiecksmuster

```
#####  
####  
###  
##  
#
```

Reihe	Spalte (6 - Row)	j: Anzahl der #
1	6-1	5
2	6-2	4
3	6-3	3
4	6-4	2
5	6-5	1

- ❖ Lösung mit geschachtelter Zählschleife:

```
for (int row = 1; row <= 5; row++) { // Für jede Reihe  
    for (int j = 1; j <= 6 - row; j++) { // Drucke j #'s in der Reihe  
        System.out.print('# ');  
    }  
    System.out.println(); // Auf zur nächsten Reihe  
}
```

While- und do-while-Schleifen

❖ Wenn die Anzahl der Iterationen unbekannt ist, dann benutzen wir eine **while-** oder **do-while-**Schleife.

❖ **do-while-**Strukturen:

– Suche nach der Studentin "Erika Wilson" und drucke ihre CATS Punkte:

```
do the following steps
  read a record from the file
while Erika Wilson's record is not read
  compute Erika Wilson's CATS
return CATS as the result
```

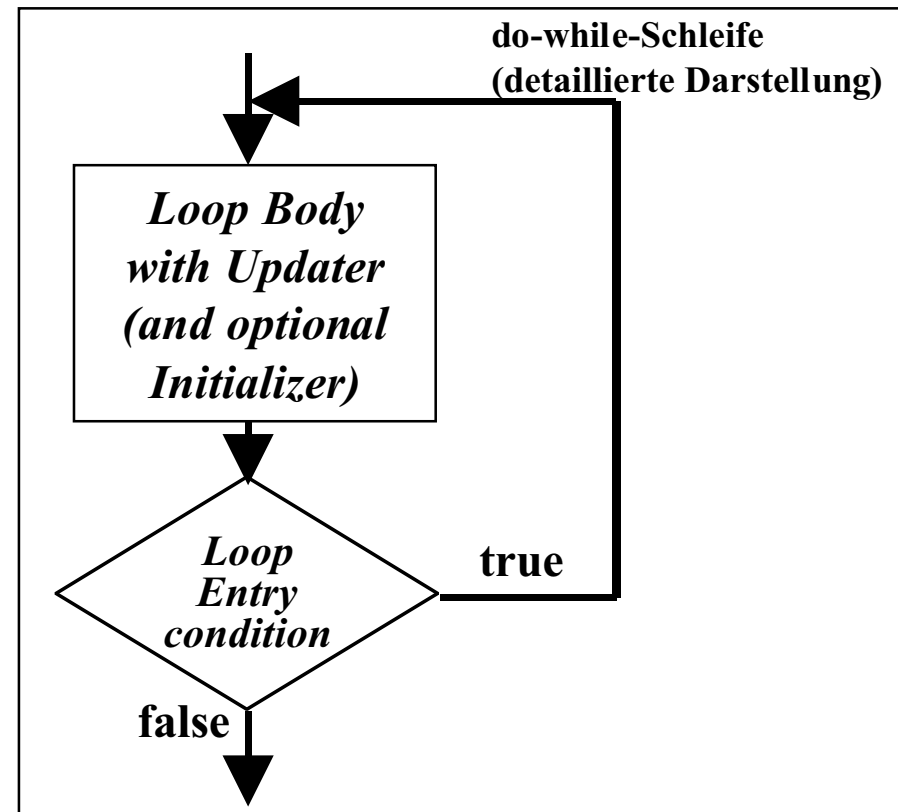
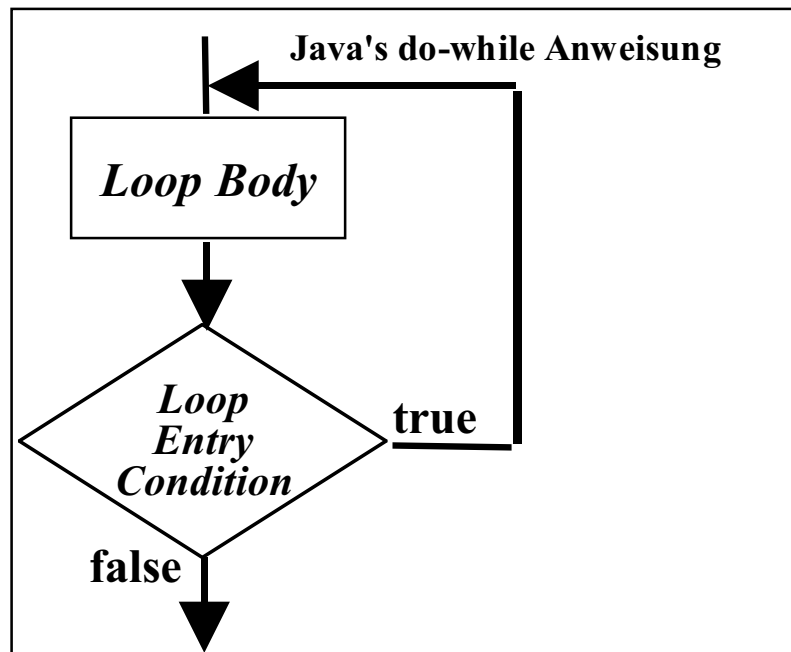
–

– Berechne die durchschnittliche Anzahl von Studenten in Info I:

```
initialize sumOfStudents and numOfLectures to 0
do the following steps
  read a number in the audience of Info I
  add it to the sumOfStudents
  add 1 to numOfLectures
while the user does not want to stop
  divide sumOfStudents by numOfLectures giving average
```

Wann soll man die *do-while*-Schleife verwenden?

- ❖ Die **do-while**-Schleife sollte man benutzen, wenn man weiß, dass die Schleife mindestens einmal exekutiert werden muss.



*Beispiel für eine **do-while**-Schleife: Validierung von Eingabedaten*

- ❖ **Problem:** Mache es unmöglich, fehlerhafte Klausurergebnisse einzugeben. Gültige Ergebnisse: 0-100 (weder -10 noch 155 sind akzeptierbare Ergebnisse)
- ❖ **Algorithmus:** Benutze eine **do-while**-Schleife für diese Aufgabe, denn der Benutzer macht eventuell mehr als einen Versuch, eine gültige Zensur einzugeben:

Initialisierung und Weberschaltung
finden in derselben Anweisung statt

```
do
  get the next grade from user           // Initialisierung
  if the grade < 0 or (grade > 100 and grade != 9999) // Fehlerfall
    print an error message
  while the grade < 0 or (grade > 100 and grade != 9999) // Wächtertest
  // usw...
```

while-Schleifen

❖ Das sogenannte **3N+1 Problem**: Sei N eine beliebige positive Zahl.

❖ Gegeben seien zwei Regeln für die Generierung von Zahlen:

Fall	Nächste Zahl
N ist ungerade	$N = 3 * N + 1$
N ist gerade	$N = N / 2$

❖ Behauptung: Bei wiederholter Anwendung der Regeln erreicht N nach endlich vielen Schritten den Wert 1

❖ Pseudocode für den Algorithmus:

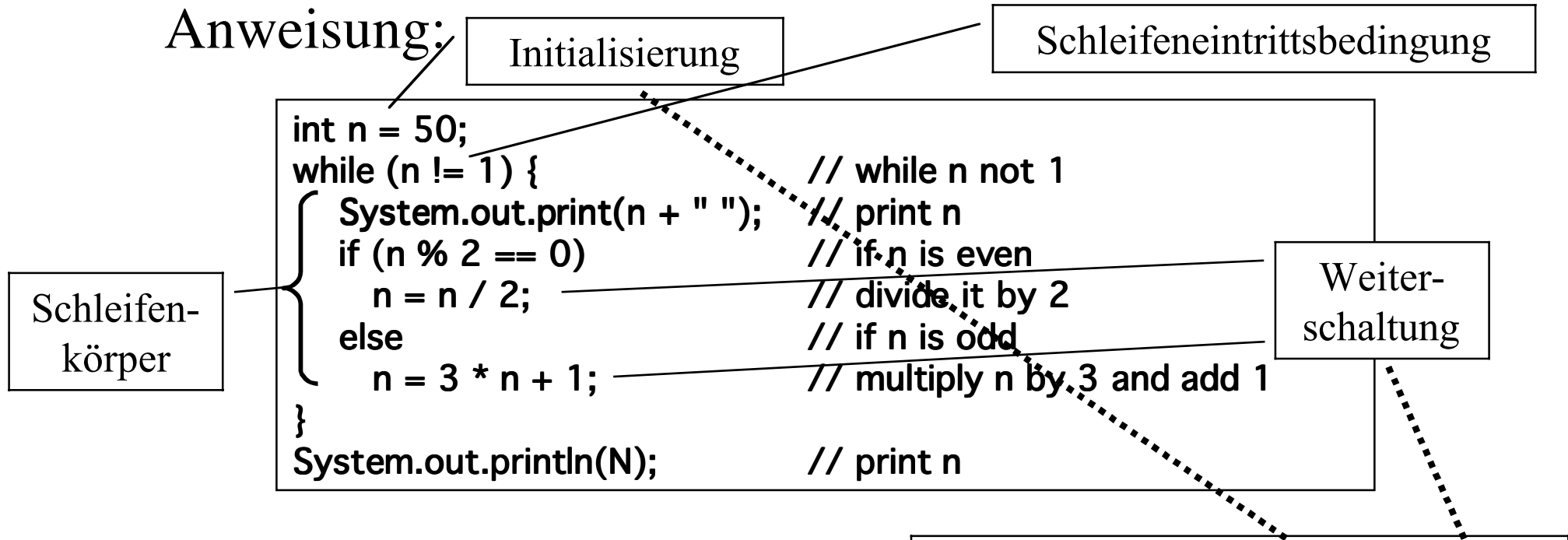
```
while N is not equal to 1, do: {  
    print N.  
    If N is even, divide it by 2.  
    If N is odd, multiply N by 3 and add 1.  
}  
print N
```

Die Schleife iteriert,
solange gilt: $N \neq 1$

Die Schleife terminiert, wenn
 N gleich 1 ist.

Die *while*-Anweisung in Java

- ❖ Lösung des 3N+1 Problems mit Hilfe einer **while**-Anweisung:



- ❖ Java's **while**-Anweisung:

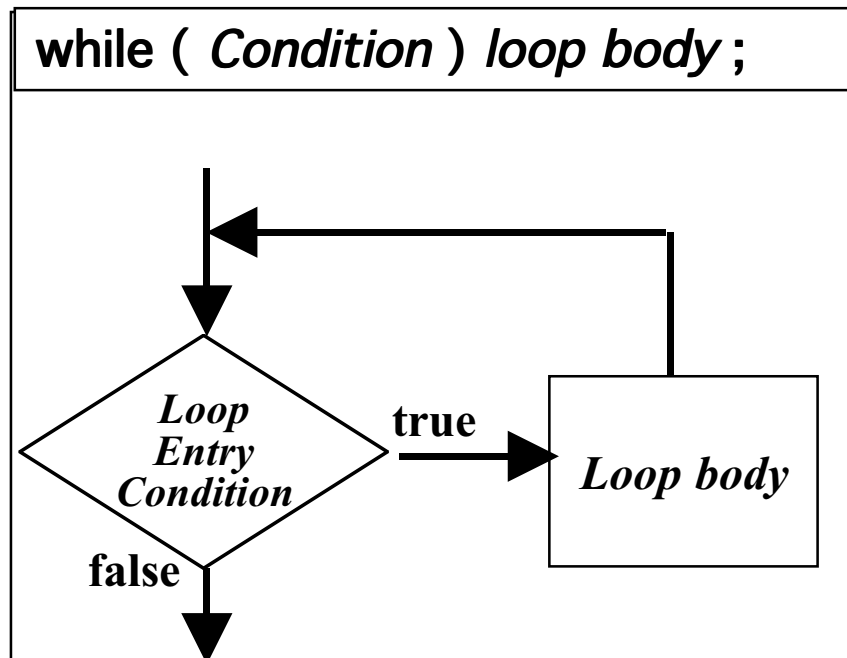
```
while ( loop entry condition )
    loop body
```

Anders als die Zählschleife hat die **while**-Schleife keine eingebaute Initialisierung und Weiterschaltung (diese müssen vom Programmierer bereitgestellt werden) .

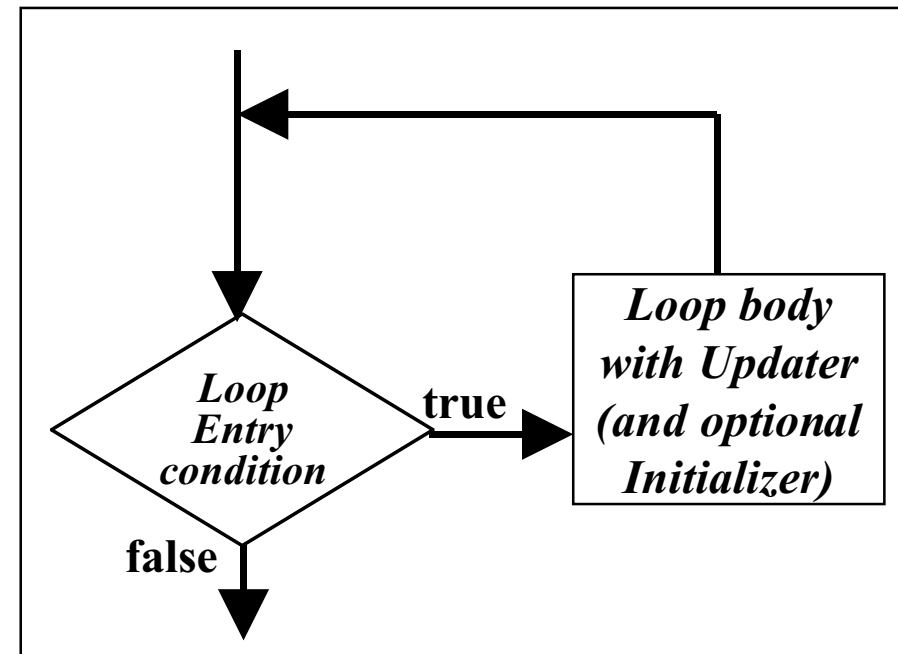
Terminierung von *while*-Schleifen

- ❖ Java's **while**-Anweisung garantiert keine automatische Terminierung.
- ❖ Eine terminierende **while-Schleife** muss die Weiterschaltung innerhalb des Schleifenkörpers enthalten. Die Weiterschaltung muss so gestaltet sein, dass die Schleifeneintrittsbedingung irgendwann nicht mehr erfüllt ist, damit die Schleife terminiert.

Java's *while*-Anweisung



terminierende *while*-Schleife



while-Schleife mit Wächterwert

- ❖ **Problem:** Berechne den Mittelwert der Info I-Klausurergebnisse. Die Klausurergebnisse werden als Liste von Zahlen von einer Hilfskraft eingetippt, der **Wächterwert** (*sentinel value*) 9999 signalisiert das Ende der Listeneingabe.

Initialisierung der Variable **grade**

```
initialize runningTotal to 0 // Initialisierung
initialize count to 0
prompt and read the first grade // Lies erstes Ergebnis
while the grade entered is not 9999 { // Wächterwert
    add it to the runningTotal
    add 1 to the count
    prompt and read the next grade // Update
}
if (count > 0) // Anzahl der Ergebnisse darf
    divide runningTotal by count // nicht 0 sein
    output the average as the result
```

Exekution des Schleifenkörpers, solange Schleifeneintrittsbedingung ("Wächterwert nicht erreicht") erfüllt ist.

Weiterschaltung der Variable **grade**

Zusammenfassung Schleifenstrukturen

- ❖ Eine **Zählschleife** sollte man benutzen, wenn man von vornherein weiss, wieviele Iterationen benötigt werden.
Für die Implementation benutzen wir Java's **for-Anweisung**.
- ❖ Eine **while-Schleife** sollte man verwenden, wenn der Schleifenkörper eventuell überhaupt nicht exekutiert werden soll.
Für die Implementation benutzen wir Java's **while-Anweisung**.
- ❖ Eine **do-while-Schleife** sollte man verwenden, wenn eine oder mehr Iterationen durchgeführt werden. Für die Implementation benutzen wir Java's **do-while-Anweisung**.
- ❖ Eine **unendliche Schleife** ist das Resultat einer inkorrekten spezifizierten Initialisierung, Weiterschaltung oder einer schlecht gewählten Schleifeneintrittsbedingung
 - oder sie ist beabsichtigt :-)