UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Programmierkurs Java

Kontrollstrukturen

Raphael Allner
Institut für Telematik
29. November 2019

Überblick

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

1. Bedingte Ausführung
   - **`if`**- und **`if-else`**-Anweisung
   - **`switch`**-Anweisung
2. Wiederholungsanweisung
   - **`while`**-Schleife
   - **`do-while`**-Schleife
   - **`for`**-Schleife

Übungsaufgabe

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

1. Laden Sie die Datei Kontrollstrukturen.java aus dem Moodle herunter
2. Verfolgen Sie die Vorlesung um die Aufgaben zu erfüllen
3. Überprüfen Sie Ihren Code
   1. Kompilieren mit `javac Kontrollstrukturen.java`
   2. Ausführen in der JVM mit `java Kontrollstrukturen`

Übungsaufgabe

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Ziel: Ausgabe der besten Rundenzeit

0	1	2	3	4	5	6	7	8	9
16.5	14.3	15.0	14.5	14.9	13.9	14.2	14.5	15.7	14.8

Algorithmus:

1. Start: Nehme die erste Rundenzeit anfangs als Bestzeit an
2. Wähle nacheinander die Rundenzeiten zwei bis zehn und vergleiche
   - mit der aktuell besten Zeit
     - Wenn sie kleiner ist, nehme sie als neue Bestzeit
3. Ende: Gib die ermittelte Bestzeit aus

Wiederholungsanweisung

Bedingte Ausführung

Quellcode des Beispielprogramms
Laufzeiten.java

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

```java
public class LapTimes {
    public static void main(String[] args) {

        double[] lapTimes = {16.5, 14.3, 15.0, 14.5, 13.9, 14.2, 14.5, 15.7, 14.8};

        // Nimm zunaechst an, dass die erste Laufzeit die Bestzeit ist
        double bestTime = lapTimes[0];

        // Ermittle die Bestzeit durch Vergleich aller aufgezeichneten Zeiten









        // Gib die ermittelte Bestzeit aus
        System.out.println("Beste Rundenzeit: " + bestTime);
    }
}
```

Kontrollstrukturen

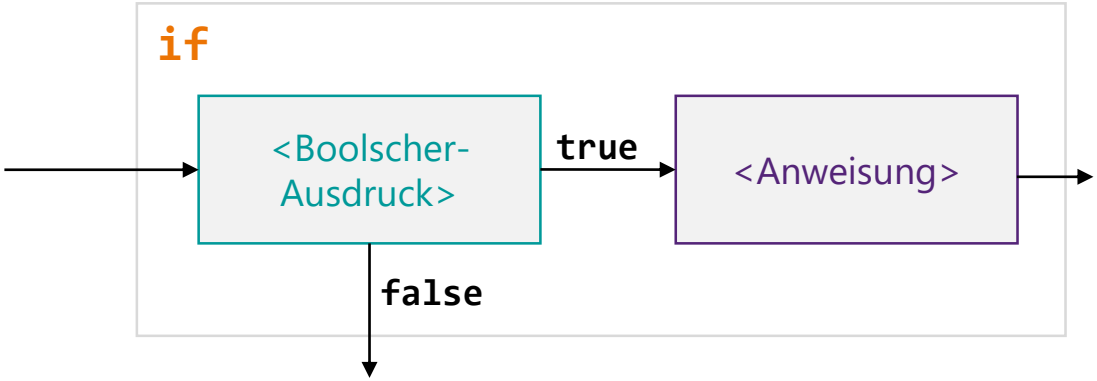
Bedingte Ausführung und Wiederholungsanweisung

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

条件语句

Bedingte Ausführung:

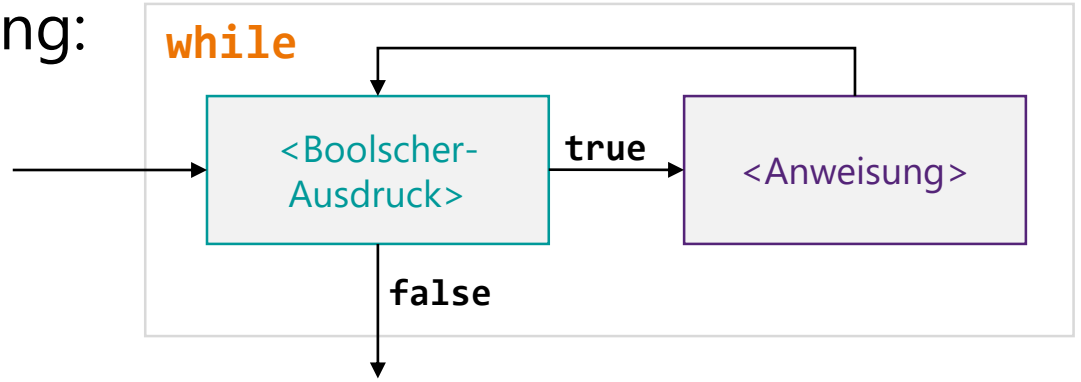
- **`if`**
- **`switch`**



重复语句

Wiederholungsanweisung:

- **`while`**
- **`do ... while`**
- **`for`**



Diese …

… können ineinander überführt werden

… sind jeweils eine Anweisung

Kontrollstrukturen

Bedingte Ausführung

Bedingte Ausführung
`if`-Abfrage

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Die `if`-Abfrage dient dazu zu entscheiden **ob** etwas ausgeführt werden soll

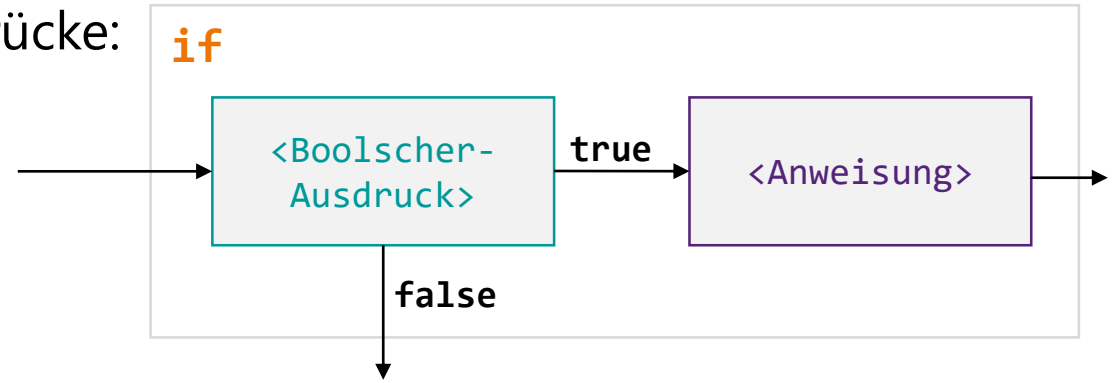
If语句：如果满足条件，则执行

Syntax: `if (<Boolescher-Ausdruck>) <Anweisung>;`

Semantik: Wenn `<Boolescher-Ausdruck>` zu **`true`** ausgewertet wird, führe `<Anweisung>` (bzw. Blockanweisung {}) aus.

Beispiel für boolesche Ausdrücke:

- `true`
- `1 > 2`
- `i != 12`
- `u – 2 > z - 3`



Beispiele für if-Anweisung:

- `if (true) System.out.println("Tautologie");`

Bedingte Ausführung
`if`-Abfrage, Fallstrick

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Beispiel:

```
1  int i=3;   if (2*i > 1000) i = i / 2;   i = 3 * i;
```

Auch wenn es optisch nicht so aussieht, aber:

- `i = 3 * i;` **ist nicht mehr Teil der if-Abfrage**
- Wird also immer ausgeführt,
  → unabhängig vom Wert von <Boolescher-Ausdruck>

Lösung:

- **Optisch formatieren**, für bessere Nachvollziehbarkeit

```
1  int i = 3;
2  if (2 * i > 1000)
3      i = i / 2;
4  i = 3 * i;
```

```
1  int i = 3;
2  if (2 * i > 1000) {
3      i = i / 2;
4  }
5  i = 3 * i;
```

Bedingte Ausführung
`if`-Abfrage, Blockanweisung

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Wenn `i = 3 * i` Teil der **`if`**-Abfrage werden soll, hilft eine Blockanweisung

```
1  int i=3;   if(2*i > 1000) { i = i / 2;   i = 3 * i; }
```

Besser formatiert (Java Code Style):

```
1  int i = 3;
2  if (2 * i > 1000) {
3      i = i / 2;
4      i = 3 * i;
5  }
```

Es ist **generell sinnvoll**, Blockanweisungen zu benutzen

- z. B. falls man später noch eine Anweisung hinzufügen will

Bedingte Ausführung
Probleme bei mehreren Bedingungen

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Was macht man um mehrere Bedingungen zu abzufragen? 更多条件

Beispiel:

```
1  if (a > 100){
2      a = a - 100;
3  }
4  if (a <= 100){
5      System.out.println("a zu klein");
6  }
```

Probleme:

- Es müssen **mehrere Bedingung** formuliert werden (Hier: 1 x positiv, 1x negativ)
- Wenn die erste Anweisung **gemeinsamen Speicher modifiziert** (wie hier geschehen), dann kommt es ggf. zu **ungewolltem Verhalten**

Bedingte Ausführung
`if-else`-Abfrage

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Um **`if`** Anweisungen zu verknüpfen: **`else`**

Syntax:

```
if (<Boolescher-Ausdruck1>)
        <Anweisung1>;
else if (<Boolescher-Ausdruck2>)
        <Anweisung2>;
else
        <Anweisung3>;
```

Semantik: **Falls** `<Boolescher-Ausdruck1>` zu **`true`** ausgewertet wird, führe `<Anweisung1>` aus, falls nicht überprüfe ob `<Boolescher-Ausdruck2>` zu **`true`** ausgewertet wird, führe `<Anweisung2>`, **ansonsten** führe `<Anweisung3>` aus.

不满足if的条件，否则else的运行

Die Anweisungen können wieder beliebige Ausdrücke sein

- Also z. B. auch weitere if-Anweisungen (Verschachtelungen)

Bedingte Ausführung
`if-else`-Abfrage

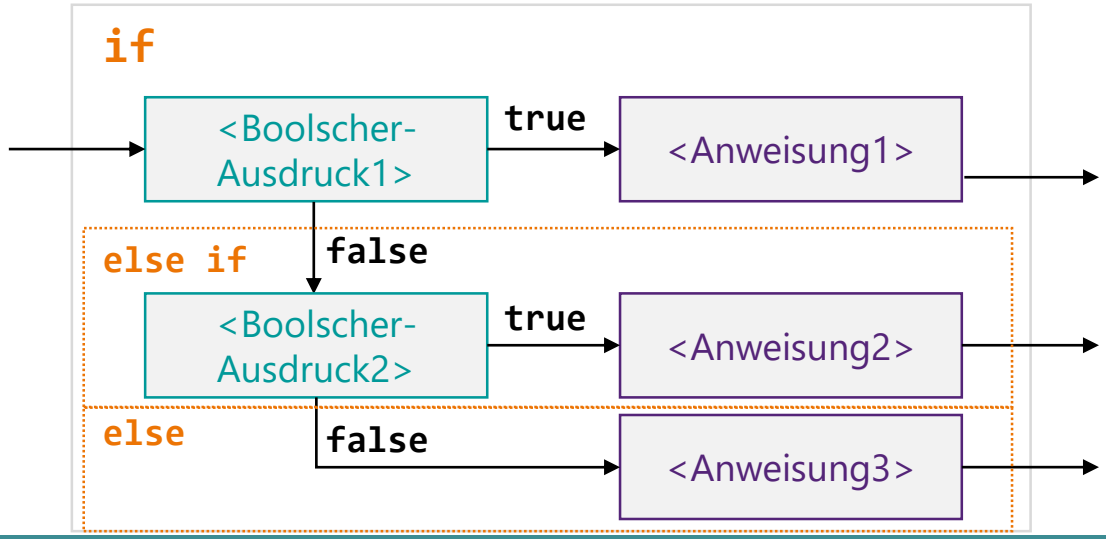
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Um **`if`** Anweisungen zu verknüpfen: **`else`**

Syntax:

```
if (<Boolescher-Ausdruck1>)
        <Anweisung1>;
else if (<Boolescher-Ausdruck2>)
        <Anweisung2>;
else
        <Anweisung3>;
```

Ablauf:



Bedingte Ausführung
`if-else`-Abfrage

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Beispiel:

```
1  if(a>100) if(b>100) b=b+a; else a=a*2;
```

```
1  if (a > 100)
2          if (b > 100)
3              b = b + a;
4  else
5          a = a * 2;
```

```
1  if (a > 100) {
2          if (b > 100) {
3              b = b + a;
4          } else {
5              a = a * 2;
6          }
7  }
```

- Explizit angeben, was gemeint ist
- Blockanweisungen verwenden
- Code lesbar machen – Hier liegt nicht in der Kürze die Würze

Bedingte Ausführung
`if-else`-Abfrage

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Beispiel - Refueling.java

```java
public class Refueling{
    public static void main(String[] args) throws NumberFormatException{

        // Der maximale Fuellstand ist konstant
        final int TANK_MAX_LITER = 50;
        if((args.length == 1)){
            //Hier werden Fehler durch einen sog. try-catch-Block abgefangen
            try{
                int eingefuellteLiterBenzin = Integer.parseInt(args[0]);
                if (eingefuellteLiterBenzin < (TANK_MAX_LITER/2)) {
                    System.out.println("Der Tank ist noch nicht mal halb voll");
                } else if (eingefuellteLiterBenzin <= 0) {
                    // Diese Anweisung wird nie ausgefuehrt.
                    System.out.println("Nichts? Weniger als nichts?");
                } else if (eingefuellteLiterBenzin == (TANK_MAX_LITER/2)) {
                    System.out.println("Der Tank ist halb voll");
                } else if (eingefuellteLiterBenzin > (TANK_MAX_LITER/2) && eingefuellteLiterBenzin < TANK_MAX_LITER) {
                    System.out.println("Der Tank ist fast voll");
                } else if (eingefuellteLiterBenzin == TANK_MAX_LITER){
                    System.out.println("Der Tank ist voll");
                } else {
                    System.out.println("Der Tank laeuft ueber");
                }
            } catch(NumberFormatException ex) {
                // Hier wird abgefangen, dass jemand etwas anderes als eine Zahl als Uebergabeparameter angibt.
                System.err.println("Das war keine Zahl! Damit kann ich nichts anfangen.");
            }
        } else {
            // Hier wird abgefangen, dass jemand mehr als einen Uebergabeparameter angibt.
            System.err.println("Es wird eine einzelne Zahl erwartet.");
        }
    }
}
```

Bedingte Ausführung
`switch`-Anweisung

选择语句

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Bei vielen alternativen Fällen: die `switch`-Anweisung

Syntax:

```
switch (<Ausdruck>){
        case <Vergleichswert1>:对比值
                <Anweisung1>;
                break;
        case <VergleichswertN>:
                <AnweisungN>;
                break;
        default:
                <AnweisungN+1>;
}
```

文本或常数

Literal oder Konstante

Hier können auch mehrere Anweisungen stehen.

一种选择的结束

break verhindert ein „Durchfallen" zur nächsten Anweisung. Programm verlässt die Switch-Abfrage.

如果没有选择值则运行default

default-Zweig: Wird ausgeführt, falls keiner der Vergleiche positiv ausfällt. *(Optional)*

Einsatzzweck begrenzt

对比值可用的数据类型

- Auf Vergleiche von **char**, **byte**, **short**, **int** und deren Wrapperklassen sowie Enums und Strings
- Keine **boolschen Ausdrücke** wie i<3, a>=40, etc. möglich 不能用boolean
- Nur Konstanten und Literale für **case**-Vergleichswerten erlaubt

Bedingte Ausführung
`switch`-Anweisung – Beispiel:

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Beispiel:

```
1  int month = 2;
2  switch (month) {
3          case 1:
4                  System.out.println("Januar");
5                  break;
6          case 2:
7                  System.out.println("Februar");
8                  break;
9          case 12:
10                 System.out.println("Dezember");
11                 break;
12         default:
13                 System.out.println("Ungueltige Eingabe.");
14                 break;
15 }
```

Siehe Monate.java

Kontrollstrukturen

Wiederholungsanweisungen: Schleifen

Wiederholungsanweisungen
`while`-Schleife

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

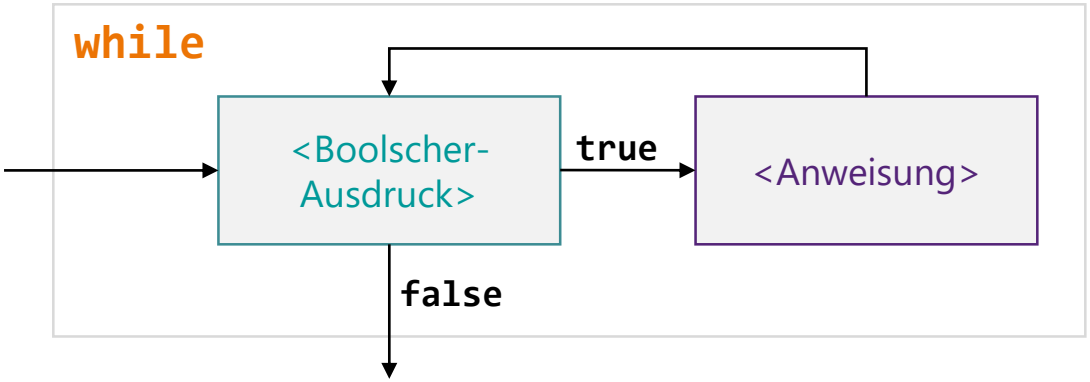
Falls etwas **immer wieder** ausgeführt werden soll: `while`-Schleife

while循环语句

Syntax: `while (<Boolescher-Ausdruck>) <Anweisung>;`

Semantik: Wiederhole die (Block-)Anweisung, solange `<Boolescher-Ausdruck>` zu **`true`** ausgewertet wird

Ablauf:



Wiederholungsanweisungen
`while`-Schleife

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Falls etwas **immer wieder** ausgeführt werden soll: `while`-Schleife

Syntax: `while (<Boolescher-Ausdruck>) <Anweisung>;`

Semantik: Wiederhole die (Block-)Anweisung, solange `<Boolescher-Ausdruck>` zu **`true`** ausgewertet wird

Beispiele:

```
1 while (i > 10) i--;
```

```
1 while (i > 10) {
2         System.out.println(i);
3         i--;
4 }
```

Wiederholungsanweisungen
`do-while`-Schleife

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Wenn etwas **mindestens einmal** ausgeführt werden soll: `do-while`-Schleife

do-while循环语句

Syntax: `do <Anweisung>; while (<Boolescher-Ausdruck>);`

Semantik: Wiederhole `<Anweisung>`, solange `<Boolescher-Ausdruck>` zu **`true`** ausgewertet wird. Führe jedoch mindestens 1x `<Anweisung>` aus.

Beispiele:

```
1  do i--; while (i > 10);
```

```
1  do {
2        System.out.println(i);
3        i--;
4    } while (i > 10);
```

Wiederholungsanweisungen
`do-while`-Schleife

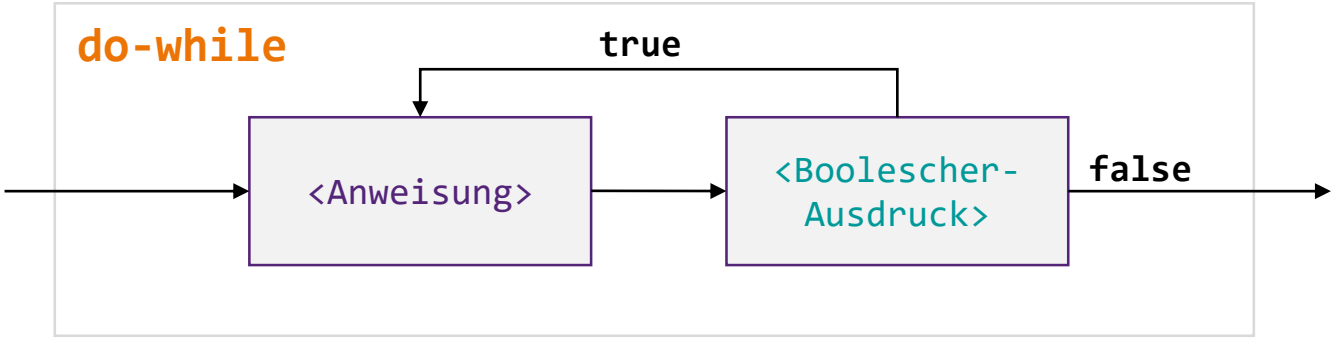
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Wenn etwas **mindestens einmal** ausgeführt werden soll: `do-while`-Schleife

Syntax: `do <Anweisung>; while (<Boolescher-ausdruck>);`

Semantik: Wiederhole `<Anweisung>`, solange `<Boolescher-Ausdruck>` zu **`true`** ausgewertet wird. Führe jedoch mindestens 1x `<Anweisung>` aus.

Ablauf:



Wiederholungsanweisungen
`for`-Schleife

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Die `for`-Schleife ist **expliziter** und **prägnanter** als `while` oder `do-while`

for循环语句

Syntax:

```
for(<Initialisierung>; <Bedingung>; <Modifikation>)
        <Anweisung>;
```

Semantik: `<Initialisierung>` einer Laufvariable (einmal zu Beginn der Schleifenausführung). Führe `<Anweisung>` **solange** aus, solange `<Bedingung>` (Boolescher Ausdruck) zu **`true`** ausgewertet wird.

`<Modifikation>` der Laufvariable nach jedem Durchlauf.

Beispiel:

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

Initialisierung | Bedingung | Modifikation | Anweisung

Wiederholungsanweisungen
for-Schleife, Ablaufdiagramm

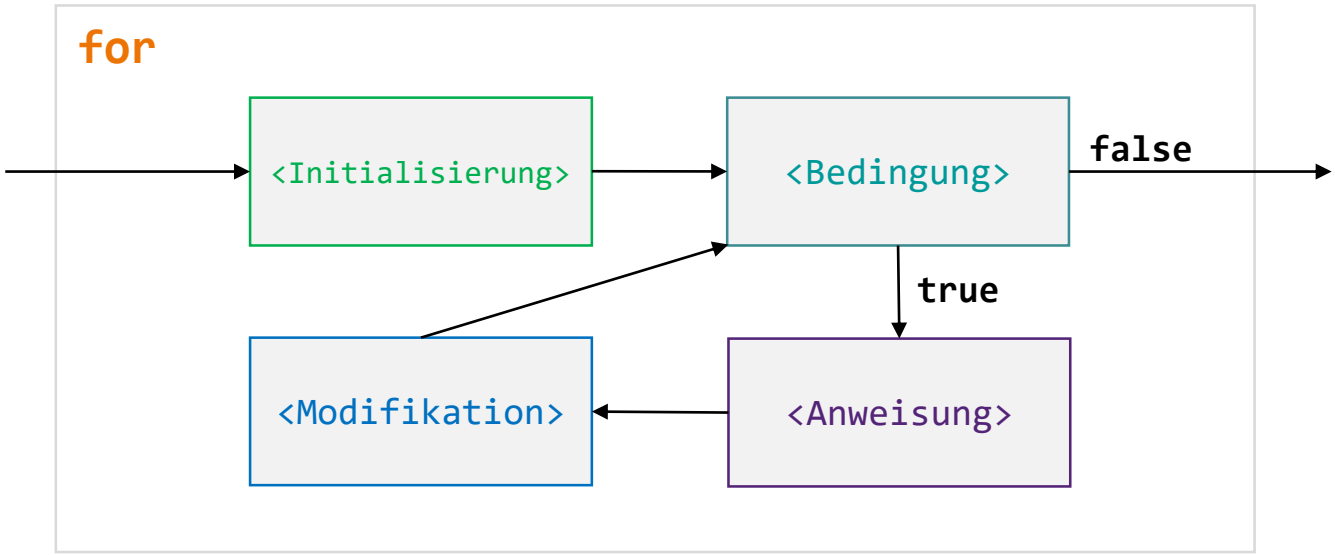
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Beispiel:

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

Initialisierung | Bedingung | Modifikation | Anweisung

Ablauf:



Wiederholungsanweisungen
`for`-Schleife, Abarbeitungsreihenfolge

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Beispiel:

Initialisierung | Bedingung | Modifikation

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

Anweisung

Ablauf:

Initialisierung	→ i = 0;
Bedingung	→ i < 10 ist **true**, also nächster Durchlauf
Anweisung	→ Ausgabe: "i: 0"
Modifikation	→ i++ → i = 1
Bedingung	→ i < 10 ist **true**, also nächster Durchlauf
Anweisung	→ Ausgabe: "i: 1"
Modifikation	→ i++ → i = 2
...	→ Durchlauf für 3, 4, ... bis Ausgabe: "i: 9"
Modifikation	→ i++ → i = 10
Bedingung -> Ende	→ i < 10 ist **false**, also Ende der Schleife

Wiederholungsanweisungen
for-Schleife, Beispiele

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Weitere Beispiele:

```
1 for (int i = 2; i < 100; i = i * i)
2         System.out.println("i=" + i);
```

```
1 int j = 20;
2 for ( ; j > 10; --j)
3           System.out.println("j=" + j);
```

```
1 for ( ; ; );
```

→ Endlosschleife

```
1 for (int i = 0; i < 10; i++)
2         System.out.println(i++);
```

Wiederholungsanweisungen
`for`, `while` oder `do-while`-Schleife?

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

`for`-Schleife
以次数为循环的条件

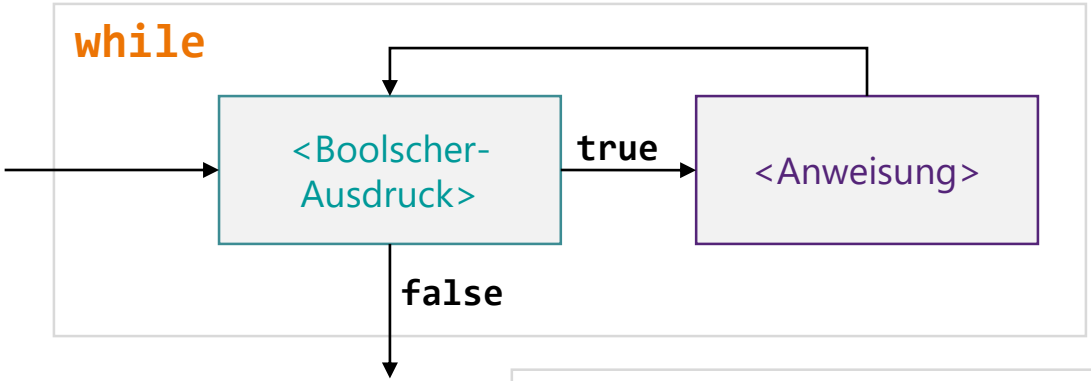
- Üblicherweise orientiert an einem Durchlaufzähler 用于Array序列
- Z.B. Array-Index (vorher bekannt)
- Ist eher explizit 条件提前确定

`while`- bzw. `do-while`-Schleife 以结果为循环的条件

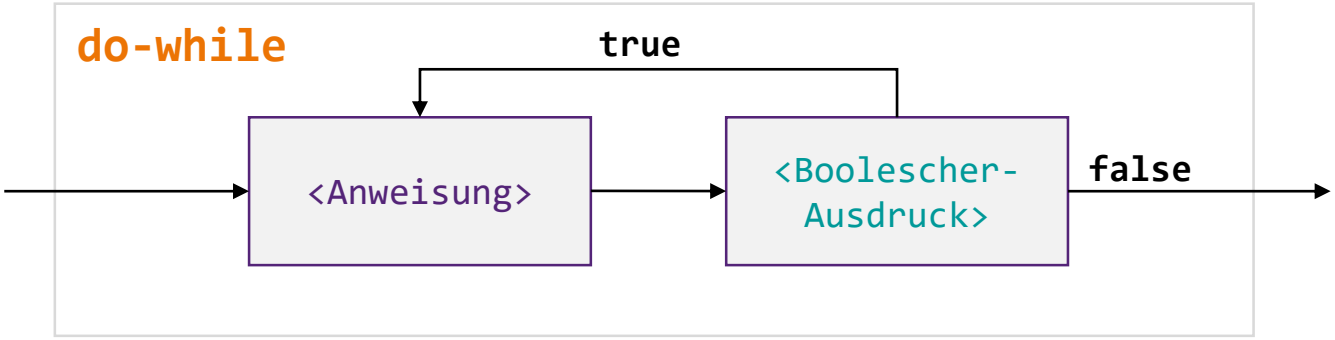
- Eher bei vorher unbekannter Anzahl Wiederholungen
- Einfachere Syntax 语句简单
- Durchlaufzahl schwerer abzuschätzen

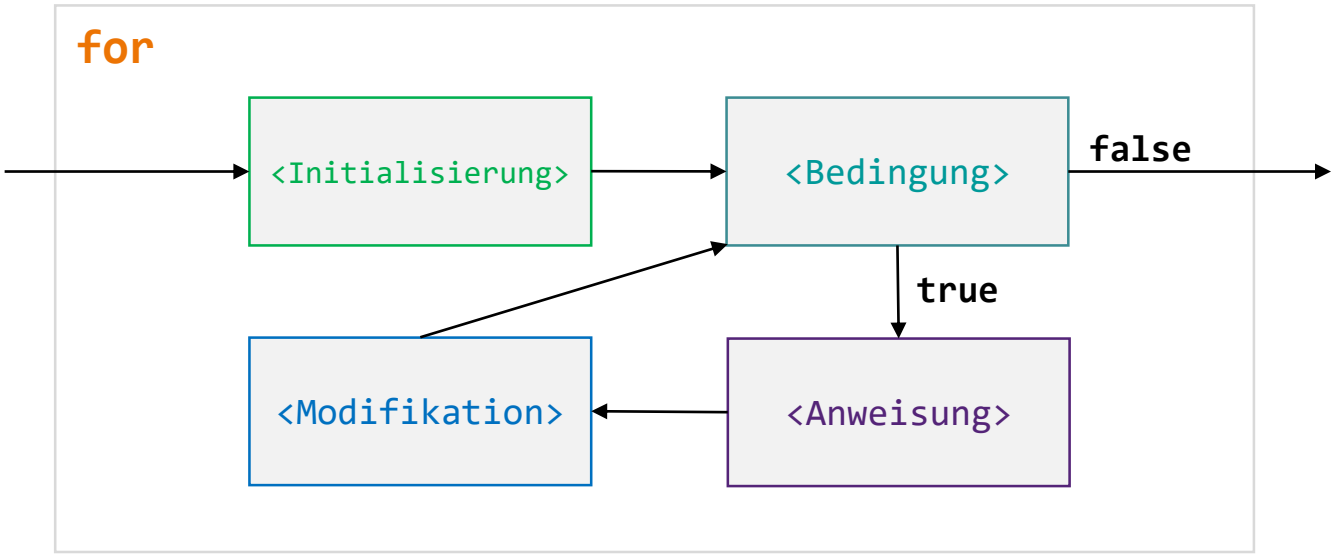
Außerdem:

- **`for`**, **`while`** oder **`do-while`**-Schleifen lassen sich ineinander überführen
- Welche Variante man nutzt, ist daher egal



UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK





Übungsaufgabe

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Ziel: Ausgabe der besten Rundenzeit

0	1	2	3	4	5	6	7	8	9
16.5	14.3	15.0	14.5	14.9	13.9	14.2	14.5	15.7	14.8

Algorithmus:

1. Start: Nehme die erste Rundenzeit anfangs als Bestzeit an
2. Wähle nacheinander die Rundenzeiten zwei bis zehn und vergleiche
   - mit der aktuell besten Zeit
     - Wenn sie kleiner ist, nehme sie als neue Bestzeit
3. Ende: Gib die ermittelte Bestzeit aus

Lösungsvorschlag → Siehe Moodle

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Kontakt

Raphael Allner, M. Sc.
Wissenschaftlicher Mitarbeiter
Institut für Telematik

Universität zu Lübeck
Ratzeburger Allee 160
23562 Lübeck

https://www.itm.uni-luebeck.de/mitarbeitende/raphael-allner.html