

# **Teil 4: Java-Sprachkonstrukte**

## **Kontrollstrukturen**

Modul „Grundlagen der objektorientierten Programmierung mit Java“

Prof. Dr. Cornelia Heinisch

## Agenda

- Selektionsanweisungen
- Iterationsanweisungen

## Es werden folgende Selektionsanweisungen unterschieden:

### ■ Bedingte Anweisung:

- Anweisung soll nur ausgeführt werden, wenn eine bestimmte Bedingung zutrifft.
- Anweisung in Programmiersprache: `if`
- **Falls** die Sonne scheint, machen wir Picknick.

### ■ Einfache Alternative:

- Es soll aus zwei Möglichkeiten (entweder/oder) ausgewählt werden.
- Anweisung in Programmiersprache: `if` und `else`
- **Falls** die Sonne scheint, machen wir Picknick, **andernfalls** gehen wir ins Kino.

### ■ Mehrfache Alternative:

- Es soll aus mehreren Möglichkeiten (entweder/oder/.../oder/ansonsten) genau eine ausgewählt werden.
- Anweisung in Programmiersprache entweder `else if` oder `switch`.
- Wenn die Sonne scheint, machen wir Picknick, **oder wenn** es regnet, gehen wir ins Kino, **oder wenn** es schneit, gehen wir Schlitten fahren, **ansonsten** spielen wir Karten.

## Einfache Alternative

**Falls** Wetter schön



Picknick im Grünen



```
if (Ausdruck)
```

```
    Anweisung_1
```

**Andernfalls**



Kinobesuch




```
else
```


```
    Anweisung_2
```

Der `Ausdruck` in Klammern wird berechnet.

- Trifft die Bedingung zu (der Rückgabewert von `Ausdruck` ist `true`), so wird die `Anweisung_1` ausgeführt.
- Trifft die Bedingung nicht zu (der Rückgabewert von `Ausdruck` ist `false`), so wird die `Anweisung_2` ausgeführt.

## Mehrfache Alternative mit else if

**Falls** Sonne scheint   
Picknick im Grünen

**oder wenn** es regnet   
Kinobesuch

**oder wenn** es schneit   
Schlitten fahren

**andernfalls**   
Karten spielen

```
if (Ausdruck_1)
    Anweisung_1
else if (Ausdruck_2)
    Anweisung_2
else if (Ausdruck_3)
    Anweisung_3
else
    Anweisung_4
```


- In der angegebenen Reihenfolge wird ein Vergleich nach dem anderen durchgeführt.
- Bei der ersten Bedingung, die wahr ist, wird die zugehörige Anweisung abgearbeitet und die Mehrfach-Selektion abgebrochen.
- Statt einer einzelnen Anweisung kann auch ein Block von Anweisungen stehen.
- Der `else`-Zweig ist optional und wird oft zum Abfangen von Fehlern (z. B. ungültige Eingabe durch den Anwender) verwendet.




## Aufgabe 1: Mehrfache Alternative mit else-if

- In das bestehende Projekt `teil_4` eine neue Klasse `WetterTest` mit `main()`-Methode anlegen.
- Es soll ein Programm geschrieben werden, das abhängig vom aktuellen Wetter eine passende Freizeitaktivität ausgibt:
  - Legen Sie eine Variable vom Datentyp `int` mit dem Namen `aktuelles_Wetter` an. Speichert die Variable den Wert 1, so steht dies für Sonne, der Wert 2 steht für Regen der Wert 3 steht für Schnee, der Wert 4 steht für Blitz und der Wert 5 steht für Sturm.
  - Machen Sie eine Ausgabe auf dem Bildschirm, die dem Anwender mitteilt, welche Werte er bei welcher Wetterlage eingeben soll.
  - Lesen Sie die Eingabe des Benutzers in die Variable `aktuelles_Wetter` ein.
  - Verwenden Sie eine `else-if`-Anweisung, um abhängig von der eingegebenen Wetterlage eine entsprechende Ausgabe (siehe vorherige Folie) zu tätigen.

## Mehrfache Alternative mit switch

**Falls** Sonne scheint   
Picknick im Grünen

**oder wenn** es regnet   
Kinobesuch

**oder wenn** es schneit   
Schlitten fahren

**andernfalls**   
Karten spielen

```
switch (Ausdruck)
{
    case Konstante_1:
        Anweisungen_1
        break; /* optional */
    case Konstante_2:
        Anweisungen_2
        break; /* optional */
    case Konstante_3:
        Anweisungen_3
        break; /* optional */
    case Konstante_4:
    case Konstante_5:
        Anweisungen_n
        break; /* optional */
    default: /* optional */
        Anweisungen_d /*optional*/
}
```



## Aufgabe 2: Mehrfache Alternative mit switch

- In das bestehende Projekt `teil_4` eine neue Klasse `WetterTestSwitch` mit `main()`-Methode anlegen.
- Schreiben Sie Ihr Programm derart um, dass jetzt eine `switch`-Anweisung zum Einsatz kommt.

```
switch (aktuelles_Wetter)
{
    case 1: // Sonne
        System.out.println ("Es geht heute zum Picknick!");
        break;
    case 2: // Regen
        .
        .
        .
        .
        .

    default:
        System.out.println ("Ungueltige Eingabe!");
}
```



## Agenda

- Selektionsanweisungen
- Iterationsanweisungen

## Es werden folgende Iterationsanweisungen unterschieden:

### ■ Abweisende Schleife mit **while**:

- Vor der ersten Iteration wird geprüft, ob die Bedingung für einen Schleifendurchlauf erfüllt ist.
- Anweisung in Programmiersprache: `while`
- **Solange** unbelegtes Brötchen da, nimm ein Brötchen und belege es.

### ■ Abweisende Schleife mit **for**:

- Vor der ersten Iteration wird geprüft, ob die Bedingung für einen Schleifendurchlauf erfüllt ist.
- Gleich wie `while`, nur geschicktere Handhabung von Laufvariablen.
- Anweisung in Programmiersprache: `for`
- **Solange** unbelegtes Brötchen da, nimm ein Brötchen und belege es.

### ■ Annehmende Schleife mit **do while**:

- Anweisungen in der Schleife werden erst einmal ausgeführt, bevor nach dem ersten Durchlauf die Bedingungsprüfung für einen weiteren Durchlauf erfolgt.
- Anweisung in Programmiersprache: `do while`
- Lege Brötchen in den Korb, **bis** der Korb voll ist.

## Abweisende Schleife mit while

### Aufgabe: Brötchen für Picknick belegen

**Solange** unbelegtes Brötchen da



belege Brötchen



lege Brötchen in den Picknickkorb



```
while (Ausdruck)
{
    Anweisung_1
    Anweisung_2
}
```

Der `Ausdruck` in Klammern wird berechnet.

- Trifft die Bedingung zu (der Rückgabewert von `Ausdruck` ist `true`), so werden die Anweisungen in der Schleife (nochmals) ausgeführt.
- Trifft die Bedingung nicht zu (der Rückgabewert von `Ausdruck` ist `false`) so werden die Anweisungen, die der `while`-Anweisung folgen, ausgeführt.

## Abweisende Schleife mit for

- Beispiel:

```
for (i = 10; i > 0; i--)  
{  
    printf ("Belege Brötchen");  
    printf ("Lege Brötchen in Picknickkorb");  
}
```

- Allgemeine Form:

```
for (Ausdruck_1; Ausdruck_2; Ausdruck_3)  
    Anweisung(en)
```

- Äquivalente while-Schleife:

```
Ausdruck_1  
while (Ausdruck_2)  
{  
    Anweisungen(en)  
    Ausdruck3;  
}
```

## Eigenschaften for-Anweisung

### Schritte einer for-Anweisung:

- Initialisierung einer Laufvariablen – entspricht `Ausdruck_1`. Zum Beispiel `i = 10`.
- Prüfung der Schleifenbedingung – entspricht `Ausdruck_2`.
- Falls Schleifenbedingung erfüllt:
  - Ausführung der Anweisungen innerhalb der Schleife
  - und Bewertung des `Ausdrucks_3`. Zum Beispiel `i--`.

### Bearbeitung von Arrays:

- Die `for`-Anweisung eignet sich besonders gut, um Arrays zu durchlaufen.
- Die Laufvariable repräsentiert hierbei gleichzeitig den Array-Index.

## Annehmende Schleife mit do-while

### Aufgabe: Waschmaschine befüllen

Lege Wäschestück in Waschmaschine

**Solange bis** Waschmaschine voll.

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

- Zuerst werden die Anweisungen der Schleife einmal ausgeführt. Es wird angenommen, dass die Ausführungsbedingungen zu Beginn erfüllt sind.
- Danach wird der Ausdruck bewertet:
  - Ist der Ausdruck wahr, werden die Anweisungen nochmals ausgeführt.
  - Ist der Ausdruck nicht wahr, wird mit den Anweisungen nach der `do-while`-Schleife fortgefahren.
- Unterschied zu den abweisenden Schleifen:  
Bei der `do-while`-Schleife wird der Schleifenkörper mindestens einmal ausgeführt, bei einer `for`- oder `while`-Schleife kann es auch vorkommen, dass die Anweisungen im Schleifenkörper gar nicht ausgeführt werden.



### Aufgabe 3: do-while-Schleife

- In das bestehende Projekt `teil_4` eine neue Klasse `DoWhileTest` mit `main()`-Methode anlegen.
- Lesen Sie mit Hilfe einer `do-while`-Schleife beliebig viele positive, ganzzahlige Werte von der Tastatur ein und bilden Sie die Summe der eingelesenen Werte.
- Gibt der Benutzer `-1` als Wert ein, so soll die `do-while`-Schleife beendet werden.
- Geben Sie die berechnete Summe auf dem Bildschirm aus.



## Aufgabe 4: Schleifen

- In das bestehende Projekt `teil_4` eine neue Klasse `SchleifenTest` mit `main()`-Methode anlegen.
- Verwenden Sie für die folgende Aufgabe zuerst eine `while`-Schleife, dann eine `do-while`-Schleife und dann einer `for`-Schleife.
- Es soll die Summe einer jeden zweiten Integerzahl berechnet werden. Beginnen Sie mit dem Wert `2` und beenden Sie die Schleife nachdem der Wert `100` addiert wurde. Es soll also die Summe aus  $2 + 4 + 6 + 8 + \dots + 100$  berechnet werden.
- Geben Sie die Summe jeweils nach der `while`-/`do-while`/`for`-Schleife aus.





## Aufgabe 5: Quersumme berechnen

- In das bestehende Projekt `teil_4` eine neue Klasse `QuersummeTest` mit `main()`-Methode anlegen.
- Schreiben Sie ein Programm, das die Quersumme einer Ganzzahl berechnet.
- Die Quersumme der Zahl 23434556 ist zum Beispiel  $2+3+4+3+4+5+5+6 = 32$ .
- Überlegen Sie sich zuerst einen Algorithmus und dann die dazu passende Iterationsanweisung.

## For-each-Schleife = erweiterte for-Schleife

- Mit Hilfe der for-each-Schleife ist der Programmcode für das Durchlaufen eines Arrays kompakter formulierbar.

- Bisherige for-Schleife zum Ausgeben der Personen:

```
for (int i = 0; i < arr.length; i++)  
{  
    System.out.println (arr[i].getName()  
                        + " " + arr[i].getVorname());  
}
```

- Ausgabe mit Hilfe der for-each-Schleife:

```
for (Person element : arr)  
{  
    System.out.println (element.getName()  
                        + " " + element.getVorname());  
}
```

- Es wird eine Variable `element` vom Typ `Person` definiert.
- Typ der Variablen `element` muss dem Typ der Array-Elemente entsprechen.
- Der Variablen `element` wird dann mit jedem Schleifendurchlauf der Wert des jeweils nächsten Array-Elementes zugewiesen.