



LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN



 mobile and
distributed systems group



Javakurs für Anfänger

Einheit 05: Programmablauf

Kyrill Schmid
Lehrstuhl für Mobile und Verteilte Systeme



1. Teil: Programmablauf

- 3 Grundstrukturen von Algorithmen
- Wie bisher: Anweisungsfolgen
- Blöcke und Sichtbarkeitsbereiche

2. Teil: Kontrollstrukturen

- Logische Operatoren
- `if`
- Verschachteltes `if`

Praxis:

- Übungen zu `if`

Lernziele

- Erweiterte Kontrollstrukturen kennenlernen
- Sichtbarkeitsbereiche von Variablen verstehen
- Mit Auswahlstrukturen umgehen können

Bisher: Einfacher sequentieller Ablauf von Befehlen

- Beispiel Student (letzte Stunde):

```
public class Hauptprogramm {  
  
    //Main-Methode  
    public static void main(String[] args){  
  
        Student stud1 = new Student("Peter",3);  
        Student stud2 = new Student("Hanna",5);  
  
        stud1.lernen();  
        stud2.lernen();  
    }  
}
```

In Zukunft wollen wir **komplexere** Abläufe programmieren können!

Komplexere Abläufe verlangen Kontrollstrukturen. Beispiele:

- Wenn ein Student im höheren Semester ist, dann...
- Ein Student lernt solange bis,....
- Alle Studenten machen

Ein Algorithmus lässt sich durch 3 Grundstrukturen beschreiben:

- Anweisungsfolge bzw. Sequenz
 - Wie bisher: Schrittweise Abarbeitung von Befehlen von oben nach unten

- Auswahlstruktur bzw. Selektion
 - Ermöglicht die **bedingte** Ausführung von Anweisungen
 - Bsp.: `if`, `else`, `switch-case`

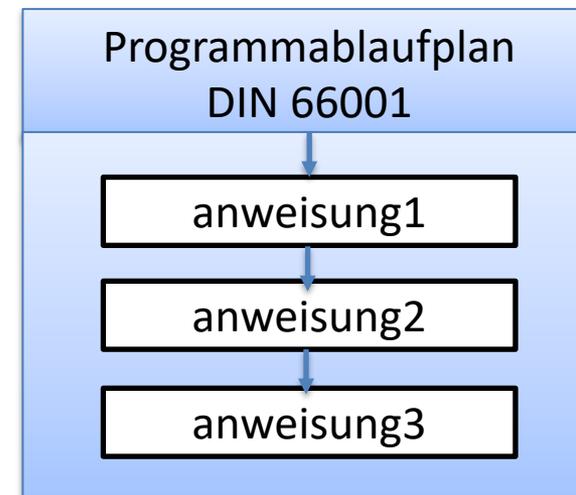
- Wiederholungsstruktur bzw. Iteration oder Schleife
 - **Mehrmalige** Ausführung der gleichen Anweisungen
 - Beispiele: `while-`, `do-`, `for-`Schleifen

Wie bei unseren bisherigen Programmen:

- Die im Quellcode vorhandenen Anweisungen werden sequentiell von oben nach unten abgearbeitet.
- Die Sequenz ist damit die einfachste Kontrollstruktur
- Beispiel:

```
anweisung1;  
anweisung2;  
anweisung3;
```

Darstellung als
Programmablaufplan



Anweisungsfolgen können auch in Blöcken (mit geschweifte Klammern) zusammengefasst werden

- Werden von außen wie eine Einheit betrachtet
- Ein Block kann selbst Variablen definieren, die nur innerhalb des Blocks sichtbar sind
- Ein Block ist selbst wieder eine Anweisung
- Vorteile: Struktur und geschlossene Einheit
- Beispiel:

```
// Beispiel für Blöcke
```

```
anweisung1;
```

```
{ //Begin Block
```

```
  anweisung2;
```

```
  anweisung3;
```

```
} //End Block
```

```
anweisung4;
```



**Anweisungen 2 und 3
bilden eine Einheit**

Die Verwendung von Blöcken wirkt sich auf die **Sichtbarkeit** von Variablen aus!

- Instanzvariablen sind mit `this.instanzvariable` innerhalb der kompletten Klasse sichtbar, also auch in allen Blöcken und Unterblöcken
- Lokale Variablen sind nur innerhalb des Blocks (inklusive aller Unterblöcke) in dem sie definiert wurden sichtbar!

```
(...)//Beispiel
public void doMyVeryBest(int meinWert){

    this.meinWert = 7; // Belegt Instanzvariable mit 7

    int x = meinWert; // Belegt lokale Variabel x mit dem Parameterwert

    { //Begin Block

        y int y = 5; //Int-Variable y wird definiert. Ist nur im Block sichtbar!
        z int z = y+x; //Lokale Variable z = 5+x. x ist hier sichtbar, da Unterblock.

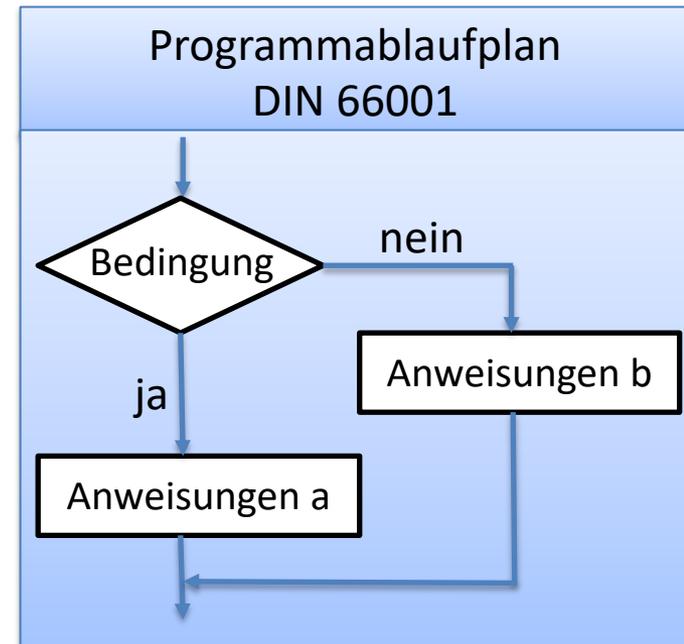
    } // End Block

    x = z; // Fehler: x ist zwar sichtbar, aber z nicht, da außerhalb des Blocks
}
```

Sichtbarkeit von x

Auswahlstrukturen ermöglichen die bedingte Ausführung von Anweisungen

- Ausführung einzelner Anweisungen bzw. Blöcke wird von Erfüllung einer **Bedingung** abhängig gemacht
- Beispiele (im Folgenden):
 - `if`-Anweisung
 - Verschachtelte `if`-Anweisungen
 - `Switch-Case`-Anweisungen



Zweiseitige Auswahlstruktur

Was ist eine Bedingung?

Unter einer Bedingung versteht man einen beliebigen Ausdruck, dessen Auswertung einen Wahrheits- bzw. booleschen Wert (`true` oder `false`) liefert.

Für eine Bedingung kann also entweder

- direkt eine Variable vom Typ `boolean` angegeben werden
- oder einen Methodenaufruf, der einen Wert vom Typ `boolean` zurückgibt
- oder man verwendet **Vergleichsoperatoren** bzw. **logische Operatoren**, wie in den folgenden Tabellen dargestellt:

Vergleichs-Operator	Bedeutung	Priorität
<	kleiner	5
<=	Kleiner gleich	5
>	größer	5
>=	größer gleich	5
==	gleich	6
!=	ungleich	6

Vergleichsoperatoren

Logischer-Operator	Bedeutung	Priorität
!	NICHT	1
&	UND (bitwise)	7
^	XOR (bitwise)	8
	ODER (bitwise)	9
&&	UND	10
	ODER	11

Logische Operatoren

Hinweis: Die Prioritäten können mit Hilfe von runden Klammern beeinflusst werden

```
// Einfache Beispiele für Vergleiche:
```

```
a == 0;      // Ist der Wert von a 0?  
b > 10;      // Ist b größer 10?  
c != 5;      // Ist c ungleich 5?  
zahl <= 100; // Ist zahl kleiner gleich 100?
```

```
// Einfache Beispiele für Logische Ausdrücke:
```

```
(a < 5) && (b > 2); // Ist a kleiner 5 UND b größer 2?
```

 // Kann auch ohne Klammern geschrieben werden, wegen Prioritäten!
// >,< hat Priorität 5 und && hat geringere Priorität 10

```
zahl >= 100 || zahl < 10; // Ist zahl größer gleich 100 ODER kleiner 10?
```

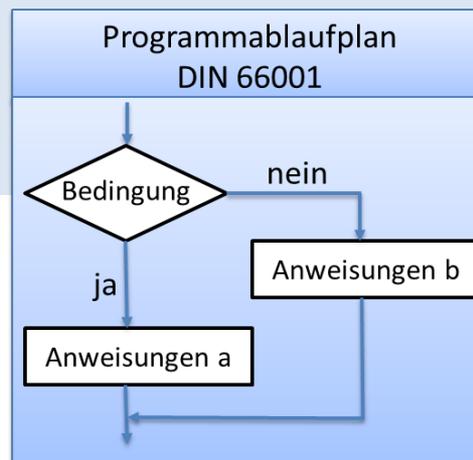
```
a == b || b != c && c < 10; //Ist c ungleich b UND kleiner 10 ODER a gleich b
```

```
(a == b || b != c) && c < 10; // Ist a gleich b ODER b ungleich c UND c  
kleiner 10
```

Die `if`-Anweisung entscheidet anhand einer Bedingung, welche Anweisungen ausgeführt werden.

```
// Falls Bedingung zutrifft, führe  
Anweisungen A1, A2 aus, sonst führe  
Anweisungen B1, B2 aus!
```

```
if(Bedingung){  
  
    anweisung_A1;  
    anweisung_A2;  
  
} else{  
    anweisung_B1;  
    anweisung_B2;  
}
```



```
// Konkretes Beispiel  
// Entscheidet, ob  $i < 20$  oder größer  
int i = 10;  
  
if(i < 20){  
  
    System.out.println(„i ist kleiner  
20“);  
  
} else{  
    System.out.println(„i ist  
groesser gleich 20“);  
}
```

**Zweiseitige
Auswahlstruktur**

Es gibt auch einfache `if`-Anweisungen (ohne `else`-Block)

- Falls die Bedingung `true` ist, wird der `if`-Block ausgeführt, sonst wird dieser einfach übersprungen

```
if (Bedingung){  
  
    anweisung_A1;  
    anweisung_A2;  
  
}  
  
anweisung_B1;  
anweisung_B2;  
//...
```

```
// Konkretes Beispiel  
// Falls i > 20, belege Variable s  
  
int i = 10;  
String s = „Ist kleiner gleich 20“;  
  
if(i>20){  
    s = „Ist groesser 20“;  
}  
  
System.out.println(s);
```

`if`-Anweisungen können auch verschachtelt werden

- Falls es mehr als eine Alternative (`else`) geben soll, aber nur **eine** ausgewählt werden darf
- Verwendung von `else if`-Anweisung
- Auch weitere `if`-Anweisungen im `else`-Block möglich

```
if (Bedingung 1){  
    Anweisungen A;  
} else if (Bedingung 2){  
    Anweisungen B;  
} else if (Bedingung XY){  
    Anweisungen XYZ;  
} else {  
    sonstige Anweisungen  
}
```

```
// Konkretes Beispiel  
// Zahl prüfen  
  
//int zahl ist definiert  
if (zahl == 0){  
    System.out.println(„Zahl ist 0“);  
} else if (zahl > 0){  
    Sytem.out.println(„Zahl ist  
positiv“);  
} else{  
    System.out.println(„Zahl ist  
negativ“)  
}
```

Ein paar Regeln:

- Es kann beliebig viele `else-if` Anweisungen geben
- Der letzte `else`-Block ist wieder optional und bezieht sich auf die letzte `if`-Bedingung

Unterschied zwischen `if` und `else if` Anweisungen:

- Alle `if`-Blöcke werden ausgeführt sobald die entsprechende Bedingung erfüllt ist, also `true` ergibt
- Nur ein `else if` Block wird ausgeführt, wenn die entsprechende Bedingung `true` ergibt
- Mit `else if` Anweisungen können wir also gegenseitige Ausschlusskriterien definieren

Erstellen Sie ein neues Projekt in IntelliJ mit Namen Uebung05

Aufgabe 1:

- Erstellen Sie in diesem Projekt ein Programm (Main-Methode) mit dem Namen *Zahlentest*
- Der Nutzer soll dabei 2 Zahlen vom Typ integer eingeben.
- Zur Erinnerung:
 - `Scanner scan = new Scanner(System.in);`
 - `Scan.nextInt();`
- Nun sollen folgende Fälle geprüft werden:
- Sind beide Zahlen gleich, dann wird auf der Konsole ausgegeben, dass die Zahlen gleich sind
- Ist eine Zahl größer als die andere, dann soll die größere der beiden Zahlen mit einem entsprechenden Hinweis (bspw.: „Die größere Zahl lautet:“) auf der Konsole ausgegeben werden.