LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Javakurs für Anfänger

Einheit 06: Einführung in Kontrollstrukturen

Lorenz Schauer
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

1. Teil: Einführung in Kontrollstrukturen

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

2. Teil: Weitere Kontrollstrukturen

- Auswahlstrukturen
  - `if`-Anweisungen, `switch-case`

Praxis:

- Übungen zu Kontrollstrukturen
  - Übungen zu `if`
  - Übungen zu `switch-case`

Lernziele

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

Steuerung des Programmablaufs: Einführung in Kontrollstrukturen

Bisher: Einfacher sequentieller Ablauf von Befehlen

- Beispiel Student (letzte Stunde):

```
public class Hauptprogramm {

   //Main-Methode
   public static void main(String[] args){

      Student studi1 = new Student("Peter",3);
      Student studi2 = new Student("Hanna",5);

      studi1.lernen();
      studi2.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 ….

Überblick über Kontrollstrukturen

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

Anweisungsfolge (Sequenz)

Wie bei unseren bisherigen Programmen:

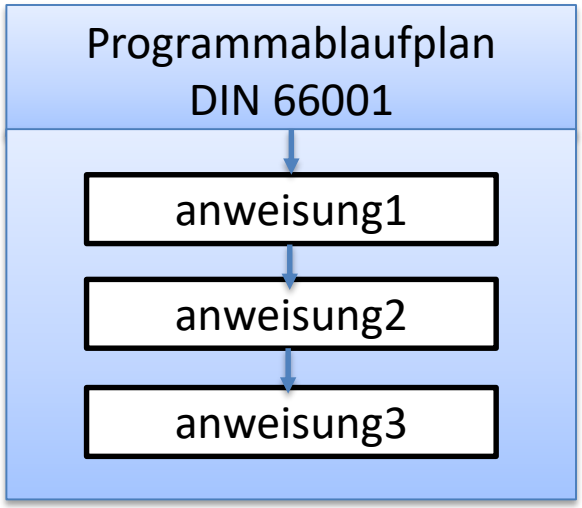
- Die im Quellcode vorhandenen Anweisungen werden sequentiell von oben nach unten abgearbeitet.
- Die Sequenz ist damit die einfachste Kontrollstruktur
- Beispiel:

```
anweisung1;

anweisung2;

anweisung3;
```

Darstellung als Programmablaufplan



Blöcke

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

Sichtbarkeit von Variablen

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

      int y = 5; //Int-Variable y wird definiert. Ist nur im Block sichtbar!
      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

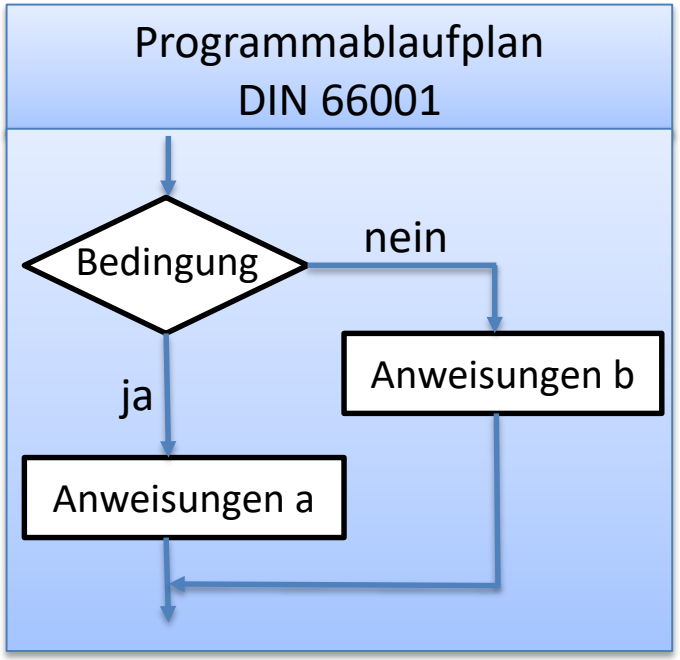
y

z

Auswahlstruktur bzw. Selektion

Auswahlstrukturen ermöglichen die bedingte Ausführung von Anweisungen

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

Was ist eine Bedingung?

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 (vollständig)	7
^	XOR	8
|	ODER (vollständig)	9
&&	UND	10
||	ODER	11

Logische Operatoren

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

Beispiele


```
// 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

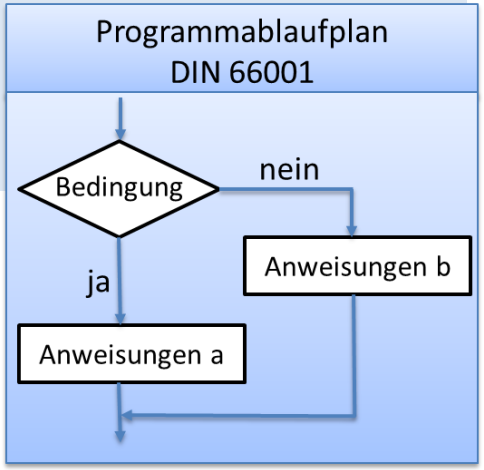
Die `if`-Anweisung entscheidet anhand einer Bedingung, welche Anweisungen ausgeführt werden.

```
// 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;
}
```



Zweiseitige Auswahlstruktur

```
// 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“);
}
```

Einfache `if`-Anweisungen

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);
```

Verschachtelte `if`-Anweisungen

`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“)

}
```

Verschachtelte `if`-Anweisungen

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 entsprechend 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

Übungen zu `if`-Anweisungen

Erstellen Sie ein neues Projekt in Eclipse mit Namen Uebung06

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
- 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.

Übungen zu `if`-Anweisungen

Aufgabe 2:

- Erstellen Sie in Ihrem Projekt *Uebung06* ein Programm *QuadratischeGleichungen,* das folgende Aufgabe übernimmt:
  - Das Programm soll zurückgeben, ob eine quadratische Gleichung der Form:
    $$ax^2 + bx + c = 0$$
    2, eine oder keine Lösung besitzt.
  - Die Antwort darauf liefert die Diskriminante D, die über a,b und c folgendermaßen berechnet werden kann:
    $$D = b * b – 4 * a * c$$
- Lassen Sie den Benutzer also 3 Werte für a,b und c eingeben (Typ integer)
- Berechnen Sie die Diskriminante
- Geben Sie auf der Konsole aus, wie viele Lösung die Gleichung dann hat
  - D = 0: eine Lösung
  - D > 0: 2 Lösungen
  - D < 0: keine Lösung

Die `switch-case`-Anweisung

Einfache verschachtelte `if`-Anweisungen können auch relativ kompakt und elegant gelöst werden mittels `switch-case` Konstrukt.

- Bietet eine mehrseitige Auswahlstruktur an

```
// Allgemeine Syntax von switch-case

switch(Ausdruck){

   case Konstante1:
      anweisungen_im_Fall_1;
      break;

   case Konstante2:
      anweisungen_im_Fall_2;
      break;

   case KonstanteXY:
      anweisungen_im_Fall_XY;
      break;

   default:
      anweisungen_im_Default_Fall;
      // wenn sonst kein Fall zutrifft
      break;


}
```

```
//Konkretes Beispiel
// int Zahl ist definiert

String out = „“;
switch(zahl){

   case 0:
      out = „Zahl ist 0“;
      break;

   case 5:
      out =„Zahl ist 5“;
      break;

   case 10:
      out = „Zahl ist 10“;
      break;

   default:
      out =„Zahl ist nicht 0,5, oder 10“;
      //-> break kann hier wegfallen

}
```

Die switch-case-Anweisung: Ein paar Regeln sind zu beachten


```
// Allgemeine Syntax von switch-case

switch(Ausdruck){

   case Konstante1:
      anweisungen_im_Fall_1;
      break;

   case Konstante2:
      anweisungen_im_Fall_2;
      break;

   case KonstanteXY:
      anweisungen_im_Fall_XY;
      break;

   default: {
      anweisungen_im_Default_Fall;
      // wenn sonst kein Fall zutrifft
      break;
   }

}
```

Ausdruck:
Der Ausdruck muss vom Typ: `char`, `byte`, `short`, oder `int` sein.
Seit Java 7 auch vom Typ `String`

Konstanten:
Müssen alle einen unterschiedlichen Wert haben, die dem Typ des Ausdrucks entsprechen!

Blöcke (geschweifte Klammern):
Sind nicht erforderlich

Default:
Anweisungen werden ausgeführt, wenn sonst kein `case`-Wert mit dem Wert des Ausdrucks übereinstimmt.

- Ist optional
- Sollte am Ende definiert werden

break:
Anweisungen werden im entspr. `case` so lange ausgeführt, bis eine `break`-Anweisung folgt, welche zum verlassen des `switch-case`-blocks führt

Übung zu `switch-case` Anweisungen

Aufgabe 3:

Erstellen Sie in Ihrem Projekt *Uebung04* ein Programm *Zahlenraten,* das folgende Aufgabe übernimmt:

- Das Programm soll den Anwender wieder auffordern eine Zahl zwischen 0 und 10 einzugeben
- Auf die eingegebene Zahl soll mittels einer `switch-case` Anweisung reagiert werden:
  - Bei Eingabe einer 5 soll der Text ausgegeben werden „Super! Treffer“
  - Bei Eingabe einer 3, 4, 6, oder 7 soll ausgegeben werden „Schade, knapp daneben“
  - Bei allen anderen Eingabe soll ausgegeben werden: „Daneben!“

Zusammenfassung

Was haben wir gelernt?

- Motivation für Kontrollstrukturen
- Blöcke und Sichtbarkeit von Variablen
- Die Auswahlstrukturen `if` und `switch`

Nächste Woche:

- Geht es weiter mit Kontrollstrukturen…
  - Wiederholungsstrukturen: `while`, `for`, …