Programmierung 2

Kapitel 3 – Kontrollstrukturen

Vielen Dank an Prof. Erik Buchmann
für die beigesteuerten Folien

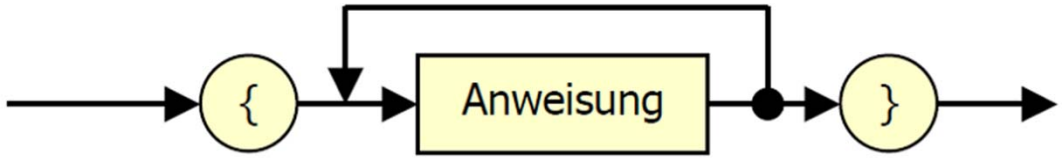
Kontrollstrukturen

Kontrolstrukturen

- Programmablauf oft abhängig von Bedingungen, z.B. Variablen
  - Zwei Varianten: Wiederholungsanweisung, Bedingte Anweisung

- Einfache Beispiele
  - Berechne für jede Zahl von 1 bis 10 die Quadratzahl
  - Solange wie der Nutzer nicht q eingibt, frage nach neuem Datensatz
  - Falls Geld auf dem Konto, führe Abbuchung durch, ansonsten Fehlermeldung
  - Umwandlung römischer Zahl in Dezimal mit Fallunterscheidung (V=5, X=10, etc.)

Verbund-Anweisung/Block-Anweisung

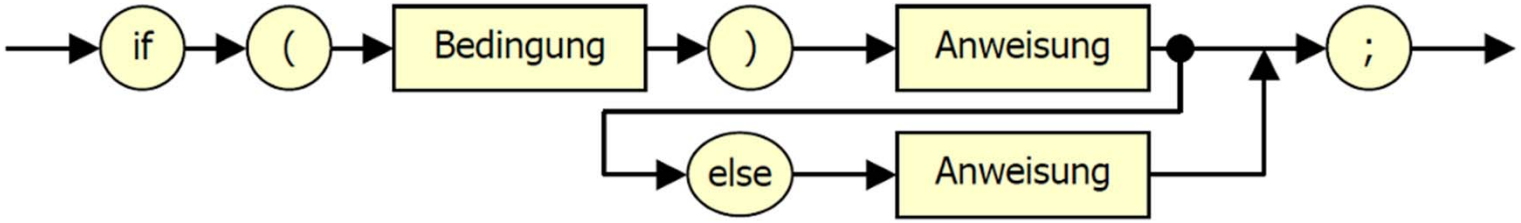


- Zusammenfassung mehrerer Anweisungen zu einer Anweisung, z.B. für Kontrollstrukturen (IF, WHILE, etc.)
- innerhalb des Blocks deklarierte Variablen sind nur innerhalb des Blocks gültig

Darf ich das? Und was ist die Ausgabe?

```
int i = 10;
{
    int i = 100;
    i = i * 10;
}
System.out.println(i);
```

IF Anweisung (Bedingte Anweisung)



- Bedingung: boolscher Ausdruck
- else-Zweig ist optional

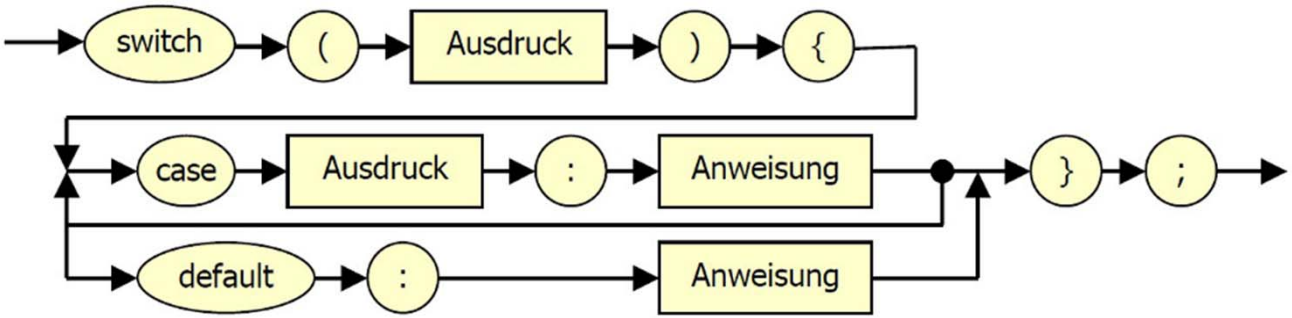
- Ablauf
  - Werte die den booleschen Ausdruck (Bedingung) aus.
  - Falls der Ausdruck den Wert true liefert, führe Anweisung aus
  - Andernfalls führe die else-Anweisung aus, falls vorhanden
- Beliebige Verschachtelung möglich
  - In if und else-Zweig können auch if-Anweisungen stehen

Beispiel

- Bestimme ob eine Zahl gerade ist

```
int zahl = ...;
if (zahl % 2 == 0)  {
    System.out.println(zahl+" ist gerade.");
} else {
    System.out.println(zahl+" ist ungerade.");
}

if (zahl % 3 == 0)  {
    System.out.println(zahl+" ist durch drei teilbar.");
}
```

Switch Anweisung (Mehrfachauswahl)



- case-Ausdrücke: Typ char, byte, short, int oder String
  - keine Dopplungen bei case-Ausdrücken!

- Ablauf
  - werte Ausdruck aus
  - falls case-Ausdruck mit dem berechneten Wert existiert:
    - fahre an der entsprechenden Stelle mit der Ausführung fort
  - falls kein passender case-Ausdruck existiert:
    - wenn default vorhanden, fahre beim default fort, sonst Ende

Beispiel

Wenn i=1wird auch Anweisungs-block 2 bearbeitet

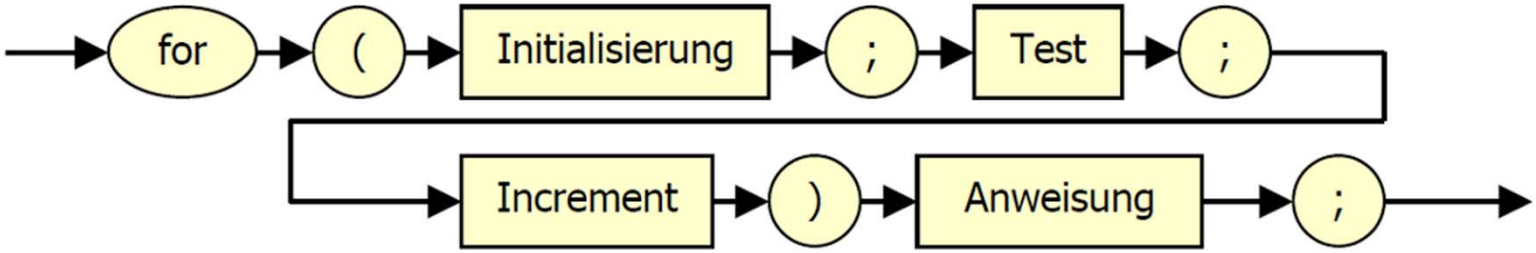
```
int auswahl = ... // Nutzereingabe
switch (auswahl) {
        case 1: // Schnitzel
        System.out.println(„auswahl == 1");

        case 2:  // Burger
        System.out.println("auswahl == 1 ODER 2");
        break;
        case 3:  // Tofu
        System.out.println("auswahl == 3");
        break;
        default: // Suppe
        System.out.println("auswahl != (1/2/3) ");
}
```

- break: breche weitere Überprüfungen ab
- *Achtung, nicht das break vergessen, da switch sonst mit dem nächsten Anweisungsblock fortfährt!*

Wiederholungsanweisungen

- Wiederholte Ausführung der selben Anweisung / des selben Anweisungsblocks
  - in der Regel veränderte Variablenwerte pro Iteration
- Steuerung der Wiederholung durch
  - Zählvariable: for
  - Bedingung: while, do
- Zusätzliche Steuerungsmöglichkeiten durch
  - (vorzeitiges) Beenden der Iteration: continue
  - (vorzeitigen) Abbruch der Wiederholungsanweisung: break

FOR Schleife



- Ablauf
  - führe Initialisierung-Anweisung aus
  - evaluiere den booleschen Ausdruck (Test)
  - falls Ausdruck true liefert:
    - führe Anweisung aus
    - berechne den Increment-Ausdruck
    - gehe zurück zur Evaluierung des Ausdrucks
  - falls Ausdruck false liefert, beende die for-Anweisung

Beispiele

- Ausgabe der Zahlen 0 bis 9

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

- Berechne die Summe der ersten n Zahlen

```
int summe = 0;
int n = 100;
for (int i=1; i <= n; i++) {
    summe += i;
}
System.out.println("Summe 1 bis " + n + " = " + summe);
```

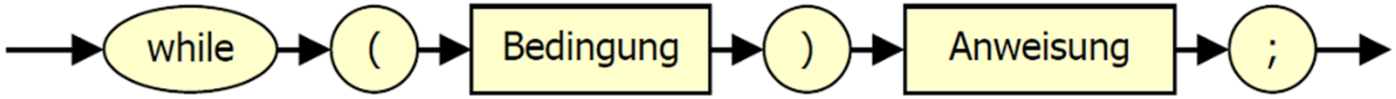
Ist die Variable i außerhalb der FOR-Schleife gültig? Wenn nein, was muss ich ändern damit das so ist?

FOR Schleife und Arrays

- FOR (variable: array) {...}
  - Iteration über ein Array
  - Variable nimmt Typ und Wert des Array-Elements an

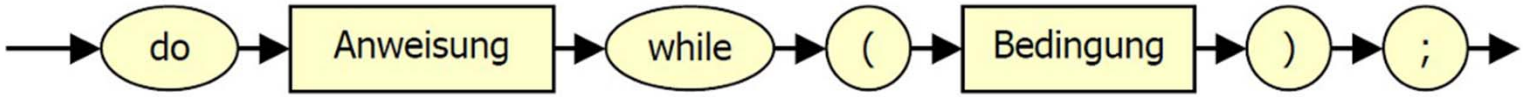
```
int sum = 0;
int[] zahlen = {1,2,3,4,5};

for (int i = 0; i < zahlen.length; i++) {
  sum += zahlen[i];
}

aquivalent zu

for (int zahl : zahlen) {
  sum += zahl;
}
```

WHILE Anweisung (Bedingung am Anfang)



- Ablauf
  - Werte den booleschen Ausdruck (Bedingung) aus
    - Falls die Bedingung den Wert false liefert:
      - beende die while-Anweisung
    - Falls die Bedingung den Wert true liefert
      - führe die Anweisung aus
      - fahre bei der Auswertung der Bedingung mit der Schleife fort

DO Anweisung (Bedingung am Ende)



- Ablauf
  - Führe die Anweisung aus
  - Werte den booleschen Ausdruck (Bedingung) aus
    - Falls die Bedingung den Wert false liefert:
      - beende die while-Anweisung
    - Falls die Bedingung den Wert true liefert
      - fahre bei der Ausführung der Anweisung mit der Schleife fort

Semantische Äquivalenz

- do <anweisung> while (<bedingung>)

  ist identisch zu

- <anweisung> while (<bedingung>) <anweisung>

```
int anzahl = 3;

do {
    System.out.println(anzahl);
    anzahl++;
} while (anzahl <= 10);
```

```
int anzahl = 3;
System.out.println(anzahl);
anzahl++;

while (anzahl <= 10) {
    System.out.println(anzahl);
    anzahl++;
};
```

Semantische Äquivalenz

- for (<init-anweisung>; <incr-anweisung>; <bedingung>) <anweisung>
  ist identisch zu
- { <init-anweisung> while (<bedingung>) {<anweisung>; <incr-anweisung>} }

- Vervollständigen Sie das Beispiel

```
for (int i=3; i<10; i++) {
    System.out.println(i);
}
```

while…?

Continue Anweisung

- Semantik
  - Springt an das Ende der innersten Schleifenanweisung einer for, while oder do-Schleife

- Beispiel, was wird ausgegeben?

```
int x = 0;
while (x < 100) {
    x++;
    if (x != 3) {
       continue;
    }
    System.out.println("x == 3");
}
```

Break Anweisung

- Semantik
  - vorzeitiges Verlassen eines Blockes, d.h., (innerste) do, while, for–Schleife wird verlassen
  - Verwendung auch bei switch
- Beispiel

```
int ergebnis = 0;
for (int zahl=1; zahl<10;  zahl++) {
    ergebnis = ergebnis + zahl;
    if (ergebnis > 6) break;
}
System.out.println(ergebnis);
```

Was gibt dieses Beispiel aus?

Methoden

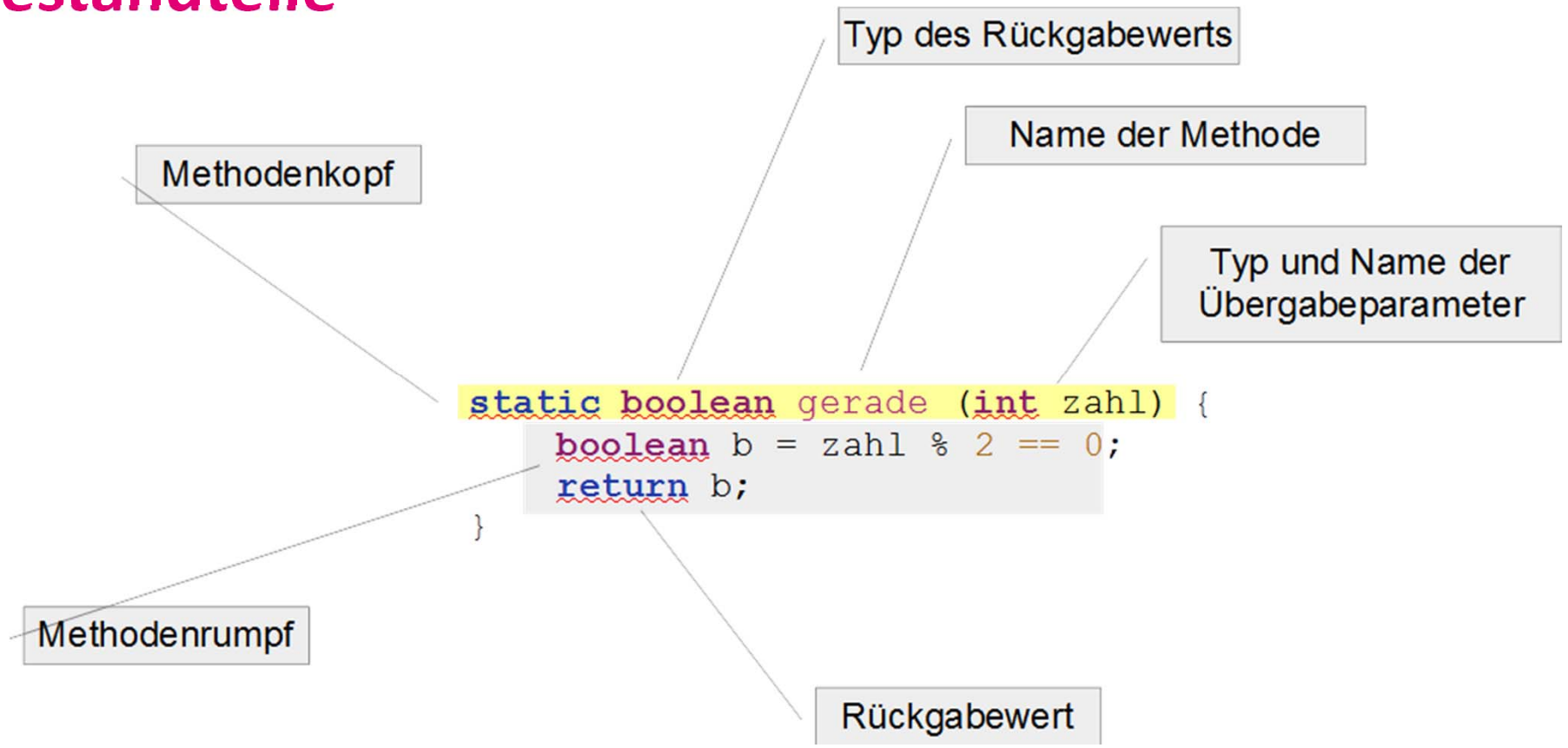
Wozu Methoden?

- Erweiterung des imperativen Programmierparadigmas
  - Zerlegen eines Programms (Algorithmus) in überschaubare Teile(„Teile und Herrsche“)

- Merkmale
  - Übergabe von Parametern (Eingabe, optional)
  - Rückgabe von Ergebnissen (Ausgabe, optional)

- Im Folgenden:
  - Aufbau einer Methode
  - Gültigkeitsbereich von Variablen
  - Rekursion, d.h. selbstaufrufende Methoden

Bestandteile



*Das **static** nehmen Sie bitte für den Moment so hin.*

Beispiele

```
static int fakultaet (int n) {
    int ergebnis = 1;
    for (int i=1; i<=n; i++) {
        ergebnis *= i; // ergebnis = ergebnis * i
    }
    return ergebnis;
}

static boolean gerade (int zahl) {
    if ((zahl % 2)==0) {
        return true;
    } else {
        return false;
    }
}
```

Rückgabewerte

- Rückgabewert im Methodenkopf kann sein
  - primitiver Datentyp
  - Referenzdatentyp
  - void (keine Rückgabeparameter)
- **Return**
  - Unmittelbares Verlassen der Methodenausführung
  - Wert beim Return wird an das aufrufende Programm geliefert
  - Wert muss den spezifizierten Rückgabetyp haben

```
static void eingabe() {
    … return;
}
```

```
static int eingabe() {
    … return 7;
}
```

Code Review

- *Finden Sie drei Fehler?*

```
static int rechnung () {
    double ergebnis = 1.0;
    for (int i=1; i<=9; i++) {
        ergebnis = ergebnis * i
    }
    return ergebnis;
    ergebnis = ergebnis + 1;
}

static boolean gerade () {
    int zahl = 7;
    if ((zahl % 2)==0) {
        return true;
    }
}
```

Methodenaufruf

- Ganz einfach über den Namen, Rückgabewert über Zuweisung

```
public static void main(String[] args) {
    int f = fakultaet();
    System.out.println("Fakultät von 9 ist " + f);
}

static int fakultaet () {
    int ergebnis = 1;
    for (int i=1; i<=9; i++) {
        ergebnis *= i;
    }
    return ergebnis;
}
```

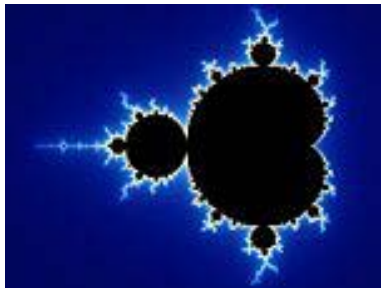
- Variablen innerhalb einer Methode sind auch nur dort gültig
  - Kein Zugriff innerhalb der „main“-Methode auf Variable „ergebnis“→ *denken Sie an die Abweichungen bei Referenztypen*

Rekursionen

Rekursionen

- Definition eines Problems, einer Funktion oder eines Verfahrens durch sich selbst

- Beispiele
  - Berechnung der Fakultät $n! = n * (n-1)!$ mit $n > 0$
  - Mandelbrot-Mengen bzw. Apfelmännchen $z_{n+1} = f_c(z_n)$ *(Anm.: Formel im Raum der komplexen Zahlen)*
  - *Traversieren eines Baumes*



Rekursionen Definition

- Definition **Rekursion**: Eine Funktion (i. Allg. Methode) heißt rekursiv, wenn sie während ihrer Abarbeitung erneut aufgerufen wird.

direkte Rekursion:
wenn erneuter Aufruf im Funktionsrumpf selbst

```
static int f (int x) {
     …   f(a) …
}
```

indirekte Rekursion:
wenn erneuter Aufruf in einer anderen Funktion steckt

```
static int f (int x) {… g(a)… }
static int g (int y) {… f(b)… }
```

- Rekursionstiefe: Anzahl der aktuellen Aufrufe einer Funktion minus 1
- Ein Algorithmus heißt
  - iterativ, wenn er Wiederholungsanweisungen (for, while, ...) verwendet
  - rekursiv, wenn er rekursive Funktionen verwendet

Beispiel 1

- Zwei äquivalente Algorithmen

Ist hier die Rekursion eine gute Idee?

Begründung?

```java
/**
 * Algorithmus mit FOR-Schleife
 */
static void loop(int i) {
    for (int z = i; z > 0; z--) {
        System.out.println(z);
    }
}
```

```java
/**
 * rekursiver Algorithmus
 */
static void rec(int i) {
    if (i == 0) return;
    System.out.println(i);
    rec(i - 1);
}
```

Beispiel 2

- Berechnung der Fakultät

  n! = n * (n-1)

  Gleiche Frage

```
static int fakultaetRekursiv (int zahl) {
    if (zahl==0) return 1;
    return zahl*fakultaetRekursiv(zahl-1);

}
```

```
static int fakultaetIterativ (int zahl) {
    int ergebnis = 1;
    for (int i=1; i<=zahl; i++) {
        ergebnis *= i;
    }
    return ergebnis;
}
```

Überlegungen zu Rekursionen

- Zu jedem rekursiv formulierten Algorithmus existiert ein äquivalenter iterativen Algorithmus.
- Vorteile rekursiver Algorithmen:
  - kürzere Formulierung
  - leichter verständliche Lösung
  - Einsparung von Zählvariablen
  - teilweise sehr effiziente Problemlösungen
- Nachteile rekursiver Algorithmen
  - Overhead beim Funktionsaufruf, Speicher auf dem Stack
  - Verständnisprobleme bei Programmieranfängern
  - Konstruktion rekursiver Algorithmen "gewöhnungsbedürftig"

Aufgabe

- Potenzieren: $x^y = 1\underbrace{*x*\ldots*x}_{\text{y mal}}$
- Beispiele:
  - $2^0 = 1$
  - $2^3 = 8$

```
public static int potenzieren(int x, int y)
{
    int potenz = 1;
    for(int i = y; i>0; i--) {
        potenz = potenz * x;
    }
    return potenz;
}
```

Wie kann man die Potenzierung rekursiv implementieren?

Zusammenfassung

- Kontrollstrukturen
  - Bedingte Anweisungen (if, switch)
  - Wiederholungsanweisungen, Schleifen (while, do, for)
- Methoden
  - Strukturierung von Anweisungsblöcken für in sich abgeschlossene Teilaufgaben
- Rekursion
  - Wiederholter Aufruf einer Methode durch sich selbst

Kontrollfragen

- Erläutern Sie die Funktionsweise der Kontrollstrukturen in Java.
- In welcher Reihenfolge werden Testausdruck, (ggf. Inkrement) und Anweisungsblock bei for, while und do-while Schleifen evaluiert?
- Welche Auswirkungen haben break und continue bei den verschiedenen Kontrollstrukturen?
- Wie ist eine Methode aufgebaut?
- Geben Sie ein eigenes Beispiel für eine Rekursion an.