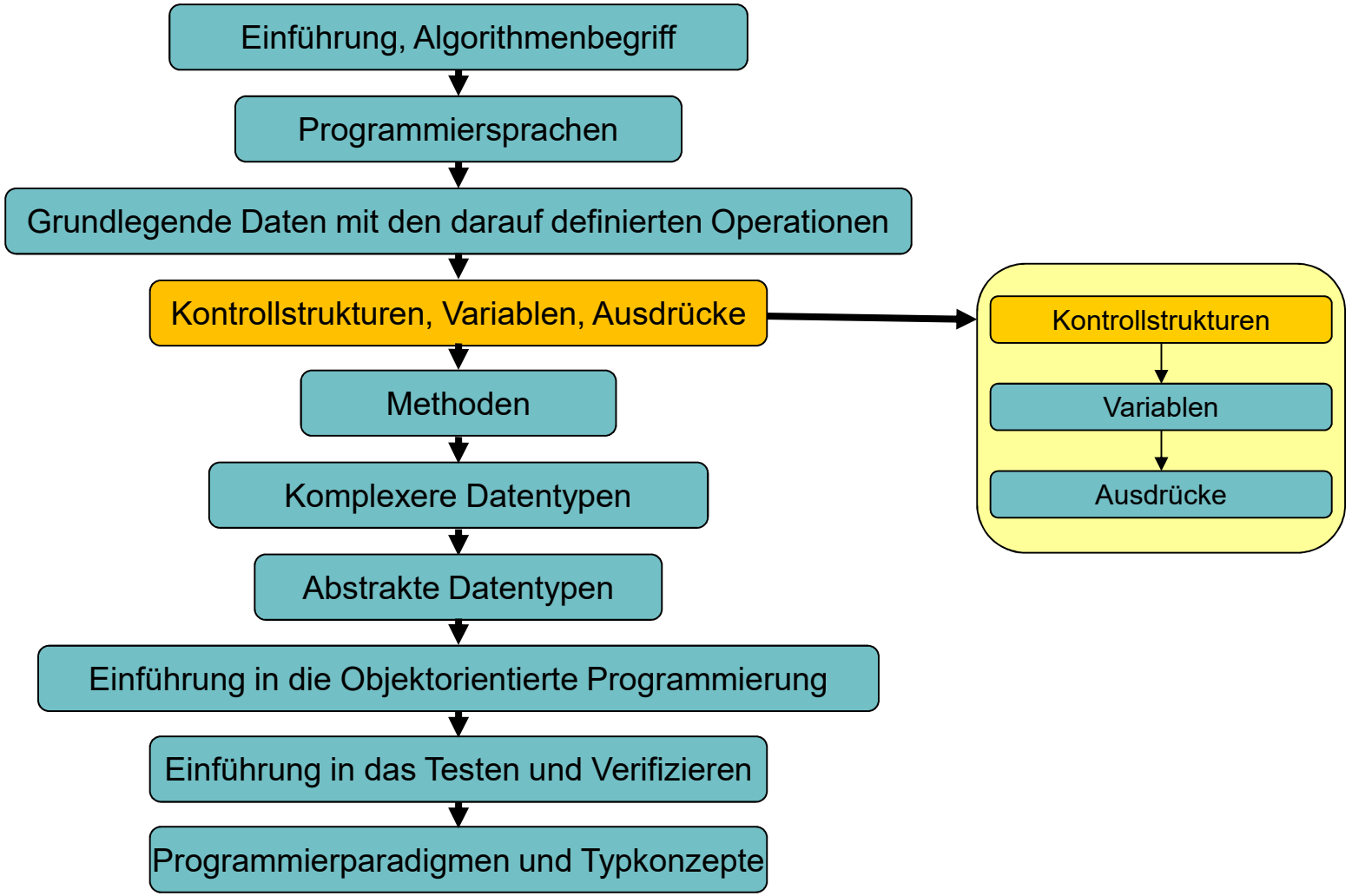
# Inhalt dieser Veranstaltung

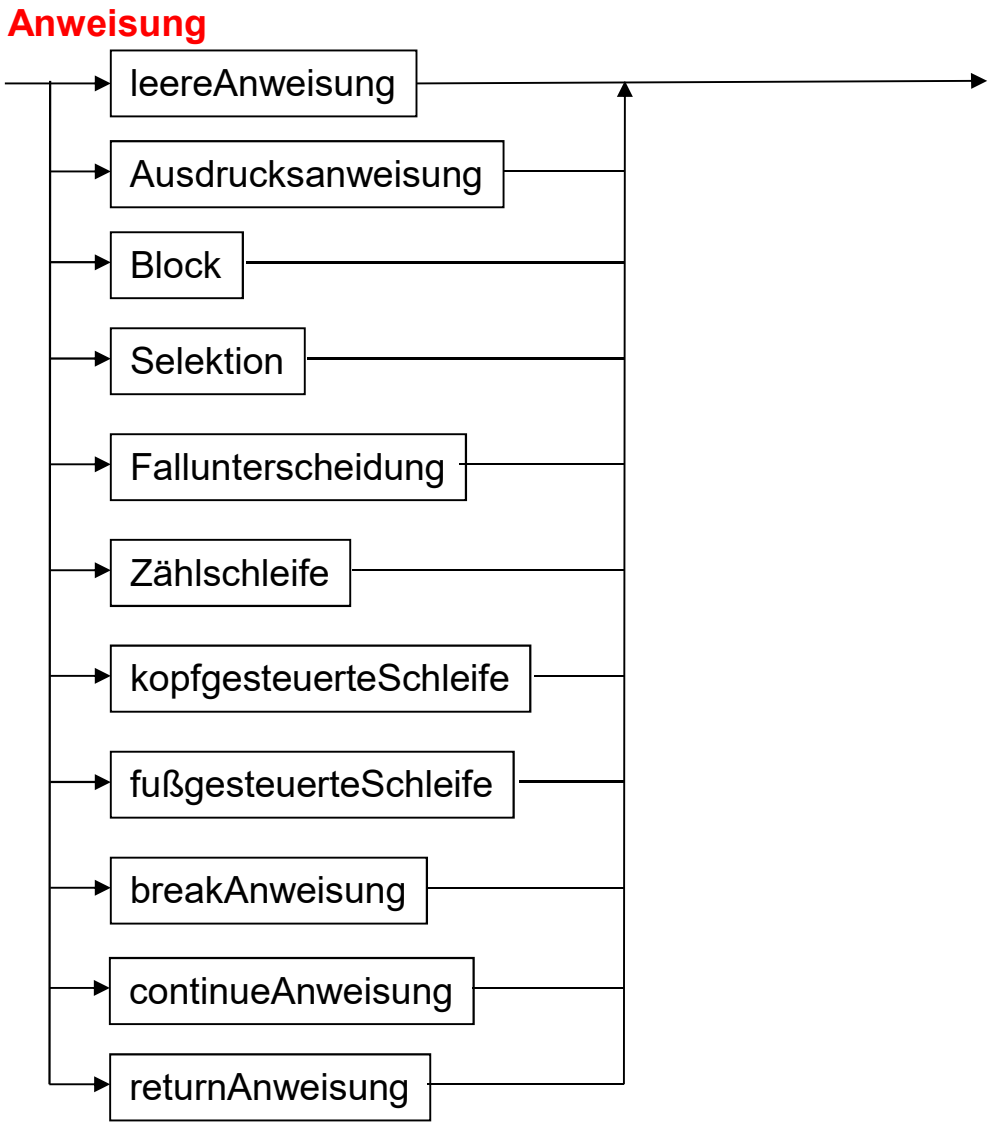


# Kontrollstrukturen

- Bis jetzt: Daten mit den darauf definierten Operationen führte uns zum Begriff des Datentyps
- Jetzt: wie kann der Ablauf in einem Programm gesteuert werden?
- Antwort: Kontrollstrukturen
- Hierbei der zentrale Begriff in Programmiersprachen: Anweisung

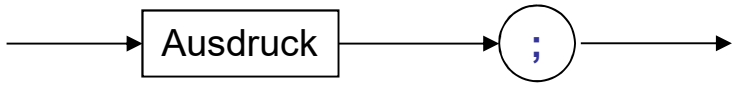
- Wiederholung: Muster
  - Einzelanweisung $A$
  - Sequenz $A_1;\ldots;A_n$
  - Selektion **if** B **then** $A_1$ **else** $A_2$ **endif**
  - Mehrfachselektion **switch**(x) **case** $x_1:A_1;$ **case** $x_2:A_2;$ … **case** $x_n:A_n;$ **endswitch**
  - Iteration **while** B **do** A **endwhile**

# Übersicht Anweisungen



# Ausdrucksanweisung

**Ausdrucksanweisung**



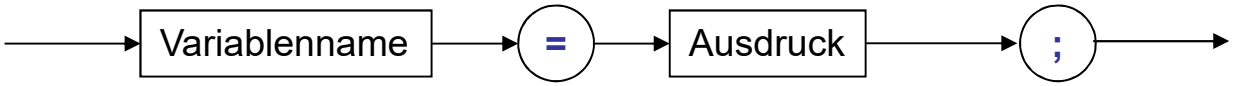

- Aus einem beliebigen Ausdruck wird syntaktisch eine Anweisung, indem ein Semikolon an den Ausdruck angehängt wird
- Bedeutung: werte den Ausdruck aus
- Dies ist nur sinnvoll, wenn in der Auswertung des Ausdrucks etwas Nachhaltiges passiert
- Nachhaltig kann zum Beispiel sein:
  - In einer Zuweisung wird einer Variablen ein neuer Wert zugewiesen (gleich mehr)
  - Es wird eine Methode aufgerufen, in deren Auswertung etwas Nachhaltiges passiert (Beispiel: `System.out.println(...);` )

- Gegenbeispiel (syntaktisch erlaubt, macht aber wenig Sinn):

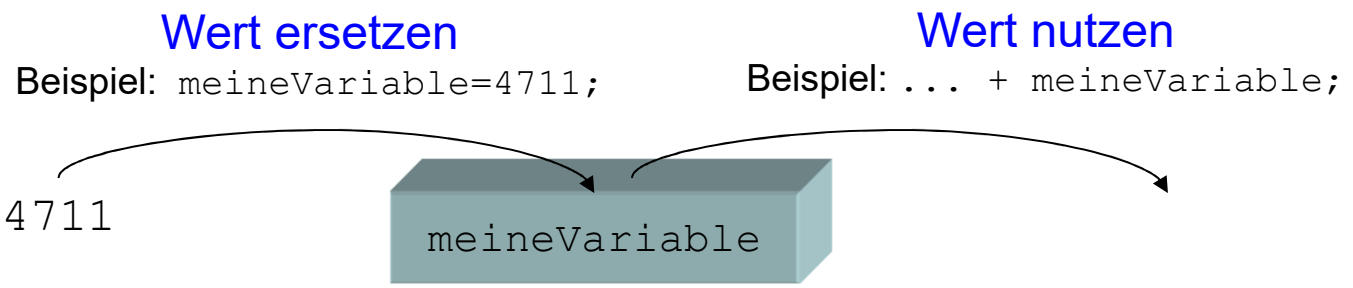
```
3 + 4;
```

# Häufige Anwendung: Zuweisung

**Zuweisung**



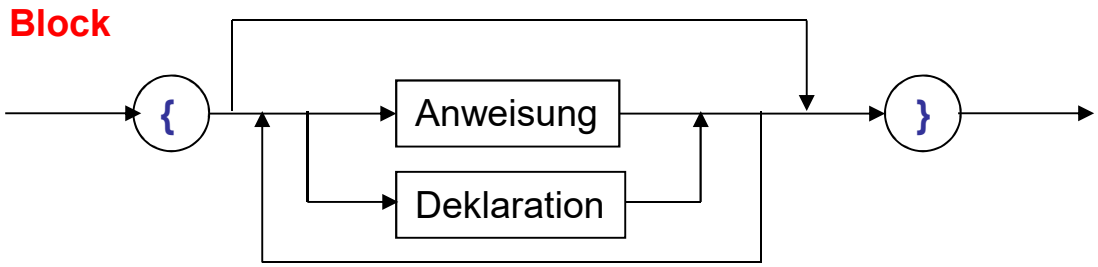

- Bedeutung:
  - Der Ausdruck auf der rechten Seite wird ausgewertet. Das Ergebnis ist ein Wert eines bestimmten Typs (ergibt sich aus dem Ausdruck).
  - Der Typ dieses Wertes muss mit dem Typ der Variablen übereinstimmen (bzw. kompatibel sein; später mehr)
  - Dann wird dieser Wert in dieser Variablen gespeichert. Der alte Inhalt der Variablen ist damit verloren.
- Erlaubt: in dem Ausdruck der rechten Seite kommt der Variablenname der linken Seite vor. Dies ist erlaubt und macht Sinn.
- Zur Erinnerung:



# Beispiel

```
/**
 * Dieses Programm wandelt eine Dollar-Angabe in Euro um
 */
public class DollarNachEuro {
    public static void main(String[] args) {

        double dollarBetrag;     // Dollar-Betrag
        double umrechnungskurs;  // Umrechnungskurs
        double euroBetrag;       // Ergebnis in Euro

        // Beispielwerte
        dollarBetrag = 37.48;
        umrechnungskurs = 1.48;

        // berechne den Euro-Betrag
        euroBetrag = dollarBetrag * umrechnungskurs;

        // gebe das Ergebnis auf dem Bildschirm aus
        System.out.println("Der Euro-Betrag ist " + euroBetrag);
    }
}
```

# Block




- An manchen Stellen der Sprachgrammatik ist genau eine Anweisung verlangt / erlaubt
- Möchte man an diesen Stellen mehrere Einzelanweisungen notieren, so muss man diese zu einem Block zusammenfassen, der syntaktisch eine Anweisung ist
- In einem Block können beliebig viele Anweisungen und / oder Deklarationen (auch gemischt) stehen
- Block: Sequenz von Anweisungen mit der zusätzlichen Möglichkeit von Deklarationen
- Wir werden später sehen, dass ein Block auch Einfluss auf verschiedene Aspekte im Zusammenhang mit Variablen hat

# Beispiel

```
/**
 * Umwandlung Grad Fahrenheit nach Grad Celsius
 */
public class FahrenheitNachCelsius2 {
    public static void main(String[] args) {

        // Deklaration
        double fahrenheit, celsius;

        // Ausgangswert
        fahrenheit = 80.0;

        // Umrechnungformel anwendungen
        celsius = 5.0 * (fahrenheit - 32.0) / 9.0;

        // Ausgabe des Celsius-Wertes
        System.out.println("Fahrenheit: " + fahrenheit + ", Celsius: " + celsius);
    }
}
```

Blockmarkierung (Beginn / Ende)

# Zwischenstand

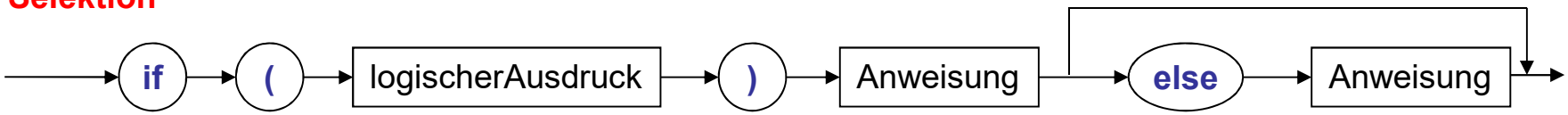
- Jeder Ausdruck gefolgt von einem Semikolon ist eine Anweisung
- Die Syntax verlangt an manchen Stellen genau eine Anweisung. In einem Block kann man mehrere Anweisungen zusammenfassen, die syntaktisch wie eine Anweisung wirken

**Reflektion**

- Kann man innerhalb eines Blocks einen weiteren Block verwenden? Wenn ja, wieso und wie? Wenn nein, wieso nicht?

# Selektion

**Selektion**




- Dient dazu, einen von zwei möglichen Wegen auszuwählen
- Der zweite Fall (`else`) kann auch fehlen (dann wird nur im ersten Fall etwas ausgeführt)

- Bedeutung:
  - Zuerst wird der logische Ausdruck ausgewertet
  - Fall der Wert `true` ist, so wird die Anweisung nach der schließenden Klammer ausgeführt. Damit ist die Anweisung beendet.
  - Ansonsten (Wert ist `false`) ist die Anweisung beendet, falls kein `else` vorkommt. Falls ein `else` vorkommt, so wird die Anweisung hinter dem `else` ausgeführt, womit die Selektion dann auch beendet ist.

# Beispiel 1

```
/**
 * Berechnung des Maximums zweier Zahlen
 */
public class MaxBerechnung {
    public static void main(String[] args) {

        // zwei Beispielwerte
        int x = 5;
        int y = 7;
        // in der Variablen max steht anschliessend das Resultat
        int max = 0;

        // welcher Wert ist der groessere?
        if(x > y)
            max = x;
        else
            max = y;

        // Ausgabe des gefundenen Wertes
        System.out.println("Der groessere der beiden Werte ist " + max);
    }
}
```

# Beispiel 2

```
/**
 * Berechnung des Maximums und Minimums zweier Zahlen
 */
public class MaxMinBerechnung {
    public static void main(String[] args) {
        int x = 5;    // zwei Beispielwerte
        int y = 7;
        int max = 0;  // in min und max stehen spaeter die Resultatwerte
        int min = 0;

        // welcher Wert ist der groessere?
        // Daraus ergibt sich auch, wer der kleinere ist
        if(x > y) {
            max = x;
            min = y;
        } else {
            max = y;
            min = x;
        }

        // Ausgabe der gefundenen Werte
        System.out.println("Der groessere der beiden Werte ist " + max);
        System.out.println("Der kleinere der beiden Werte ist " + min);
    }
}
```

# Problem (des Programmieres): kein Block

```
/**
 * Berechnung des Maximums und Minimums zweier Zahlen
 */
public class MaxMinBerechnung {
    public static void main(String[] args) {
        int x = 5;    // zwei Beispielwerte
        int y = 7;
        int max = 0;  // in min und max stehen spaeter die Resultatwerte
        int min = 0;

        // welcher Wert ist der groessere?
        // Daraus ergibt sich auch, wer der kleinere ist
        if(x > y)
            max = x;
            min = y;
        else
            max = y;
            min = x;

        // Ausgabe der gefundenen Werte
        System.out.println("Der groessere der beiden Werte ist " + max);
        System.out.println("Der kleinere der beiden Werte ist " + min);
    }
}
```

Fehler

# Problem durch Syntaxdefinition: "dangling else"

```
public class DanglingElse {
  public static void main(String[] args) {
    int x=1, y=2;

    if(x <= y)
      if(x == y)
        System.out.println("x==y");
      else
        System.out.println("x<y");
  }
}
```

bessere Lösung

```
public class DanglingElse2 {
  public static void main(String[] args) {
    int x=1, y=2;

    if(x <= y) {
      if(x == y) {
        System.out.println("x==y");
      } else {
        System.out.println("x<y");
      }
    }
  }
}
```

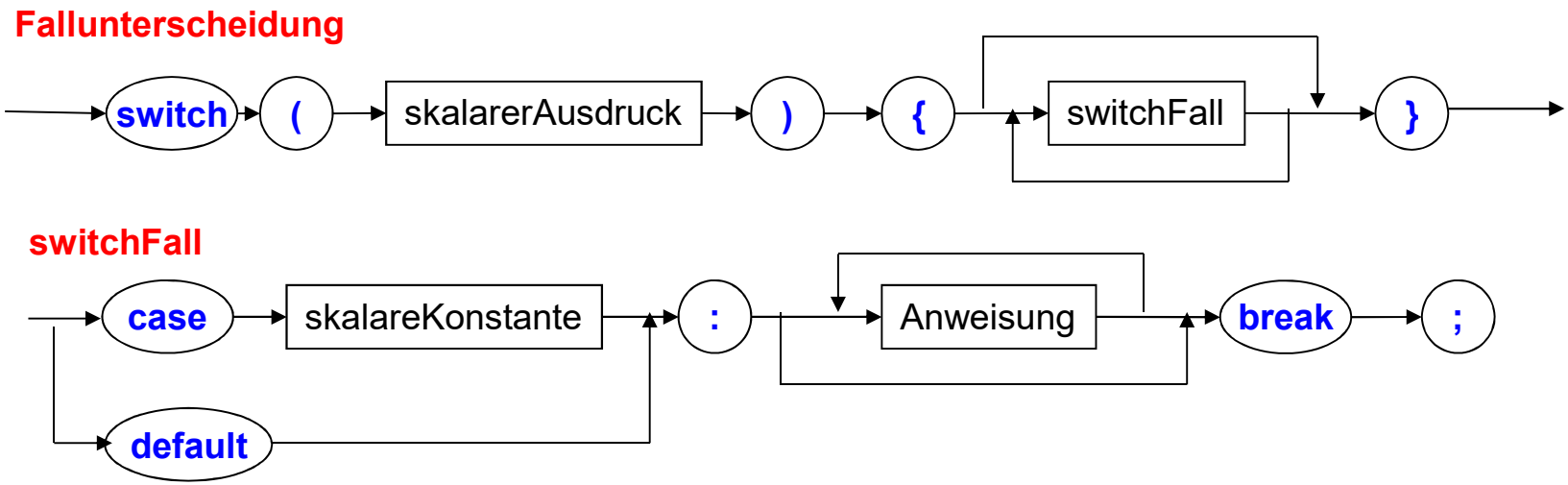
- Wozu gehört else-Fall mit `System.out.println("x<y")`? 1. oder 2. `if`?
- Laut Syntax ist beides möglich!

- Lösung 1 (in Sprache definiert; linke Seite): Bezug zum "letzten" `if`
- Lösung 2 (durch Programmierer; rechte Seite): durch Block-Bildung eindeutig

# Schaltjahr diesmal mit if-else

```
public class Schaltjahr2 {
    public static void main(String[] args) {
        int jahr = Integer.parseInt(args[0]);  // Beispieljahr
        boolean istSchaltjahr;

        // Ein Wert x ist durch y teilbar, wenn x % y == 0
        if ((jahr % 4) == 0) {
            if ((jahr % 100) == 0) {
                if ((jahr % 400) == 0) {
                    istSchaltjahr = true;
                } else {
                    istSchaltjahr = false;
                }
            } else {
                istSchaltjahr = true;
            }
        } else {
          istSchaltjahr = false;
        }
    }
}
```

# Fallunterscheidung




- Hier vereinfachte Syntax (Sprachsyntax erlaubt mehr)
- Bedeutung:
  - Der skalare Ausdruck wird ausgewertet (seit Java 7 auch Strings erlaubt)
  - Anschließend wird dieser Wert der Reihe nach mit den skalaren Konstanten der einzelnen Fälle verglichen und im Fall der Übereinstimmung werden die dazu gehörenden Anweisungen ausgeführt
  - Der default-Fall wird dann ausgeführt, wenn kein Fall vorher eingetreten ist

# Beispiel

```
public class Dezimalziffernumwandlung {
    public static void main(String[] args) {
        char ziffer = '8';    // Beispielziffer
        int wert = 0;         // hier drin wird der Wert stehen

        // Fallunterscheidung
        switch(ziffer) {
            case '0' : wert = 0;  break;
            case '1' : wert = 1;  break;
            case '2' : wert = 2;  break;
            case '3' : wert = 3;  break;
            case '4' : wert = 4;  break;
            case '5' : wert = 5;  break;
            case '6' : wert = 6;  break;
            case '7' : wert = 7;  break;
            case '8' : wert = 8;  break;
            case '9' : wert = 9;  break;
            default  : System.out.println("hier stimmt was nicht");
                       wert = -1;
                       break;
        }

      System.out.println("Der Wert der Ziffer " + ziffer + " ist " + wert);
    }
}
```

# Zwischenstand

- In einer Selektion kann man eine Anweisung in Abhängigkeit einer Bedingung ausführen
- Möchte man mehr als eine Anweisung dabei ausführen, muss man einen Block verwenden
- Mit Blöcken lässt sich auch das Problem des dangling else explizit lösen
- In Fallunterscheidungen (switch-Anweisung) lassen sich n Fälle unterscheiden inklusive einem default-Fall

**Reflektion**

- Wie kann man (mit den bis jetzt verfügbaren Mitteln) das Minimum und Maximum von drei Werten bestimmen?

# Iteration

- Wozu benötigt man das Konstrukt der Iteration?

- Aufgabe 1: berechne für eine Variable x den Wert $x^4$ nur mit Multiplikationen
- Lösung: `resultat = x * x * x * x;`

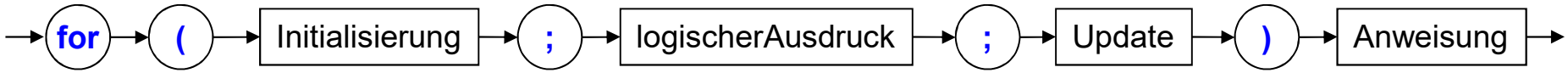
- Aufgabe 2: berechne für zwei Variablen x,y den Wert $x^y$
- Lösung?
- Das Problem ist hier, dass die Anzahl der durchzuführenden Multiplikationen zum Zeitpunkt der Programmierung und sogar des Programmstarts nicht feststeht bzw. feststehen muss

- Lösung: Konstrukt, das wiederholte Ausführung von Code zulässt
  $\rightarrow$ Iteration

- Mehrere Formen der Iteration möglich

# Zählschleife

**Zählschleife**




- Vereinfachte Syntax
- Initialisierung: Variablendeklaration mit Zuweisung. Beispiel: `int i=0`
- Update: Wertänderung dieser Variablen. Beispiel: `i=i+1`
- Den ersten Teil nennt man Schleifenkopf, die Anweisung Schleifenrumpf

- Bedeutung:
  1. Zuerst wird die Initialisierung ausgeführt
  2. Danach wird der logische Ausdruck ausgewertet. Falls der Wert `false` ist, bricht die Schleife ab (die Ausführung wird nach der Schleife fortgesetzt)
  3. Ansonsten (Wert des logischen Ausdrucks ist `true`) wird die Anweisung ausgeführt , anschließend Update ausgeführt und dann mit Schritt 2 fortgefahren.

# Beispiel 1

Zur Erinnerung: $x^y = \prod_{i=1}^{y} x$ Spezialfall $x^0=1$

```
/**
 * Potenzberechnung mit einer Zaehlschleife
 */
public class PotenzberechnungZaehlschleife {
    public static void main(String[] args) {

        // Beispielwerte
        int x = 5, y = 4;

        // Ergebnisvariable mit neutralem Element der Multiplikation
        int potenz = 1;

        // zaehle i von 1 bis y hoch
        for(int i = 1; i <= y; i = i + 1) {
            potenz = potenz * x;
        }

        System.out.println(x + " hoch " + y + "=" + potenz);
    }
}
```

# Weitere Beispiele

Spezialfall 0!=1

$$x! = \prod_{i=1}^{x} i$$

```
// neutrales Element
int fakultaet = 1;

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

Spezialfall $\sum_{i=1}^{0} i = 0$

$$summe(x) = \sum_{i=1}^{x} i$$

```
// neutrales Element
int summe = 0;

for(int i=1; i <= n; i = i + 1) {
    summe = summe + i;
}
```

# Geschachtelte Schleife

- Beobachtung: Eine Zählschleife ist eine Anweisung
- Also möglich: Schleifenrumpf ist wieder eine Zählschleife (oder jede andere Anweisung)
- Aufgabenstellung: berechne $\sum_{i=1}^{n}\sum_{j=1}^{n}(i+j)$

```
public class SummeUeberIUndJ {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);     // Beispielwert

        // Ergebnisvariable mit neutralem Element der Operation
        int summe = 0;

        for(int i=1; i <= n; i = i + 1) {
            for(int j=1; j <= n; j = j + 1) {
                summe = summe + (i + j);
            }
        }

        System.out.println("Doppelsumme ueber " + n + " ist " + summe);
    }
}
```

# Alle möglichen Kombinationen erzeugen

Aufgabe: erzeuge alle dreistelligen Bit-Kombination xyz, x,y,z $\in$ {0,1} und rechne den Zweierkomplementwert dieser Bitkombinationen aus

```
/**
 * gebe alle moeglichen 3-stelligen Zweierkomplementzahlen aus
 */
public class Binaerzahlen {
    public static void main(String[] args) {

        // Erzeugen aller moeglichen Kombination von drei Binaerziffern
        for(int i2=0; i2 <= 1; i2=i2+1) {
            for(int i1=0; i1 <= 1; i1=i1+1) {
                for(int i0=0; i0 <= 1; i0=i0+1) {
                  // dies ist die Formel fuer den Zweierkomplementwert
                  int wert = ((-i2*4) + i1*2 + i0*1);
                  System.out.println("" + i2 + i1 + i0 + " entspricht " + wert);
                }
            }
        }
    }
}
```

# Beliebige Start-/Endwerte

Aufgabe: gebe alle Quadrat- und Kubikzahlen zwischen 10 und 20 aus

```
/**
 * Quadratzahlen und Kubikzahlen zwischen 10 und 20 erzeugen
 */
public class Quadratzahlen {
    public static void main(String[] args) {

        // Ueberschrift ausgeben
        System.out.println("n  n^2  n^3");

        // von 10 beginnend bis einschliesslich 20 zaehlen
        for(int i=10; i <= 20; i=i+1) {
            // i*i ist die Quadratzahl zu i, i*i*i die Kubikzahl
            System.out.println(i + " " + (i*i) + " " + (i*i*i));
        }
    }
}
```

# Rückwärtszählen auch möglich

Aufgabe: gebe alle Quadrat- und Kubikzahlen zwischen 10 und 20
in umgekehrter Reihenfolge aus

```
/**
 * Quadratzahlen und Kubikzahlen zwischen 10 und 20 rueckwaerts erzeugen
 */
public class Quadratzahlen2 {
    public static void main(String[] args) {

        // Ueberschrift ausgeben
        System.out.println("n  n^2  n^3");

        // von 20 beginnend bis einschliesslich 10 runterzaehlen
        for(int i=20; i >= 10; i=i-1) {
            // i*i ist die Quadratzahl zu i, i*i*i die Kubikzahl
            System.out.println(i + " " + (i*i) + " " + (i*i*i));
        }
    }
}
```

# Wir finanzieren ein Haus...

- Ein Darlehensnehmer leiht sich das Kapital K Euro (Darlehen) zu einem Zinssatz von p Prozent pro Jahr für n Monate
- Über die Laufzeit des Darlehens zahlt er pro Monat eine feste Summe R
- Von R werden die fälligen Monatszinsen $R_1$ abgehalten, der Rest $R_2=R-R_1$ dient zur Tilgung der Darlehenssumme
- Durch eine Tilgung reduziert sich der Darlehensbetrag, der für die Zinsberechnung im nächsten Monat maßgeblich ist
- Es sollte in der Praxis gelten: $R > R_1$

- Ist die Tilgung im ersten Monat größer 0, so gilt für den nächsten Monat (und damit auch für alle nachfolgenden):
  - Der Darlehensbetrag reduziert sich
  - Dadurch sind die Zinsen $R_1$ geringer
  - Dadurch wird die Tilgung $R_2=R-R_1$ höher

# Programm

```
public class Annuitaetendarlehen {
  public static void main(String[] args) {
    double kapital = 100000.0;     // Darlehnssumme 100000 Euro
    double zinssatz = 0.06;        // Zinssatz 6% p.a.
    double zahlung = 1000.0;       // monatliche Zahlung 1000 Euro
    int laufzeit = 12;             // 12 Monate Laufzeit
    double tilgung, zinsen, summe_zinsen = 0.0, summe_tilgung = 0.0;

    System.out.println("Monat Restschuld Zinsen Tilgung");
    for(int monat = 1; monat <= laufzeit; monat = monat + 1) {
        zinsen = kapital * zinssatz / 12.0;
        tilgung = zahlung - zinsen;
        kapital = kapital - tilgung;
        summe_zinsen = summe_zinsen + zinsen;
        summe_tilgung = summe_tilgung + tilgung;
        System.out.println(monat + "  "+kapital+"  "+zinsen+"  "+tilgung);
    }

    System.out.println("Summe Zahlungen: " + (summe_zinsen + summe_tilgung));
    System.out.println("Summe Zinsen: " + summe_zinsen);
    System.out.println("Summe Tilgung: " + summe_tilgung);
    System.out.println("Restschuld: " + kapital);
  }
}
```

# Zwischenstand

- Zählschleifen dienen i.d.R. der Wiederholung von Aktionen in Abhängigkeit von einer Zählung
- Mit der ganzzahligen Schleifenvariablen steht ein Zähler zur Verfügung, dessen Wert man im Schleifenrumpf nutzen kann
- Die Aufzählung kann vorwärts, rückwärts, in größeren Schritten,… erfolgen
- Schleifen können geschachtelt werden

**Reflektion**

- Wie müsste der Kopf einer Zählschleife aussehen, so dass die Zählvariable für ein gegebenes n die Werte 1,2,4,8,16,32,64,…,$2^n$ annimmt?