Einführung in die Programmierung

Vorlesungsteil 3
Imperative Kontrollstrukturen

PD Dr. Thomas Hinze

Brandenburgische Technische Universität Cottbus – Senftenberg
Institut für Informatik

Sommersemester 2016

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Imperativ heißt befehlsorientiert

Bildquelle: www.wikipedia.org

Computer

Programm

Schreibe 20 mal auf englisch an die Tafel: "Ich darf keine Kreide verschwenden!"

Kurze Befehle können viel Ausführungsarbeit nach sich ziehen.

Kontrollstrukturen erschließen die Rechenkraft des Computers

"Der Computer ist unendlich dumm,
aber auch unendlich fleissig."

Kontrollstrukturen erschließen die Rechenkraft des Computers

**"Der Computer ist unendlich dumm,
aber auch unendlich fleissig."**

Der Computer kann *gleiche Anweisungsfolgen* (Berechnungsschritte) mit immer wieder anderen oder sich verändernden Werten sehr oft *wiederholen*.

Kontrollstrukturen erschließen die Rechenkraft des Computers

"Der Computer ist unendlich dumm, aber auch unendlich fleissig."

Der Computer kann *gleiche Anweisungsfolgen* (Berechnungsschritte) mit immer wieder anderen oder sich verändernden Werten sehr oft *wiederholen*.

Dadurch wird in der imperativen Programmierung eine *kurze Beschreibung langwieriger Berechnungen* möglich.

Vorlesung Einführung in die Programmierung mit Java

1. **Einführung und erste Schritte** . . . . . .
   . . Installation Java-Compiler, ein erstes Programm: HalloWelt, Blick in den Computer
2. **Elementare Datentypen, Variablen, Arithmetik, Typecast** . . . . . .
   Java als Taschenrechner nutzen, Tastatureingabe $\rightarrow$ Formelberechnung $\rightarrow$ Ausgabe
3. **Imperative Kontrollstrukturen** . . . . . .
   . . . Befehlsfolgen, Verzweigungen, Schleifen und logische Ausdrücke programmieren
4. **Methoden selbst programmieren** . . . . . .
   . . . . Methoden als wiederverwendbare Funktionen, Werteübernahme und -rückgabe
5. **Rekursion** . . . . . .
   . . . . selbstaufrufende Funktionen als elegantes algorithmisches Beschreibungsmittel
6. **Objektorientiert programmieren** . . . . . .
   . . . . . Klassen, Objekte, Attribute, Methoden, Sichtbarkeit, Vererbung, Polymorphie
7. **Felder und Graphen** . . . . . .
   . . . . . effizientes Handling größerer Datenmengen und Beschreibung von Netzwerken
8. **Sortieren** . . . . . .
   . . . . . klassische Sortierverfahren im Überblick, Laufzeit und Speicherplatzbedarf
9. **Zeichenketten, Dateiarbeit, Ausnahmen** . . . . . .
   . . . Texte analysieren, ver-/entschlüsseln, Dateien lesen/schreiben, Fehler behandeln
10. **Dynamische Datenstruktur „Lineare Liste“** . . . . . .
    . . . . . unsere selbstprogrammierte kleine Datenbank
11. **Ausblick und weiterführende Konzepte** . . . . . .

Imperative Kontrollstrukturen

Imperative Kontrollstrukturen sind die Beschreibungsmittel, mit denen der Programmierer im Programmquelltext festlegt, *welche Befehle unter welchen Bedingungen in welcher Reihenfolge* durch den Computer abgearbeitet werden.
Dazu gehören:

Imperative Kontrollstrukturen

Imperative Kontrollstrukturen sind die Beschreibungsmittel, mit denen der Programmierer im Programmquelltext festlegt, *welche Befehle unter welchen Bedingungen in welcher Reihenfolge* durch den Computer abgearbeitet werden.
Dazu gehören:

- **Befehlsfolgen**,
  auch *Anweisungsfolgen*, *Sequenzen* oder *Blöcke* genannt

Imperative Kontrollstrukturen

Imperative Kontrollstrukturen sind die Beschreibungsmittel, mit denen der Programmierer im Programmquelltext festlegt, *welche Befehle unter welchen Bedingungen in welcher Reihenfolge* durch den Computer abgearbeitet werden.
Dazu gehören:

- **Befehlsfolgen**,
  auch *Anweisungsfolgen*, *Sequenzen* oder *Blöcke* genannt
- **Fallunterscheidungen**,
  auch *Verzweigungen* oder *bedingte Anweisungen* genannt

Imperative Kontrollstrukturen

Imperative Kontrollstrukturen sind die Beschreibungsmittel, mit denen der Programmierer im Programmquelltext festlegt, *welche Befehle unter welchen Bedingungen in welcher Reihenfolge* durch den Computer abgearbeitet werden.
Dazu gehören:

- **Befehlsfolgen**,
  auch *Anweisungsfolgen*, *Sequenzen* oder *Blöcke* genannt
- **Fallunterscheidungen**,
  auch *Verzweigungen* oder *bedingte Anweisungen* genannt
- **Schleifen**,
  auch *iterative Konstrukte* (engl. iterate = wiederholen)

Imperative Kontrollstrukturen

Imperative Kontrollstrukturen sind die Beschreibungsmittel, mit denen der Programmierer im Programmquelltext festlegt, *welche Befehle unter welchen Bedingungen in welcher Reihenfolge* durch den Computer abgearbeitet werden.
Dazu gehören:

- **Befehlsfolgen**,
  auch *Anweisungsfolgen*, *Sequenzen* oder *Blöcke* genannt
- **Fallunterscheidungen**,
  auch *Verzweigungen* oder *bedingte Anweisungen* genannt
- **Schleifen**,
  auch *iterative Konstrukte* (engl. iterate = wiederholen)

$\Longrightarrow$ Imperative Kontrollstrukturen *steuern* (engl. control) den Programmablauf.

Die 51 Schlüsselwörter von Java

Java als kompakte Programmiersprache

```
abstract  double      long      static
boolean   else        native    super
break     extends     new       switch
byte      final       null      synchronized
case      finally     operator  this
cast      float       outer     throw
catch     for         package   throws
char      if          private   transient
class     implements  protected try
const     import      public    var
continue  instanceof  rest      void
default   int         return    while
do        interface   short
```

Die 51 Schlüsselwörter von Java

Heute und nächste Woche lernen wir davon kennen . . .

abstract	double	long	static
boolean	**else**	native	super
break	extends	new	**switch**
byte	final	null	synchronized
case	finally	operator	this
cast	float	outer	throw
catch	**for**	package	throws
char	**if**	private	transient
class	implements	protected	try
const	import	public	var
continue	instanceof	rest	void
default	int	return	**while**
do	interface	short	

Block

Ein **Block** beschreibt die *Hintereinanderausführung* (sequentielle Abarbeitung) von Anweisungen. Jeder Block wird durch `{` und `}` begrenzt.

```
{
   Deklarationen; //lokale Variablen und Konstanten
   Anweisung 1;
   Anweisung 2;
   :
   Anweisung k;
}
```

- Hinter schließender Klammer `}` am Blockende kein Semikolon
- Jeder Block gilt von außen betrachtet wie einzelne Anweisung
- Rumpf jeder Methode ist stets ein Block (vgl. `main`-Methode)
- Variablen und Konstanten gelten stets innerhalb des Blocks, in welchem sie deklariert wurden

Blockschachtelungen und Lebensdauer von Variablen

```
public class Bloecke {
  public static void main(String[] args) {
    int a = 3;

    {
        System.out.printf("a: %d\n", a);
        //a ist 3
    }

    {
        int b = 5;
        System.out.printf("b: %d\n", b);
        //b ist 5
        {
            int c = 7;
            System.out.printf("c: %d\n", c);
            //c ist 7
        }
    }
    System.out.printf("a: %d\n", a);
    //b und c nicht mehr existent.
  }

  /* ausserhalb der main-Methode existieren
     die Variablen a, b und c nicht.
  */

}
```

- Variablen und Konstanten innerhalb einer Methode stets gültig in dem Block, in dem sie deklariert wurden einschließlich aller darin eingeschachtelten Blöcke.
- Sobald die Programmabarbeitung ein Blockende erreicht, endet die Lebensdauer der dort deklarierten Variablen und Konstanten, und ihr Speicherplatz wird freigegeben.

Fallunterscheidungen in der Programmierung

www.webdesign.org

Fallunterscheidung mit `if else`

In Abhängigkeit davon, ob eine vorgegebene *Bedingung* erfüllt ist oder nicht, werden unterschiedliche Blöcke abgearbeitet.

```
if (Bedingung)      //Falls Bedingung erfuellt
  Block 1
else                //sonst
  Block 2
```

Fallunterscheidung mit `if else`

In Abhängigkeit davon, ob eine vorgegebene *Bedingung* erfüllt ist oder nicht, werden unterschiedliche Blöcke abgearbeitet.

```
if (Bedingung)      //Falls Bedingung erfuellt
  Block 1
else                //sonst
  Block 2
```

- **`Bedingung`** ist ein logischer Ausdruck vom Typ **`boolean`**, der einen Wahrheitswert (**`true`** oder **`false`**) als Ergebnis liefert

Fallunterscheidung mit `if else`

In Abhängigkeit davon, ob eine vorgegebene *Bedingung* erfüllt ist oder nicht, werden unterschiedliche Blöcke abgearbeitet.

```
if (Bedingung)      //Falls Bedingung erfuellt
  Block 1
else                //sonst
  Block 2
```

- **`Bedingung`** ist ein logischer Ausdruck vom Typ **`boolean`**, der einen Wahrheitswert (**`true`** oder **`false`**) als Ergebnis liefert
- Ist die Bedingung *wahr* (**`true`**), wird **`Block 1`** abgearbeitet

Fallunterscheidung mit `if else`

In Abhängigkeit davon, ob eine vorgegebene *Bedingung* erfüllt ist oder nicht, werden unterschiedliche Blöcke abgearbeitet.

```
if (Bedingung)      //Falls Bedingung erfuellt
  Block 1
else                //sonst
  Block 2
```

- **`Bedingung`** ist ein logischer Ausdruck vom Typ **`boolean`**, der einen Wahrheitswert (**`true`** oder **`false`**) als Ergebnis liefert
- Ist die Bedingung *wahr* (**`true`**), wird **`Block 1`** abgearbeitet
- Ist die Bedingung *falsch* (**`false`**), wird **`Block 2`** abgearbeitet

Fallunterscheidung mit `if else`

In Abhängigkeit davon, ob eine vorgegebene *Bedingung* erfüllt ist oder nicht, werden unterschiedliche Blöcke abgearbeitet.

```
if (Bedingung)      //Falls Bedingung erfuellt
  Block 1
else                //sonst
  Block 2
```

- **`Bedingung`** ist ein logischer Ausdruck vom Typ **`boolean`**, der einen Wahrheitswert (**`true`** oder **`false`**) als Ergebnis liefert
- Ist die Bedingung *wahr* (**`true`**), wird **`Block 1`** abgearbeitet
- Ist die Bedingung *falsch* (**`false`**), wird **`Block 2`** abgearbeitet
- Bedingung stets in runde Klammern **`()`** einbetten

Fallunterscheidung mit `if else`

In Abhängigkeit davon, ob eine vorgegebene *Bedingung* erfüllt ist oder nicht, werden unterschiedliche Blöcke abgearbeitet.

```
if (Bedingung)      //Falls Bedingung erfuellt
  Block 1
else                //sonst
  Block 2
```

- **`Bedingung`** ist ein logischer Ausdruck vom Typ **`boolean`**, der einen Wahrheitswert (**`true`** oder **`false`**) als Ergebnis liefert
- Ist die Bedingung *wahr* (**`true`**), wird **`Block 1`** abgearbeitet
- Ist die Bedingung *falsch* (**`false`**), wird **`Block 2`** abgearbeitet
- Bedingung stets in runde Klammern **`()`** einbetten
- **`else`**-Zweig darf weggelassen werden, wenn **`Block 2`** leer ist

Fallunterscheidung mit `if else`

In Abhängigkeit davon, ob eine vorgegebene *Bedingung* erfüllt ist oder nicht, werden unterschiedliche Blöcke abgearbeitet.

```
if (Bedingung)      //Falls Bedingung erfuellt
  Block 1
else                //sonst
  Block 2
```

- **`Bedingung`** ist ein logischer Ausdruck vom Typ **`boolean`**, der einen Wahrheitswert (**`true`** oder **`false`**) als Ergebnis liefert
- Ist die Bedingung *wahr* (**`true`**), wird **`Block 1`** abgearbeitet
- Ist die Bedingung *falsch* (**`false`**), wird **`Block 2`** abgearbeitet
- Bedingung stets in runde Klammern **`()`** einbetten
- **`else`**-Zweig darf weggelassen werden, wenn **`Block 2`** leer ist
- Gesamtes **`if`**-**`else`**-Konstrukt nach außen wie einzelner Block aufgefasst

Beispiel: Maximum aus zwei Zahlen (Max2.java)

```
import java.util.Scanner;

public class Max2 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    double a, b, max;

    System.out.println("Maximum aus zwei Zahlen");
    System.out.print("Erste Zahl: ");
    a = Double.parseDouble(sc.next());
    System.out.print("Zweite Zahl: ");
    b = Double.parseDouble(sc.next());

    if (a > b)
    {
      max = a;
    }
    else
    {
      max = b;
    }

    System.out.printf("Maximum: %f\n", max);
  }
}
```

- Besteht der Block im if- oder else-Zweig nur aus einer einzigen Anweisung, dann dürfen die geschweiften Blockklammern weggelassen werden.
- Dies verschlechtert aber die Lesbarkeit des Quelltextes, deshalb sei davon abgeraten.

`if-else` zur Gültigkeits- und Plausibilitätsprüfung

Beispiel: Quadratisches Polynom $x^2 + px + q$ besitzt reelle Nullstellen

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

Berechnung nur dann ausführen, wenn sie zulässig ist

```
if (p*p/4.0 < q)
{
  System.out.print("keine reellen Nullst.");
}
else
{
  double z = -p/2.0;
  double x1 = z + Math.sqrt(z*z - q);
  double x2 = z - Math.sqrt(z*z - q);
  System.out.printf("x1: %f, x2: %f", x1, x2);
}
```

Mehrfachverzweigungen mit `else if`

zum Auswählen aus mehr als zwei Alternativen:

```
if (Bedingung 1)
  Block 1
else if (Bedingung 2)
  Block 2
else if (Bedingung 3)
  Block 3
:
else if (Bedingung k)
  Block k
else
  Block k+1
```

Mehrfachverzweigungen mit `else if`

zum Auswählen aus mehr als zwei Alternativen:

```
if (Bedingung 1)
   Block 1
else if (Bedingung 2)
   Block 2
else if (Bedingung 3)
   Block 3
:
else if (Bedingung k)
   Block k
else
   Block k+1
```

- Das letzte `else` ist optional und bezieht sich auf `Bedingung k`
- Abarbeitung verfolgt das Ausschließungsprinzip:
  Ist `Bedingung 1` nicht erfüllt, teste `Bedingung 2` usw.

Beispiel: Body-Mass-Index (Bmi.java)

```
public class Bmi {
  public static void main(String[] args) {
    double h = 1.77; //Koerpergroesse in m
    double m = 83.1; //Koerpermasse in kg
    double bmi = m / (h * h);

    if (bmi < 18.5) {
      System.out.println("Untergewichtig");
    }
    else if (bmi < 25) {
      System.out.println("Normalgewichtig");
    }
    else if (bmi < 30) {
      System.out.println("Leicht uebergewichtig");
    }
    else {
      System.out.println("Fettleibig");
    }
  }
}
```

Fallunterscheidungen mit `switch case`

zum Auswählen aus einem Pool ganzzahliger Werte als Alternativen:

```
switch (Ganzzahlausdruck)
{
  case Konst_1:  Block 1
                 break;        // optional
  case Konst_2:  Block 2
                 break;        // optional
  :
  case Konst_k:  Block k
                 break;        // optional
  default:        Block k+1    // optional
}
```

Fallunterscheidungen mit `switch case`

zum Auswählen aus einem Pool ganzzahliger Werte als Alternativen:

```
switch (Ganzzahlausdruck)
{
  case Konst_1:  Block 1
                 break;        // optional
  case Konst_2:  Block 2
                 break;        // optional
  :
  case Konst_k:  Block k
                 break;        // optional
  default:        Block k+1    // optional
}
```

- Die einzelnen **`break`**-Befehle sowie die **`default`**-Zeile können weggelassen werden
- **`Konst_1`** bis **`Konst_k`** müssen jeweils Konstantenwerte sein, (Formel)ausdrücke sind dort nicht zulässig

Funktionsweise von `switch case`

- Hinter `switch` angegebener `Ganzzahlausdruck` mit den Konstantenwerten beginnend ab `Konst_1` auf *Gleichheit* verglichen

Funktionsweise von `switch case`

- Hinter **`switch`** angegebener **`Ganzzahlausdruck`** mit den Konstantenwerten beginnend ab **`Konst_1`** auf *Gleichheit* verglichen
- Sobald erstmalig Übereinstimmung gefunden (*„Match“*), von dort an ALLE weiteren Blöcke bis zum nächsten **`break`** oder Ende des **`switch`**-Konstrukts ausgeführt (*„Durchfalleffekt“*)

Funktionsweise von `switch case`

- Hinter **`switch`** angegebener **`Ganzzahlausdruck`** mit den Konstantenwerten beginnend ab **`Konst_1`** auf *Gleichheit* verglichen
- Sobald erstmalig Übereinstimmung gefunden (*„Match“*), von dort an ALLE weiteren Blöcke bis zum nächsten **`break`** oder Ende des **`switch`**-Konstrukts ausgeführt (*„Durchfalleffekt“*)
- Wenn keinerlei Übereinstimmung, Block des optionalen **`default`**-Falls ausgeführt

Beispiel: Würfel mit Pünktchenausgabe (Wuerfel.java)

```
public class Wuerfel {
  public static void main(String[] args) {

    int augenzahl = (int) (Math.random() * 6) + 1;

    switch (augenzahl)
    {
      case 1: System.out.printf("\n     \n     \n  *  \n     \n     \n\n"); break;
      case 2: System.out.printf("\n    *\n     \n     \n     \n*    \n\n"); break;
      case 3: System.out.printf("\n    *\n     \n  *  \n     \n*    \n\n"); break;
      case 4: System.out.printf("\n*   *\n     \n     \n     \n*   *\n\n"); break;
      case 5: System.out.printf("\n*   *\n     \n  *  \n     \n*   *\n\n"); break;
      case 6: System.out.printf("\n*   *\n     \n*   *\n     \n*   *\n\n"); break;
      default : System.out.printf("\n\n");
    }
  }
}
```

Den „Durchfall“ ohne `break` geschickt nutzen

TageProMonat2016.java – Monatslängen in Tagen 2016

```
public class TageProMonat2016 {
  public static void main(String[] args) {
    int monat = 11; //1: Januar, 2: Februar, ..., 12: Dezember
    int anzTage;

    switch(monat)
    {
      case 1:
      case 3:
      case 5:
      case 7:
      case 8:
      case 10:
      case 12: anzTage = 31; break;

      case 4:
      case 6:
      case 9:
      case 11: anzTage = 30; break;

      case 2: anzTage = 29; break;
      default: anzTage = -1; //Fehlerfall
    }
    System.out.printf("Anzahl Tage: %d\n", anzTage);
  }
}
```

Gegenüberstellung `if-else` und `switch-case`

Für welche Einsatzszenarien empfiehlt sich welches Konstrukt?

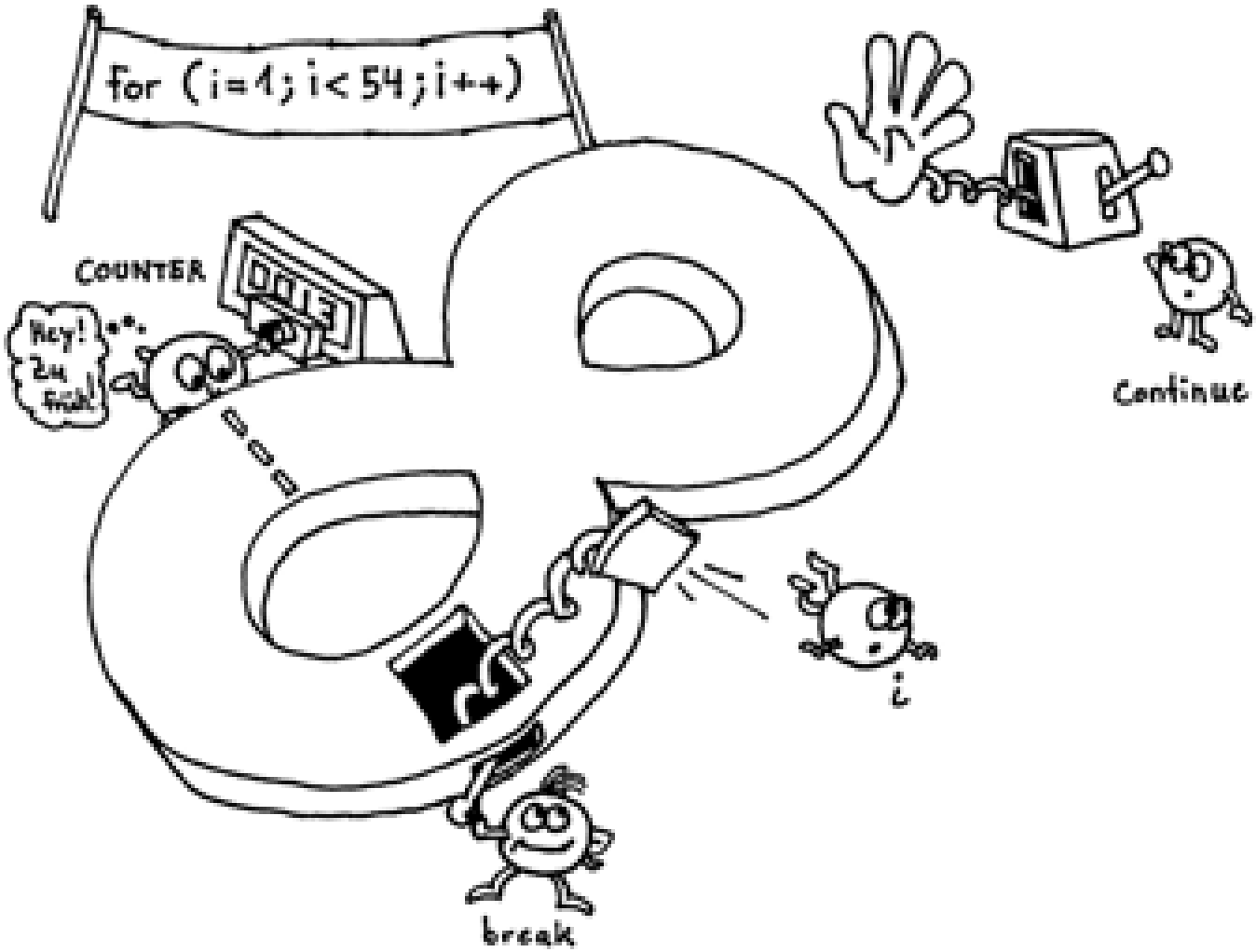
`if-else`

- flexibler
- < und > in Bedingungen zulässig, somit Intervalle beschreibbar
- Vergleiche nicht auf Ganzzahlen eingeschränkt
- Quelltext schwerfälliger und schreibaufwendiger
- compilertechnisch weniger optimierbar

`switch-case`

- im Maschinenprogramm nach Compilierung oft schneller ausführbar
- übersichtlicher im Quelltext und damit geringere Wahrscheinlichkeit für Programmierfehler (wie vergessene Fallalternativen)
- durch recht starre Strukturvorgabe weniger beschreibungsmächtig

Schleifen in der Programmierung



openbook.galileocomputing.de

Schleifen in der Programmierung

- Eine *Schleife* dient dazu, einen Block von Anweisungen *wiederholt* abzuarbeiten.

Schleifen in der Programmierung

- Eine *Schleife* dient dazu, einen Block von Anweisungen *wiederholt* abzuarbeiten.
- Mit jedem *Schleifendurchlauf* (jeder *Iteration*) können sich innerhalb des Blocks Variablenwerte verändern

Schleifen in der Programmierung

- Eine *Schleife* dient dazu, einen Block von Anweisungen *wiederholt* abzuarbeiten.
- Mit jedem *Schleifendurchlauf* (jeder *Iteration*) können sich innerhalb des Blocks Variablenwerte verändern
- Vor oder nach jedem Durchlauf (je nach Schleifenart) wird eine frei definierbare *Bedingung* geprüft.

Schleifen in der Programmierung

- Eine *Schleife* dient dazu, einen Block von Anweisungen *wiederholt* abzuarbeiten.
- Mit jedem *Schleifendurchlauf* (jeder *Iteration*) können sich innerhalb des Blocks Variablenwerte verändern
- Vor oder nach jedem Durchlauf (je nach Schleifenart) wird eine frei definierbare *Bedingung* geprüft.
- Ist die Bedingung erfüllt (`true`), wird die Schleife (erneut) durchlaufen.

Schleifen in der Programmierung

- Eine *Schleife* dient dazu, einen Block von Anweisungen *wiederholt* abzuarbeiten.
- Mit jedem *Schleifendurchlauf* (jeder *Iteration*) können sich innerhalb des Blocks Variablenwerte verändern
- Vor oder nach jedem Durchlauf (je nach Schleifenart) wird eine frei definierbare *Bedingung* geprüft.
- Ist die Bedingung erfüllt (`true`), wird die Schleife (erneut) durchlaufen.
- Ist sie nicht erfüllt (`false`), endet die Abarbeitung der Schleife und das Programm wird mit dem nächsten Befehl unmittelbar hinter dem Block der Schleife fortgesetzt.

Schleifen in der Programmierung

- Eine *Schleife* dient dazu, einen Block von Anweisungen *wiederholt* abzuarbeiten.
- Mit jedem *Schleifendurchlauf* (jeder *Iteration*) können sich innerhalb des Blocks Variablenwerte verändern
- Vor oder nach jedem Durchlauf (je nach Schleifenart) wird eine frei definierbare *Bedingung* geprüft.
- Ist die Bedingung erfüllt (`true`), wird die Schleife (erneut) durchlaufen.
- Ist sie nicht erfüllt (`false`), endet die Abarbeitung der Schleife und das Programm wird mit dem nächsten Befehl unmittelbar hinter dem Block der Schleife fortgesetzt.
- Schleifen können auch unendlich oft durchlaufen werden (*Endlosschleife*), dann ist das Programm abgestürzt und kann per Tastatur durch *Strg-C* abgebrochen werden. Endlosschleifen sollten vermieden werden.

Schleifen in der Programmierung

- Eine *Schleife* dient dazu, einen Block von Anweisungen *wiederholt* abzuarbeiten.
- Mit jedem *Schleifendurchlauf* (jeder *Iteration*) können sich innerhalb des Blocks Variablenwerte verändern
- Vor oder nach jedem Durchlauf (je nach Schleifenart) wird eine frei definierbare *Bedingung* geprüft.
- Ist die Bedingung erfüllt (`true`), wird die Schleife (erneut) durchlaufen.
- Ist sie nicht erfüllt (`false`), endet die Abarbeitung der Schleife und das Programm wird mit dem nächsten Befehl unmittelbar hinter dem Block der Schleife fortgesetzt.
- Schleifen können auch unendlich oft durchlaufen werden (*Endlosschleife*), dann ist das Programm abgestürzt und kann per Tastatur durch *Strg-C* abgebrochen werden. Endlosschleifen sollten vermieden werden.
- Schleifenarten: `while`, `do while` und `for`

`while`-Schleife

Prüfung der Bedingung **vor** jedem Schleifendurchlauf

```
while (Bedingung)
  Block
```

- Solange (engl. *while*) `(Bedingung)` erfüllt, führe `Block` aus
- Ist `(Bedingung)` beim ersten Test nicht erfüllt, wird `Block` gar nicht ausgeführt
- kopfgesteuerte Schleife

Beispiel für eine `while`-Schleife (PiReihe.java)

Annäherung von π durch endlichen Teil der Reihe $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$

```
public class PiReihe {
  public static void main(String[] args) {
    int k = 1;
    int n = 30; //Anzahl Summanden
    double x = 0.0;
    double y;
    while (k <= n)
    {
      x = x + 1.0 / (k*k);
      y = Math.sqrt(6*x);
      System.out.printf("Bei k=%d: %f\n",k,y);
      k = k + 1;
    }
  }
}
```

Schleifenabbruch mit `break` (PiReihe2.java)

```
public class PiReihe2 {
  public static void main(String[] args) {
    final double EPSILON = 1E-5; //Genauigkeitsgrenze
    int k = 1;
    double y = 0.0, x = 0.0;

    while (true)
    {
      y = x;
      x = x + 1.0 / (k*k);
      if ((x-y)/x < EPSILON*EPSILON)
      {
        break;
      }
      System.out.printf("Bei k = %d ist pi %f\n", k, Math.sqrt(6*x));
      k = k + 1;
    }
  }
}
```

- `while (true) {...}` definiert eine *Endlosschleife*
- Bei Erreichen des Schlüsselwortes `break` sofortiger Schleifenabbruch

Vorzeitige Rückkehr zur Bedingungsprüfung (Cont.java)

```
public class Cont {
  public static void main(String[] args) {
    int i = 1;

    while (i < 40)
    {
      if (i % 8 == 0)
      {
        i = i + 7;
        continue;
      }
      i = i + 1;
      System.out.printf("i ist %d\n", i);
    }
  }
}
```

```
i ist 2
i ist 3
i ist 4
i ist 5
i ist 6
i ist 7
i ist 8
i ist 16
i ist 24
i ist 32
i ist 40
```

- Bei Erreichen des Schlüsselwortes **`continue`** sofortiges Verlassen des Schleifenblocks und Rückkehr zur Bedingungsprüfung
- **`continue`** ermöglicht mitunter kompakte Notation, führt aber oft zu schwer verständlichem Quelltext

`do while`-Schleife

Prüfung der Bedingung **nach** jedem Schleifendurchlauf

```
do
   Block
while (Bedingung);
```

- Führe `Block` aus, solange `(Bedingung)` erfüllt.
- `Block` wird mindestens einmal durchlaufen
- fußgesteuerte Schleife

`do while`-Schleife

Prüfung der Bedingung **nach** jedem Schleifendurchlauf

```
do
   Block
while (Bedingung);
```

- Führe `Block` aus, solange `(Bedingung)` erfüllt.
- `Block` wird mindestens einmal durchlaufen
- fußgesteuerte Schleife
- kann als Endlosschleife agieren: `do {...} while (true);`

`do while`-Schleife

Prüfung der Bedingung **nach** jedem Schleifendurchlauf

```
do
   Block
while (Bedingung);
```

- Führe `Block` aus, solange `(Bedingung)` erfüllt.
- `Block` wird mindestens einmal durchlaufen
- fußgesteuerte Schleife
- kann als Endlosschleife agieren: `do {...} while (true);`
- `break` und `continue` erlaubt mit gleicher Wirkung wie in `while`-Schleife

Beispiel: Wiederhole Eingabe, solange ungültig

(Age.java)

```
import java.util.Scanner;

public class Age {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int age;

    do
    {
      System.out.print("Bitte geben Sie Ihr Alter in Jahren ein: ");
      age = Integer.parseInt(sc.next());
    } while ((age < 0) || (age > 130));
  }
}
```

`for`-Schleife: Intention einer Zählschleife

Prüfung der Bedingung **vor** jedem Schleifendurchlauf

```
for (Initialisierung; Bedingung; Zaehlschritt)
  Block
```

1. Zuerst Anweisung `Initialisierung` ausgeführt
2. Danach `Bedingung` geprüft. Falls nicht erfüllt (`false`): Schleifenabarbeitung beendet
3. Ansonsten Abarbeitung der Anweisungen im `Block`
4. Anschließend Abarbeitung der Anweisung im `Zaehlschritt`
5. Weiter mit 2.

`for`-Schleife als Zählschleife (For10.java)

```
public class For10 {
  public static void main(String[] args) {
    int i;

    for (i = 1; i <= 10; i++)
    {
      System.out.printf("i ist %d\n", i);
    }
  }
}
```

```
i ist 1
i ist 2
i ist 3
i ist 4
i ist 5
i ist 6
i ist 7
i ist 8
i ist 9
i ist 10
```

`for`-Schleife als Zählschleife (For10.java)

```
public class For10 {
  public static void main(String[] args) {
    int i;

    for (i = 1; i <= 10; i++)
    {
      System.out.printf("i ist %d\n", i);
    }
  }
}
```

```
i ist 1
i ist 2
i ist 3
i ist 4
i ist 5
i ist 6
i ist 7
i ist 8
i ist 9
i ist 10
```

- kopfgesteuerte Schleife
- Komponenten **`Initialisierung`**, **`Bedingung`** und/oder **`Zaehlschritt`** dürfen auch leergelassen werden
- Endlosschleife: **`for(;;) {...}`**
- **`break`** und **`continue`** erlaubt mit gleicher Wirkung wie in **`while`**-Schleife

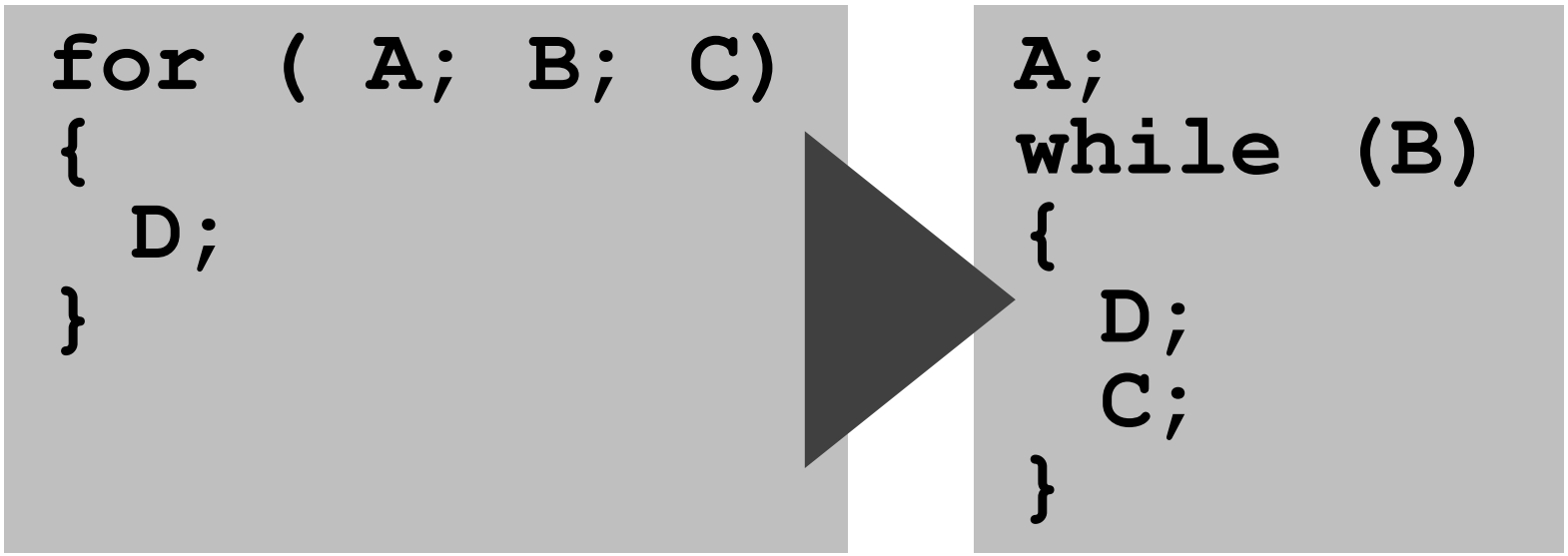
Einfacher Pseudozufallszahlengenerator (Pseudozfg.java)

```
public class Pseudozfg {
  public static void main(String[] args) {
    int i;
    int x = 5; //Startwert

    for (i = 1; i <= 6; i++)
    {
      x = (4*(x + 7)) % 15;
      System.out.printf("Pseudozufallszahl: %d\n", x);
    }
  }
}
```

$\Longrightarrow$ Periodenlänge: 6, dann wiederholt sich Zahlenfolge

Simulation einer `for`-Schleife als `while`-Schleife



$\Longrightarrow$ Alle drei Schleifenarten **`while`**, **`do while`** und **`for`** besitzen *gleiche Ausdrucksmächtigkeit*, können sich also beliebig gegenseitig simulieren. Eigentlich würde zum Programmieren eine beliebige der drei Schleifenarten ausreichen.

Gegenüberstellung `while`, `do while` und `for`

Wofür empfiehlt sich welche Schleifenart?

`while`	`do while`	`for`
• Schleifen, die u.U. gar nicht durchlaufen werden • Bedingung muss vor Schleifendurch-lauf auswertbar sein • numerische Berechnungen, bei denen mit jeder Iteration Werte aktualisiert werden	• Schleifen, die mindestens einmal durchlaufen werden müssen • Bedingung muss erst nach Schleifendurchlauf auswertbar sein • Eingaben, die solange wiederholt werden, bis gültige Werte vorliegen	• Zählschleife mit bekannter Anzahl Durchläufe • schrittweises / elementweises Durchlaufen von Zahlenfolgen a_i über einen Index i • Aufzählungen

Ineinanderschachteln mehrerer Schleifen

Beispiel: Alle dreibuchstabigen Wörter aus den Zeichen A, B und C (Abc.java)

```
public class Abc {
  public static void main(String[] args) {
    char x, y, z;

    for (x = 1; x <= 3; x++)
    {
      for (y = 1; y <= 3; y++)
      {
        for (z = 1; z <= 3; z++)
        {
          System.out.printf("%c%c%c\n", x+64, y+64, z+64);
        }
      }
    }
  } //main
} //class
```

```
AAA
AAB
AAC
ABA
ABB
ABC
ACA
ACB
ACC
BAA
BAB
BAC
BBA
BBB
BBC
BCA
BCB
BCC
CAA
CAB
CAC
CBA
CBB
CBC
CCA
CCB
CCC
```

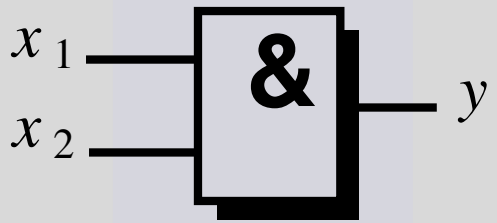
$\Longrightarrow$ insgesamt $3 \cdot 3 \cdot 3 = 27$ Kombinationen erzeugt

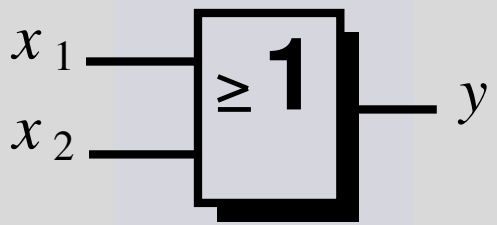
Arbeiten mit logischen Ausdrücken

```
if (Bedingung)
{
  ...
}
```

```
while (Bedingung)
{
  ...
}
```

```
(!(x1 || x2) && (x3 || !x4))
```





In der Programmierung werden häufig aussagenlogische (Boolesche) Formeln verwendet, um *Bedingungen* zu beschreiben, die sich aus *Wahrheitswertetabellen* oder *Schaltfunktionen* ergeben.

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe .....

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe ..... wahr

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe ..... wahr
$5 < 7$ .....

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar
Gleichung $x^x = 10$ nicht exakt lösbar im Bereich rationaler Zahlen	

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar
Gleichung $x^x = 10$ nicht exakt lösbar im Bereich rationaler Zahlen	wahr

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar
Gleichung $x^x = 10$ nicht exakt lösbar im Bereich rationaler Zahlen	wahr
$0^0 = 1$	

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar
Gleichung $x^x = 10$ nicht exakt lösbar im Bereich rationaler Zahlen	wahr
$0^0 = 1$	keine Aussage (nicht definiert)

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar
Gleichung $x^x = 10$ nicht exakt lösbar im Bereich rationaler Zahlen	wahr
$0^0 = 1$	keine Aussage (nicht definiert)
Ich lüge jetzt.	

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar
Gleichung $x^x = 10$ nicht exakt lösbar im Bereich rationaler Zahlen	wahr
$0^0 = 1$	keine Aussage (nicht definiert)
Ich lüge jetzt.	keine Aussage (Antinomie)

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar
Gleichung $x^x = 10$ nicht exakt lösbar im Bereich rationaler Zahlen	wahr
$0^0 = 1$	keine Aussage (nicht definiert)
Ich lüge jetzt.	keine Aussage (Antinomie)
Das Wetter ist schön.	

Begriff Aussage (im logischen Sinne)

Eine **Aussage** ist ein *mathematischer Term* oder ein *sprachliches Gebilde*, dem ein konkreter Wahrheitswert (entweder *wahr* bzw. *falsch*) zugeordnet werden kann.

Blau ist eine Farbe	wahr
$5 < 7$	wahr
$9.81 \in \mathbb{N}$	falsch
4 ist eine Primzahl	falsch
Der Planet Erde wiegt exakt $5.9736 \cdot 10^{24}$kg.	Wahrheitswert derzeit nicht angebbar
Gleichung $x^x = 10$ nicht exakt lösbar im Bereich rationaler Zahlen	wahr
$0^0 = 1$	keine Aussage (nicht definiert)
Ich lüge jetzt.	keine Aussage (Antinomie)
Das Wetter ist schön.	keine Aussage (subjektiver Interpretationsspielraum)

Rechnen mit Aussagen

„Nicht alle Spielzeugbausteine sind weiß oder rot.“

ist gleichbedeutend mit

„Es gibt mindestens einen Spielzeugbaustein, der weder weiß noch rot ist.“

Rechnen mit Aussagen

„Nicht alle Spielzeugbausteine sind weiß oder rot.“

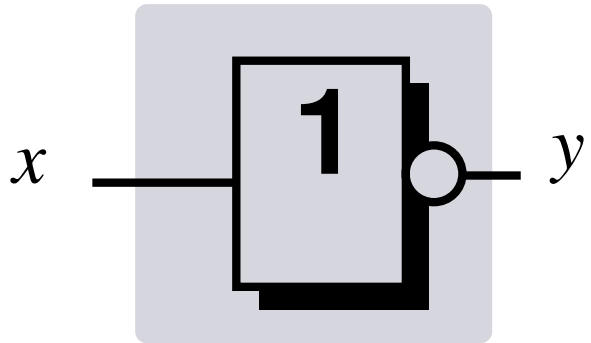
ist gleichbedeutend mit

„Es gibt mindestens einen Spielzeugbaustein, der weder weiß noch rot ist.“

- Eine Aussage lässt sich formalisieren durch eine *Variable* vom Typ `boolean`, die den Wert `true` (wahr) oder `false` (falsch) annehmen kann. Alternativ kann man auch mit den Ganzzahlwerten 1 (wahr) bzw. 0 (falsch) arbeiten.
- *Logische Verknüpfungen* sind Operatoren auf Aussagen.
- Entsprechende Rechengesetze (Boolesche Algebra) erlauben Vereinfachung aussagenlogischer Terme und logisches Schließen.

NICHT-Verknüpfung (NOT, Inverter)

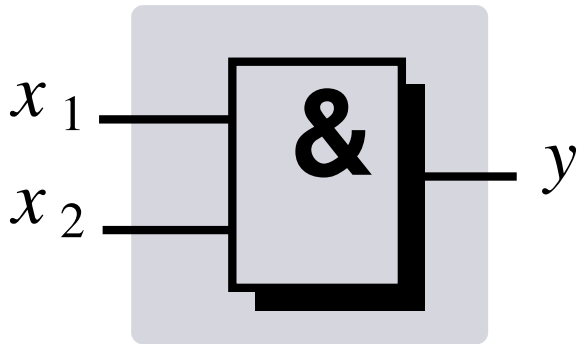
x	y
false	true
true	false



- Boolesche Notation: $y = \overline{x}$
- Java-Notation: `y = !x`
- Bei $x \in \{0, 1\}$ arithmetische Entsprechung: $y = 1 - x$

UND-Verknüpfung (AND, Konjunktion)

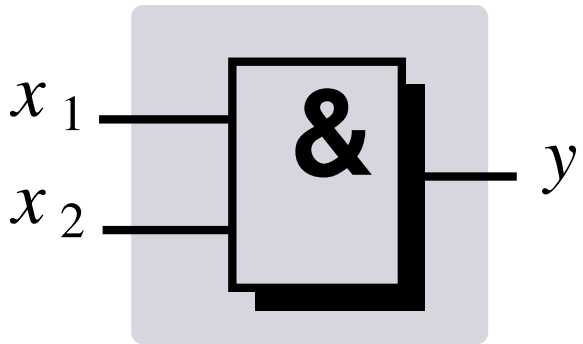
x_1	x_2	y
false	false	false
false	true	false
true	false	false
true	true	true



- Boolesche Notation: $y = x_1 \wedge x_2$ bzw. kurz: $y = x_1 x_2$
- Java-Notation: `y = x1 && x2`
- Bei $x_1, x_2, y \in \{0, 1\}$ arithmetische Entsprechung: $y = x_1 \cdot x_2$
- Genau dann, wenn beide Werte `x1` und `x2` jeweils `true` sind, wird als Ergebnis `true` geliefert, ansonsten `false`

UND-Verknüpfung (AND, Konjunktion)

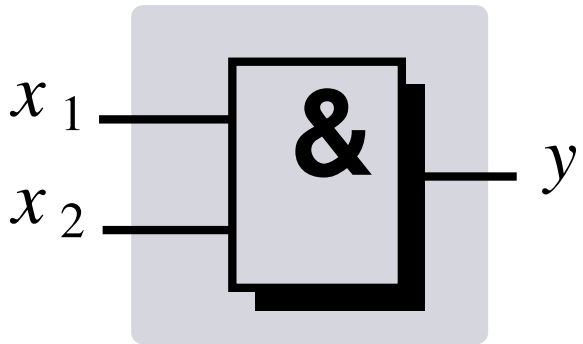
x_1	x_2	y
false	false	false
false	true	false
true	false	false
true	true	true



- Boolesche Notation: $y = x_1 \wedge x_2$ bzw. kurz: $y = x_1 x_2$
- Java-Notation: `y = x1 && x2`
- Bei $x_1, x_2, y \in \{0, 1\}$ arithmetische Entsprechung: $y = x_1 \cdot x_2$
- Genau dann, wenn beide Werte `x1` und `x2` jeweils `true` sind, wird als Ergebnis `true` geliefert, ansonsten `false`

UND-Verknüpfung (AND, Konjunktion)

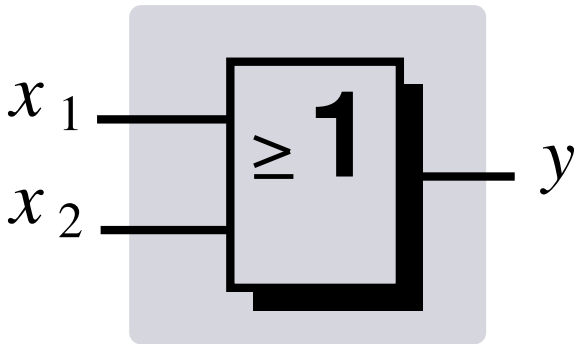
x_1	x_2	y
false	false	false
false	true	false
true	false	false
true	true	true



- Boolesche Notation: $y = x_1 \wedge x_2$ bzw. kurz: $y = x_1 x_2$
- Java-Notation: `y = x1 && x2`
- Bei $x_1, x_2, y \in \{0, 1\}$ arithmetische Entsprechung: $y = x_1 \cdot x_2$
- Genau dann, wenn beide Werte `x1` und `x2` jeweils `true` sind, wird als Ergebnis `true` geliefert, ansonsten `false`

ODER-Verknüpfung (OR, Disjunktion)

x_1	x_2	y
false	false	false
false	true	true
true	false	true
true	true	true

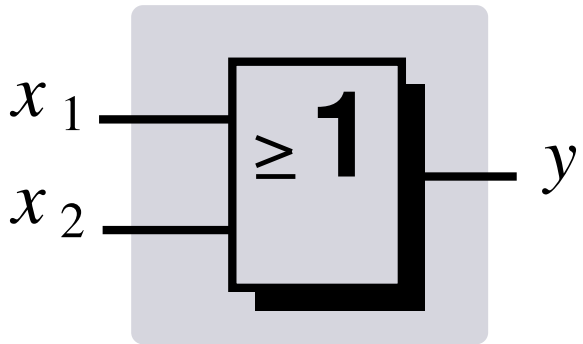


- Boolesche Notation: $y = x_1 \vee x_2$
- Java-Notation: `y = x1 || x2`
- Bei $x_1, x_2, y \in \{0, 1\}$ arithmetische Entsprechung:

  $y = x_1 + x_2 - x_1 \cdot x_2$
- Genau dann, wenn mindestens einer der beiden Werte `x1` und `x2` `true` ist, wird als Ergebnis `true` geliefert, ansonsten `false`

ODER-Verknüpfung (OR, Disjunktion)

x_1	x_2	y
false	false	false
false	true	true
true	false	true
true	true	true

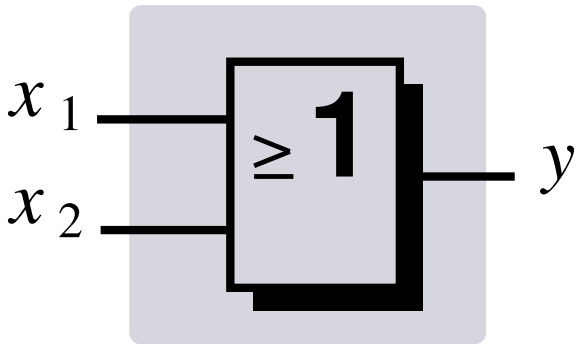


- Boolesche Notation: $y = x_1 \vee x_2$
- Java-Notation: `y = x1 || x2`
- Bei $x_1, x_2, y \in \{0, 1\}$ arithmetische Entsprechung:

  $y = x_1 + x_2 - x_1 \cdot x_2$
- Genau dann, wenn mindestens einer der beiden Werte `x1` und `x2` `true` ist, wird als Ergebnis `true` geliefert, ansonsten `false`

ODER-Verknüpfung (OR, Disjunktion)

x_1	x_2	y
false	false	false
false	true	true
true	false	true
true	true	true



- Boolesche Notation: $y = x_1 \vee x_2$
- Java-Notation: `y = x1 || x2`
- Bei $x_1, x_2, y \in \{0, 1\}$ arithmetische Entsprechung:

  $y = x_1 + x_2 - x_1 \cdot x_2$
- Genau dann, wenn mindestens einer der beiden Werte `x1` und `x2` `true` ist, wird als Ergebnis `true` geliefert, ansonsten `false`

Priorisierung von Negation, UND- und ODER-Verknüpfung

Negation

x	1–x
0	1
1	0

```
!x
```

UND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

```
x && y
```

ODER

x	y	x+y–xy
0	0	0
0	1	1
1	0	1
1	1	1

```
x || y
```

Prioritaet

Bedingungen dürfen Boolesche Operatoren enthalten

```
boolean esRegnet = true;
boolean habeSchirm = false;
int t = 20; //Temperatur in Grad Celsius

if ((esRegnet) && (!habeSchirm) && (t > 0))
{
  System.out.println("Ich werde nass.");
}
```

- Merke: Vergleiche `==`, `!=`, `<`, `<=`, `>`, `>=` liefern bei wahr den Booleschen Wert `true`, sonst `false`
- Dies kann unmittelbar zur kompakten Notation logischer Ausdrücke genutzt werden, z.B. `boolean x = (a > b);`
- Bedingungen bitte stets sauber klammern und den Überblick über die Klammerungen behalten

Rechengesetze der Aussagenlogik

$$x \vee \overline{x} = \text{true}$$
$$x \ \overline{x} = \text{false}$$

Rechengesetze der Aussagenlogik

$$
\begin{aligned}
x \vee \overline{x} &= \text{true} \\
x \ \overline{x} &= \text{false} \\
\\
x \vee x &= x \\
x \ x &= x
\end{aligned}
$$

Rechengesetze der Aussagenlogik

$$x \vee \overline{x} = \text{true}$$
$$x \ \overline{x} = \text{false}$$

$$x \vee x = x$$
$$x \ x = x$$

$$\overline{\overline{x}} = x \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{ doppelte Negation}$$

Rechengesetze der Aussagenlogik

$$x \vee \overline{x} = \text{true}$$
$$x \ \overline{x} = \text{false}$$

$$x \vee x = x$$
$$x \ x = x$$

$$\overline{\overline{x}} = x \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{ doppelte Negation}$$

$$\overline{x \vee y} = \overline{x} \ \overline{y} \quad \ldots\ldots\ldots\ldots \text{ De Morgansche Regel}$$
$$\overline{x \ y} = \overline{x} \vee \overline{y} \quad \ldots\ldots\ldots\ldots \text{ De Morgansche Regel}$$

Rechengesetze der Aussagenlogik

$$x \vee \overline{x} = \text{true}$$
$$x \ \overline{x} = \text{false}$$

$$x \vee x = x$$
$$x \ x = x$$

$$\overline{\overline{x}} = x \quad \ldots\ldots\ldots\ldots\ldots \text{doppelte Negation}$$

$$\overline{x \vee y} = \overline{x} \ \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$
$$\overline{x \ y} = \overline{x} \vee \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$

$$x \vee y = y \vee x \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$
$$x \ y = y \ x \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$

Rechengesetze der Aussagenlogik

$$x \vee \overline{x} = \text{true}$$
$$x \ \overline{x} = \text{false}$$

$$x \vee x = x$$
$$x \ x = x$$

$$\overline{\overline{x}} = x \quad \ldots\ldots\ldots\ldots\ldots \text{doppelte Negation}$$

$$\overline{x \vee y} = \overline{x} \ \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$
$$\overline{x \ y} = \overline{x} \vee \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$

$$x \vee y = y \vee x \quad \ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$
$$x \ y = y \ x \quad \ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$

$$a \vee (b \vee c) = (a \vee b) \vee c \quad \ldots\ldots\ldots\ldots\ldots \text{Assoziativität}$$
$$a \ (b \ c) = (a \ b) \ c \quad \ldots\ldots\ldots\ldots\ldots \text{Assoziativität}$$

Rechengesetze der Aussagenlogik

$$x \vee \overline{x} = \text{true}$$
$$x \ \overline{x} = \text{false}$$

$$x \vee x = x$$
$$x \ x = x$$

$$\overline{\overline{x}} = x \quad \ldots\ldots\ldots\ldots\ldots \text{doppelte Negation}$$

$$\overline{x \vee y} = \overline{x} \ \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$
$$\overline{x \ y} = \overline{x} \vee \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$

$$x \vee y = y \vee x \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$
$$x \ y = y \ x \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$

$$a \vee (b \vee c) = (a \vee b) \vee c \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Assoziativität}$$
$$a \ (b \ c) = (a \ b) \ c \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Assoziativität}$$

$$a \ (b \vee c) = a \ b \vee a \ c \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Distributivität}$$

Rechengesetze der Aussagenlogik

$$x \vee \overline{x} = \text{true}$$
$$x \ \overline{x} = \text{false}$$

$$x \vee x = x$$
$$x \ x = x$$

$$\overline{\overline{x}} = x \quad \ldots\ldots\ldots\ldots\ldots \text{doppelte Negation}$$

$$\overline{x \vee y} = \overline{x} \ \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$
$$\overline{x \ y} = \overline{x} \vee \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$

$$x \vee y = y \vee x \quad \ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$
$$x \ y = y \ x \quad \ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$

$$a \vee (b \vee c) = (a \vee b) \vee c \quad \ldots\ldots\ldots\ldots\ldots \text{Assoziativität}$$
$$a \ (b \ c) = (a \ b) \ c \quad \ldots\ldots\ldots\ldots\ldots \text{Assoziativität}$$

$$a \ (b \vee c) = a \ b \vee a \ c \quad \ldots\ldots\ldots\ldots\ldots \text{Distributivität}$$
$$a \vee (b \ c) = (a \vee b) \ (a \vee c) \quad \ldots\ldots\ldots\ldots \text{Distributivität}$$

Rechengesetze der Aussagenlogik

$$x \vee \overline{x} = \text{true}$$
$$x \ \overline{x} = \text{false}$$

$$x \vee x = x$$
$$x \ x = x$$

$$\overline{\overline{x}} = x \quad \ldots\ldots\ldots\ldots\ldots \text{doppelte Negation}$$

$$\overline{x \vee y} = \overline{x} \ \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$
$$\overline{x \ y} = \overline{x} \vee \overline{y} \quad \ldots\ldots\ldots\ldots \text{De Morgansche Regel}$$

$$x \vee y = y \vee x \quad \ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$
$$x \ y = y \ x \quad \ldots\ldots\ldots\ldots\ldots \text{Kommutativität}$$

$$a \vee (b \vee c) = (a \vee b) \vee c \quad \ldots\ldots\ldots\ldots\ldots \text{Assoziativität}$$
$$a \ (b \ c) = (a \ b) \ c \quad \ldots\ldots\ldots\ldots\ldots \text{Assoziativität}$$

$$a \ (b \vee c) = a \ b \vee a \ c \quad \ldots\ldots\ldots\ldots\ldots \text{Distributivität}$$
$$a \vee (b \ c) = (a \vee b) \ (a \vee c) \quad \ldots\ldots\ldots\ldots \text{Distributivität}$$

$\Longrightarrow$ Nachprüfbar durch Ausfüllen der Belegungstabellen

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Der findige Wohnheimleiter

„Der Wohnheimleiter musste einige Studenten wegen einer größeren Reparatur im Wohnheim umquartieren. *Vier von acht* umzusiedelnden Studenten sollten gemeinsam in ein anderes Zimmer ziehen. Auf die Frage, wer mit wem wohnen wolle, kamen folgende Antworten:

- Andrej ist mit beliebigen Mitbewohnern einverstanden.
- *Boris* zieht ohne Kostja nicht um.
- *Kostja* will nicht in einem Zimmer mit Wassili wohnen.
- Wassili ist mit jedem einverstanden.
- *Dima* will nicht ohne Jura umziehen.
- *Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen, ohne Dima wird er nicht in einem Zimmer mit Kostja wohnen.
- *Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen, und außerdem will er nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.
- *Jura* zieht nur um mit Boris oder Fedja. Außerdem will er nicht in einem Zimmer mit Kostja wohnen, wenn Grischa nicht mitzieht, und möchte auch weder mit Andrej noch mit Wassili zusammen wohnen.

Der Wohnheimleiter konnte alle Wünsche berücksichtigen. Wie?“

Alle $2^8 = 256$ Kombinationen erzeugen und testen

```
public class Wohnheim {
  public static void main(String[] args) {
    boolean a, b, k, w, d, j, f, g;

    for (byte a_ = 0; a_ <= 1; a_++) {
      a = (a_ == 1); //Andrej
      for (byte b_ = 0; b_ <= 1; b_++) {
        b = (b_ == 1); //Boris
        for (byte k_ = 0; k_ <= 1; k_++) {
          k = (k_ == 1); //Kostja
          for (byte w_ = 0; w_ <= 1; w_++) {
            w = (w_ == 1); //Wassili
            for (byte d_ = 0; d_ <= 1; d_++) {
              d = (d_ == 1); //Dima
              for (byte j_ = 0; j_ <= 1; j_++) {
                j = (j_ == 1); //Jura
                for (byte f_ = 0; f_ <= 1; f_++) {
                  f = (f_ == 1); //Fedja
                  for (byte g_ = 0; g_ <= 1; g_++) {
                    g = (g_ == 1); //Grischa

                    /* Testen der Bedingungen */

                  } //for g_
                } //for f_
              } //for j_
            } //for d_
          } //for w_
        } //for k_
      } //for b_
    } //for a_
  } //main
} //class
```

Boris zieht ohne Kostja nicht um.

b	k	
false	false	true
false	true	false
true	false	false
true	true	true

$$b\ k \vee \overline{b}\ \overline{k}$$

Boris zieht ohne Kostja nicht um.

b	k	
false	false	true
false	true	false
true	false	false
true	true	true

$$b \; k \vee \overline{b} \; \overline{k}$$

Kostja will nicht in einem Zimmer mit Wassili wohnen.

k	w	
false	false	true
false	true	true
true	false	true
true	true	false

$$\overline{k \; w}$$

***Dima* will nicht ohne Jura umziehen.**

d	j	
false	false	true
false	true	false
true	false	false
true	true	true

$$d \; j \vee \overline{d} \; \overline{j}$$

***Dima* will nicht ohne Jura umziehen.**

d	j	
false	false	true
false	true	false
true	false	false
true	true	true

$$d \;\; j \vee \overline{d} \;\; \overline{j}$$

***Fedja* wird nicht ohne Grischa in einem Zimmer zusammen mit Dima wohnen.**

f	g	d	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	true
true	false	true	false
true	true	false	true
true	true	true	true

$$\overline{f \;\; \overline{g} \;\; d}$$

***Fedja* wird ohne Dima nicht in einem Zimmer mit Kostja wohnen.**

f	d	k	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	true
true	false	true	false
true	true	false	true
true	true	true	true

$$\overline{f \; \overline{d} \; k}$$

***Fedja* wird ohne Dima nicht in einem Zimmer mit Kostja wohnen.**

f	d	k	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	true
true	false	true	false
true	true	false	true
true	true	true	true

$$\overline{f \; \overline{d} \; k}$$

***Grischa* will nicht, dass seine Mitbewohner Boris und Kostja ein Zimmer teilen.**

b	k	
false	false	true
false	true	true
true	false	true
true	true	false

$$\overline{b \; k}$$

***Grischa* will nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.**

g	a	w	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	true
true	false	true	false
true	true	false	false
true	true	true	false

$$\overline{g} \vee g\ \overline{a}\ \overline{w}$$

Grischa will nicht in einem Zimmer mit Andrej oder mit Wassili wohnen.

g	a	w	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	true
true	false	true	false
true	true	false	false
true	true	true	false

$$\overline{g} \vee g\ \overline{a}\ \overline{w}$$

Jura zieht nur um mit Boris oder Fedja.

j	b	f	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	false
true	false	true	true
true	true	false	true
true	true	true	true

$$\overline{j\ \overline{b}\ \overline{f}}$$

***Jura* zieht nicht in ein Zimmer mit Kostja, wenn Grischa nicht mitzieht.**

j	k	g	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	true
true	false	true	true
true	true	false	false
true	true	true	true

$$\overline{j \; k \; \overline{g}}$$

***Jura* zieht nicht in ein Zimmer mit Kostja, wenn Grischa nicht mitzieht.**

j	k	g	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	true
true	false	true	true
true	true	false	false
true	true	true	true

$$\overline{j\ k\ \overline{g}}$$

***Jura* möchte weder mit Andrej noch mit Wassili zusammen wohnen.**

j	a	w	
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	true
true	false	true	false
true	true	false	false
true	true	true	false

$$\overline{j} \vee j\ \overline{a}\ \overline{w}$$

Alle Bedingungen müssen erfüllt sein (Wohnheim.java)

```
/* Teste Bedingungen */

if ((a_ + b_ + k_ + w_ + d_ + j_ + f_ + g_ == 4) &&  //vier aus acht
    ((b && k) || (!b && !k)) && //Boris nicht ohne Kostja
    (!(k && w)) && //Kostja nicht mit Wassili
    ((d && j) || (!d && !j)) && //Dima nicht ohne Jura
    (!(f && !g && d)) && //Fedja nicht ohne Grischa zusammen mit Dima
    (!(f && !d && k)) && //Fedja ohne Dima nicht gemeinsam mit Kostja
    (!(b && k)) && //Boris und Kostja nicht gemeinsam
    (!g || (g && !a && !w)) && //Grischa nicht mit Andrej oder Wassili
    (!(j && !b && !f)) && //Jura nur mit Boris oder Fedja
    (!(j && k && !g)) && //Jura nicht mit Kostja, wenn Grischa nicht mitzieht
    (!j || (j && !a && !w))) //Jura weder mit Andrej noch mit Wassili
{
  System.out.println("Zimmerbelegung:");
  if (a) {System.out.printf("Andrej  ");}
  if (b) {System.out.printf("Boris  ");}
  if (k) {System.out.printf("Kostja  ");}
  if (w) {System.out.printf("Wassili  ");}
  if (d) {System.out.printf("Dima  ");}
  if (j) {System.out.printf("Jura  ");}
  if (f) {System.out.printf("Fedja  ");}
  if (g) {System.out.printf("Grischa  ");}
  System.out.println();
}
```

Alle Bedingungen müssen erfüllt sein (Wohnheim.java)

```
/* Teste Bedingungen */

if ((a_ + b_ + k_ + w_ + d_ + j_ + f_ + g_ == 4) &&  //vier aus acht
    ((b && k) || (!b && !k)) && //Boris nicht ohne Kostja
    (!(k && w)) && //Kostja nicht mit Wassili
    ((d && j) || (!d && !j)) && //Dima nicht ohne Jura
    (!(f && !g && d)) && //Fedja nicht ohne Grischa zusammen mit Dima
    (!(f && !d && k)) && //Fedja ohne Dima nicht gemeinsam mit Kostja
    (!(b && k)) && //Boris und Kostja nicht gemeinsam
    (!g || (g && !a && !w)) && //Grischa nicht mit Andrej oder Wassili
    (!(j && !b && !f)) && //Jura nur mit Boris oder Fedja
    (!(j && k && !g)) && //Jura nicht mit Kostja, wenn Grischa nicht mitzieht
    (!j || (j && !a && !w))) //Jura weder mit Andrej noch mit Wassili
{
  System.out.println("Zimmerbelegung:");
  if (a) {System.out.printf("Andrej  ");}
  if (b) {System.out.printf("Boris  ");}
  if (k) {System.out.printf("Kostja  ");}
  if (w) {System.out.printf("Wassili  ");}
  if (d) {System.out.printf("Dima  ");}
  if (j) {System.out.printf("Jura  ");}
  if (f) {System.out.printf("Fedja  ");}
  if (g) {System.out.printf("Grischa  ");}
  System.out.println();
}
```

```
Zimmerbelegung:
Dima  Jura  Fedja  Grischa
```