# Informatik

Grundwissen

Klett

# Grundwissen – Tabellenkalkulationssysteme

**Inhalte von Zellen**

Die Zellen von Rechenblättern können konstante Werte (Daten) oder Formeln enthalten. Werte können mithilfe von Zellbezügen als Eingabe für Formeln verwendet werden. Eine Formel berechnet zu jedem Satz von Eingabewerten jeweils genau einen Ausgabewert.

| B2 | Formel | =SUMME(A1:B1) | | |
|---|---|---|---|---|
| | A | B | C | D |
| 1 | 3 | 4 | | |
| 2 | | 7 | | |
| 3 | | | | |

**Anpassung von Zellbezügen**

Beim Kopieren einer Formel in eine andere Zelle des Rechenblattes werden Zellbezüge in der Regel automatisch an die neue Lage angepasst. Will man dies verhindern, so verwendet man absolute Zellbezüge (gekennzeichet durch $). Beim Ausschneiden und anschließenden Einfügen einer Formel in eine andere Zelle werden die Zellbezüge dagegen nicht geändert.

| | A | B |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | ausschneiden – | =SUMME(A9:A10) |
| 7 | einfügen | |
| 8 | | |
| 9 | 3 | |
| 10 | 4 | |
| 11 | =SUMME(A9:A10) | |
| 12 | | |
| 13 | kopieren – | |
| 14 | einfügen | =SUMME(B12:B13) |
| 15 | | |
| 16 | | |
| 17 | | |

**Datentypen**

Als konstante Werte werden in den Zellen eines Rechenblattes genau genommen nur Texte oder Zahlen gespeichert. Über die Formatierung einer Zelle kann man ihren Zahlenwert in einer Vielzahl verschiedener Datenformate darstellen, z. B. als Datum, Uhrzeit, Bruch oder Wahrheitswert.

| C2 | Formel | =A2 | |
|---|---|---|---|
| | A | B | C |
| 1 | **Zahlenwert** | **formatiert als** | **ergibt:** |
| 2 | 12,1234 | Datum | 12.01.1900 |
| 3 | | Uhrzeit | 02:57:42 |
| 4 | | Bruch | 12 1/8 |
| 5 | | | |



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
**ISBN:** 978-3-12-731068-9

# Grundwissen – Funktionen

**Zuordnungen und Funktionen**
Zuordnungen werden durch Zuordnungsvorschriften (Tabellen, Diagramme, Aussageformen) beschrieben. Eine Zuordnung heißt Funktion, wenn jedem Element der Ausgangsmenge höchstens ein Element der Zielmenge zugeordnet wird.

Tabellenkalkulationssysteme enthalten viele fertig eingebaute Funktionen.

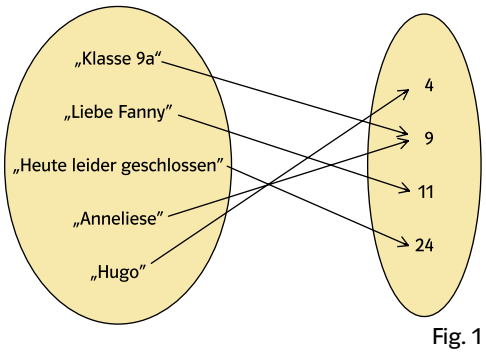


Fig. 1

*Funktionen eines Tabellenkalkulationssystems:*
*LÄNGE (Fig. 1); RÖMISCH*

**Mehrstellige Funktionen**
In der Informatik trifft man oft auf Funktionen mit zwei oder noch mehr Argumenten.
Manche Funktionen kommen sogar mit einer beliebigen Anzahl von Argumenten zurecht.
Bei zweistelligen Funktionen unterscheidet man zwischen Präfix- und Infixschreibweise.

*mehrstellige Funktionen:*
*RUNDEN(1,3342;2); FINDEN(„Peter“;A2;0)*

*Funktionen mit einer beliebigen Anzahl von Argumenten:*
*MITTELWERT; SUMME*

*Präfixschreibweise:* *SUMME(3;4)*
*Infixschreibweise:* *3 + 4*

**Verkettung von Funktionen**
Bei der Verkettung zweier Funktionen übernimmt eine Funktion den Wert einer anderen Funktion als Argument.
Verkettete Funktionen kann man sehr übersichtlich in Datenflussdiagrammen darstellen. Will man hierbei ein Zwischenergebnis mehrfach verwenden, kann man dies durch Einbau eines Datenverteilers erreichen.

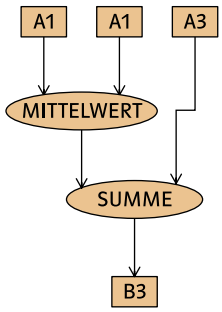


Fig. 2

*SUMME(MITTELWERT(A1;A2);A3)*

**WENN-Funktion**
In Termen, die je nach Wert einer bestimmten Bedingung unterschiedliche Berechnungsverfahren erfordern, verwendet man die WENN-Funktion.

*WENN(A1>A2;A1−A2;A2−A1)*

**Datenflussmodellierung**
In allgemeinen Datenflussdiagrammen können auch Prozesse vorkommen, die nicht durch eine Funktion beschrieben werden können (z. B. solche mit mehreren Ausgängen, die zu unterschiedlichen Zeiten Daten liefern). Die Symbolik von Datenflussdiagrammen kann auch für Verarbeitungs- und Transportvorgänge von materiellen Dingen (wie Autoersatzteilen) eingesetzt werden.



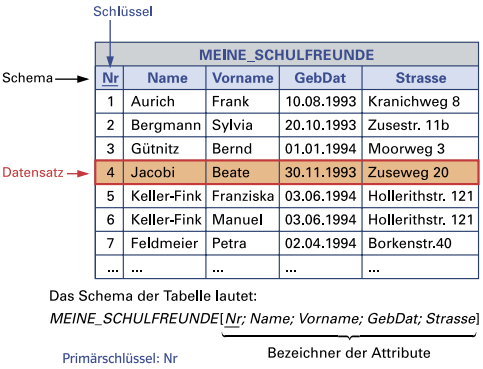
**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Grundwissen – Tabellen in Datenbanken

**Tabellen**
Daten werden in Tabellen gespeichert.
Die Spalten der Tabelle stehen für die Attribute der zugrunde liegenden Klasse und bilden das Schema der Tabelle.
Jede Zeile der Tabelle steht für ein einzelnes Objekt, enthält dessen Attributwerte und steht somit für einen Datensatz.

Der Primärschlüssel der Tabelle besteht aus einem Attribut oder mehreren Attributen (Spalten), durch deren Werte jeder Datensatz eindeutig identifiziert werden kann. In der Praxis werden ausschließlich künstliche Schlüssel verwendet.




*Die Tabelle MEINE_SCHULFREUNDE hat die* Attribute *Nr, Name, Vorname, GebDat und Strasse.*
*Jeder Datensatz steht für ein Objekt dieser Klasse, also einen Schulfreund.*

**Abfragen**
Mithilfe von Abfragen lassen sich Informationen aus Tabellen filtern.
Eine Abfrage ist eine Funktion mit den Eingabeparametern *Tabelle, Bedingung, Spaltenliste.*
Abfrage(Tabelle; Bedingung; Spaltenliste) liefert die gewünschten Spalten derjenigen Datensätze der angesprochenen Tabelle, die die übergebene Bedingung erfüllen.

Jede Abfrage ist eine Kombination einer Selektion, welche Datensätze einer Tabelle mittels einer Bedingung auswählt, mit einer Projektion, die nur gewünschte Spalten einer Tabelle wiedergibt.

In der Sprache SQL lautet der entsprechende Befehl

```
SELECT Spalten
FROM TABELLE
WHERE Bedingung
```

*Abfrage(MEINE_SCHULFREUNDE;* Name=*'Keller-Fink'; [*Vorname, Strasse*]) liefert*

| Vorname | Strasse |
|---|---|
| Franziska | Hollerithstraße 121 |
| Manuel | Hollerithstraße 121 |

*In SQL lautet diese Abfrage*

```
SELECT Vorname, Strasse
FROM MEINE_SCHULFREUNDE
WHERE Name = 'Keller-Fink'
```

*Die Abfrage ist eine Verkettung von der Selektion derjenigen Datensätze, deren Name 'Keller-Fink' lautet, und der Projektion auf die Spalten* Vorname *und* Strasse.
*Projektion(Selektion(MEINE_SCHULFREUNDE;* Name=*'Keller-Fink'); [*Vorname, Name*])*

| SCHUELER | | | |
|---|---|---|---|
| Name | Vorname | Klasse | Raum |
| Aurich | Frank | 9a | 121 |
| Bergmann | Sylvia | 9c | 133 |
| Gütnitz | Bernd | 9c | 133 |
| Jacobi | Beate | 9a | 121 |
| Keller-Fink | Franziska | 9c | 133 |
| Keller-Fink | Manuel | 9b | 243 |
| Feldmeier | Petra | 9b | 243 |
| ... | ... | ... | ... |

**Redundanz und Konsistenz**
In manchen Tabellen wird die gleiche Information mehrfach gespeichert, man spricht hier von redundanten Daten. Diese Redundanz von Daten kann bei Änderungen zu Mehrdeutigkeiten und Fehlern (Anomalien) in der Datenbank führen, sie ist in sich nicht mehr stimmig (inkonsistent).

*Die Nummern der Klassenzimmer sind mehrfach (redundant) gespeichert. Muss eine Klasse beispielsweise wegen Umbauarbeiten umziehen, müssen alle Datensätze passend geändert werden, ansonsten ist die Datenbank inkonsistent.*



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

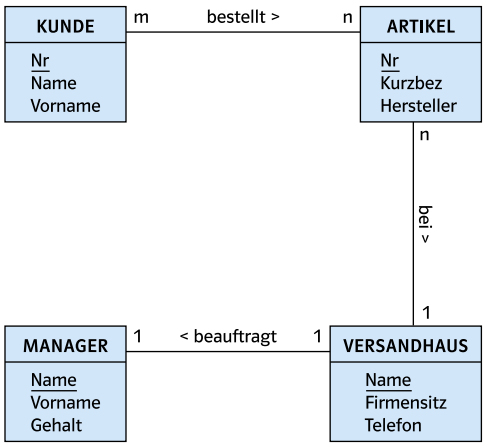
# Grundwissen – Datenbankentwurf

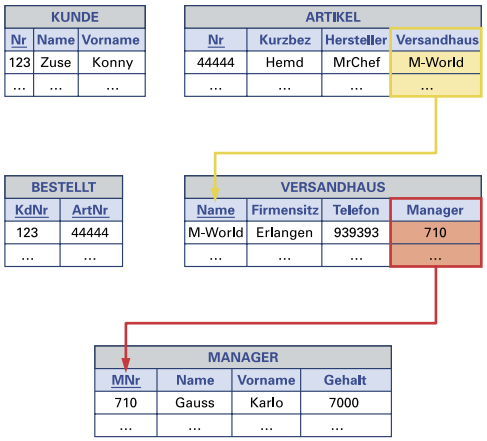
**Datenmodellierung**
Beim Modellieren betrachtet man nur einen kleinen Ausschnitt aus der realen Welt, die sogenannte „Miniwelt“. Welche Informationen beim Datenmodell wichtig sind, muss vorher geklärt werden.

Man verwendet das „Klassen-Beziehungs-Modell“, welches alle benötigten Klassen mit ihren Attributen und eventuell zugehörigen Datentypen umfasst. Beziehungen zwischen den betrachteten Klassen vervollständigen zusammen mit den zugehörigen Kardinalitäten das Klassendiagramm.

**Relationales Datenbankmodell**
Das Klassendiagramm kann systematisch in ein relationales Datenbankmodell umgewandelt werden. Dieses besteht aus mehreren Tabellen, die folgendermaßen aus dem Klassen-Beziehungs-Modell hervorgehen:
Jede Klasse im Klassen-Beziehungs-Modell wird in eine eigene Tabelle umgesetzt und gegebenenfalls um einen Primärschlüssel ergänzt.
Jede n:m-Beziehung wird zu einer Tabelle, welche die Primärschlüssel beider Tabellen als Fremdschlüssel enthält und deren Kombination als eigenen Primärschlüssel festlegt.
Jede n:1-Beziehung wird aufgelöst, indem die Tabelle derjenigen Klasse auf der Seite mit Kardinalität n um den Primärschlüssel des Beziehungspartners erweitert wird.
Jede 1:1-Beziehung wird analog der n:1-Beziehung umgewandelt, wobei es hier keine Rolle spielt, welche der beiden zugehörigen Klassen um den Primärschlüssel der anderen erweitert wird.

**Integritätsbedingungen**
Sobald eine Tabelle um ein neues Attribut erweitert wurde, welches auf den Primärschlüssel einer anderen Tabelle verweist, spricht man von einem Fremdschlüssel. Hierbei muss die referentielle Integrität gewährleistet sein: Jeder Fremdschlüsselwert muss als Primärschlüsselwert in der referenzierten Tabelle vorkommen.
Eine weitere strukturelle Integritätsregel fordert die Existenz und Eindeutigkeit eines Primärschlüssels.
Außerdem gibt es noch anwendungsspezifische Integritätsbedingungen, beispielsweise durch Vorgaben in der „Miniwelt“ oder Einschränkung des Wertebereichs auf bestimmte Attributwerte.


*Versandhaus ist Fremdschlüssel in ARTIKEL und verweist auf den Primärschlüssel von VERSANDHAUS.*
*Manager ist Fremdschlüssel in VERSANDHAUS und verweist auf den Primärschlüssel von MANAGER. Hier könnte auch als neues Attribut „von“ der Primärschlüssel von VERSANDHAUS als Fremdschlüssel in MANAGER ergänzt werden.*



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

**Join aus mehreren Tabellen**
Daten aus zwei oder mehreren Tabellen, die über Fremdschlüssel miteinander in Beziehung stehen, können durch einen Join gewonnen werden.

Die Funktion Join kann man als Hintereinanderausführung von Kreuzprodukt und Selektion verstehen. Die Bedingung muss sicherstellen, dass nur die Datensätze des Kreuzprodukts ausgewählt werden, bei denen Fremdschlüsselwert und zugehöriger Primärschlüsselwert übereinstimmen.

In SQL lautet die Abfrage:

```
SELECT *
FROM TABELLE1, TABELLE2
WHERE TABELLE2.Fremdschlüssel=TABELLE1.Primärschlüssel
```

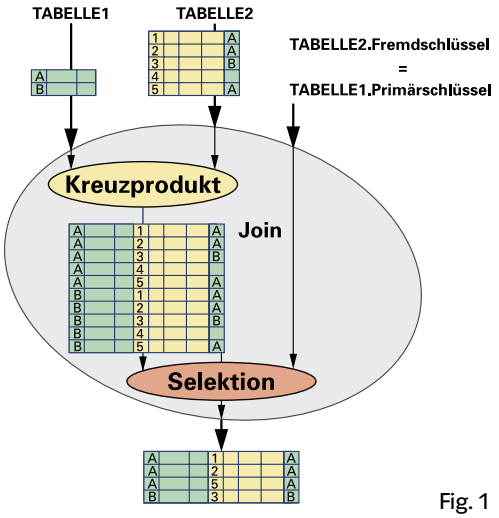



Fig. 1

**Tabellen erweitern**
Die Funktion *Erweiterung(TABELLE1; Spaltenname; Rechenterm)* gibt eine Tabelle mit zusätzlicher Spalte mit einem neuen Spaltennamen aus. Sie enthält die aus dem Rechenterm berechneten Ergebniswerte.

**Aggregatfunktionen auf gruppierten Daten**
Die Funktion *Gruppierung(TABELLE1; Spalte3)* fasst alle Datensätze der Tabelle TABELLE1 zusammen, die in *Spalte3* denselben Wert haben. Aggregatfunktionen verarbeiten gruppierte Werte einer Spalte, wie den Mittelwert (AVG), die Anzahl (COUNT), den größten Wert (MAX) oder den kleinsten Wert (MIN) der Eingabeargumente.

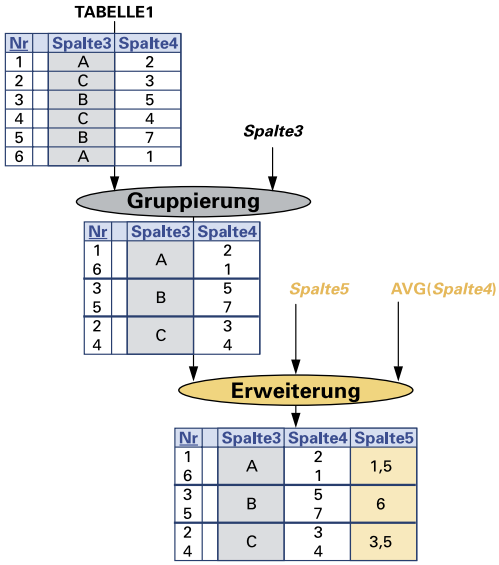



Fig. 2

*Die Abfrage in SQL zur Ausgabe der Durchschnittswerte (Fig. 2) lautet beispielsweise:*

```
SELECT AVG(Spalte4) AS Spalte5
FROM TABELLE1
GROUP BY Spalte3
```

*Sollen in TABELLE1 (Fig. 2) die Werte in* Spalte4 *verdoppelt werden, die in* Spalte3 *den Buchstaben C eingetragen haben, kann dies durch die SQL-Anweisung*

```
UPDATE TABELLE1
SET Spalte4=2*Spalte4
WHERE Spalte3='C'
```

*erreicht werden.*

**Datenmanipulation und Datenschutz**
Die Sprache SQL kann zum automatischen Einfügen (INSERT), Löschen (DELETE) bzw. Ändern (UPDATE) von Datensätzen eingesetzt werden.
Zur Vermeidung unerwünschter Manipulationen werden in Datenbanksystemen Zugriffsrechte für Benutzer festgelegt. Es können z. B. Schreib- und Leserechte für Ausschnitte des Datenmodells (Views) vergeben werden.
Der Schutz vor Datenmissbrauch wird durch den Datenschutz gesetzlich geregelt.



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Grundwissen – Objekte und Zustände

**Objekte und Klassen**
Nicht nur elektronische Dokumente enthalten eine Vielzahl von Objekten, die ganze Welt ist aus verschiedenen Objekten aufgebaut. Jedes Objekt wird dabei durch seine Eigenschaften (Attribute) mit ihren aktuellen Attributwerten und seinen möglichen Methoden beschrieben. Jedes Objekt gehört zu einer Klasse.

Eine Klasse ist ein Konstruktionsplan für Objekte. Gehören zwei Objekte zu derselben Klasse, so besitzen sie dieselben Attribute und Methoden, können sich jedoch in ihren Attributwerten unterscheiden. Das Objekt ist also konkret, die Klasse abstrakt.

Auf Befehl können Objekte verschiedene Methoden ausführen. Oft müssen dazu ein oder mehrere Parameter mit übergeben werden.

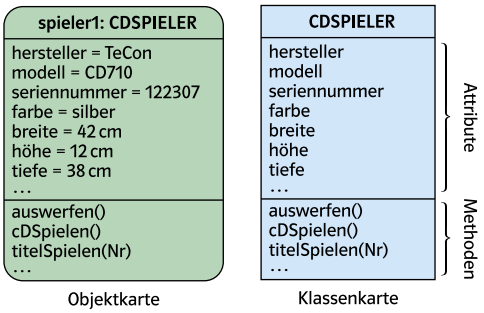
Beziehungen zwischen verschiedenen Objekten lassen sich im Objektdiagramm veranschaulichen. Da dieses bei mehreren beteiligten Objekten schnell unübersichtlich wird, verwendet man oft ein Klassendiagramm, welches die Beziehungen aller Objekte einer Klasse zu allen Objekten anderer Klassen darstellt.

**Algorithmus**
Eine Verarbeitungsvorschrift, die aus einer endlichen Anzahl von eindeutig formulierten Befehlen besteht, sodass diese tatsächlich ausführbar sind, nennt man Algorithmus. Die einzelnen Operationen können Methodenaufrufe sein, die eventuell wieder selbst als Algorithmus dargestellt werden können.

Algorithmen können in natürlicher Sprache oder auch in einer Programmiersprache abgefasst werden.

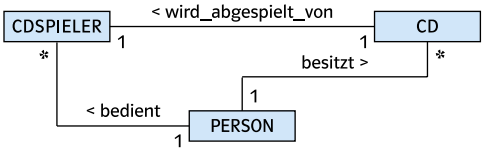
Elementare Strukturelemente von Algorithmen sind Sequenzen, bedingte Verarbeitungsschritte (Fallunterscheidungen) und Wiederholungen (durch eine Bedingung gesteuert oder mit vorher festgelegter Anzahl von Durchläufen).

**Zustände und Zustandsdiagramme**
Während seiner Lebensdauer kann ein Objekt verschiedene Zustände annehmen. Diese werden durch seine aktuellen Attributwerte beschrieben. Aktionen können Übergänge von einem Zustand in einen anderen Zustand auslösen. Mithilfe von Zustandsdiagrammen lässt sich das Verhalten vieler Objekte beschreiben. Solche Objekte nennt man auch Zustandsautomaten.




*Ein weiterer CD-Spieler des gleichen Modells hat beispielsweise nur eine andere Seriennummer und ansonsten die gleichen Attributwerte wie* spieler1.

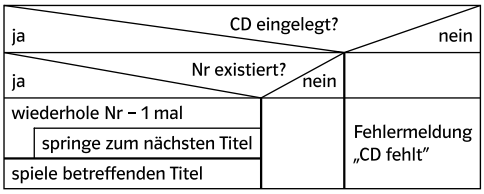
*Der Methodenaufruf* spieler1.titelSpielen(4) *veranlasst den CD-Player* spieler1, *das vierte Lied auf der CD abzuspielen.*

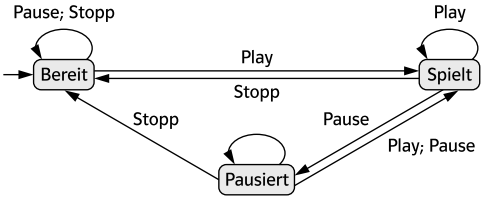



*Methode* titelSpielen(nr):

```
Wenn CD eingelegt, dann
   wenn nr existiert auf CD, dann
      wiederhole nr-1 mal
         springe zum nächsten Titel
      Ende wiederhole
      spiele betreffenden Titel
   Ende wenn
sonst
   melde „CD fehlt"
Ende wenn
```

**titelSpielen(int Nr)**






**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

### Definition von Klassen

Jede Klasse muss mit einem eindeutigen Bezeichner versehen werden, z. B. die Klasse RECHTECK, welche rechts im Beispiel die Struktur für achsenparallele Rechtecke vorgibt. Sie beinhaltet:

- die **Deklaration von Attributen** mit jeweiliger Angabe des Zugriffsmodifikators, des Datentyps und Attributbezeichners, z. B. die Attribute für die Eckpunkte des Rechtecks und die Farbe;
- die **Definition von Methoden** mit jeweiliger Angabe des Zugriffsmodifikators, des Typs des Rückgabewertes, des Methodenbezeichners und in Klammern die Angabe eventueller Eingabeparameter mit jeweiligem Typ (Methodenkopf). Beispiele sind die Methode *verschieben* (ohne Rückgabewert und mit zwei Eingabeparametern) bzw. die Methode *breite*, die als Ergebnis eine Zahl vom Typ `int` zurückgibt.
  Der Methodenrumpf beschreibt den Algorithmus, der nach dem Aufruf der Methode ausgeführt wird. Dabei können innerhalb der Methodendefinition ebenso auch Methodenaufrufe stehen, z. B. werden in der Methode *umfang* die Methoden *breite* und *länge* aufgerufen.
  Der **Konstruktor** ist eine spezielle Methode ohne Rückgabewert, der bei der Erzeugung neuer Objekte aufgerufen wird. Konstruktoren haben denselben Bezeichner wie die Klasse.

### Wertzuweisung

Wertzuweisungen dienen der direkten Änderung eines Attribut- oder Variablenwertes. Der bisherige Wert geht dabei verloren. Sie haben in *Java* die Form *Bezeichner = Term*.

### Anlegen und Löschen von Objekten

Zur Laufzeit eines Programms werden Programmobjekte angelegt. Dies geschieht mithilfe von Konstruktormethoden, die z. B. sinnvolle Anfangswerte für die Attribute setzen. Im Beispiel werden im Konstruktor *Rechteck* die Anfangswerte für die Koordinaten der Eckpunkte P(0, 0) und Q(10, 10) und die Farbe *rot* gesetzt. Das Löschen von Objekten geschieht z. B. in *Java* automatisch.

### Implementieren von Algorithmen

Jede Programmiersprache stellt Kontrollstrukturen zur Implementierung der algorithmischen Strukturelemente zur Verfügung: Sequenz, Wiederholung mit fester Anzahl (z. B. in der Methode *dinA_Setzen*) bzw. bedingte Wiederholung, bedingte Anweisung und Alternative (z. B. in der Methode *istQuadrat*).

### Felder

Ein Feld fasst Elemente mit dem gleichen Datentyp zusammen. Über einen Index kann auf jedes Feldelement zugegriffen werden. In der Methode *dinA_Setzen* werden z. B. die ersten k Standardlängen bzw. -breiten für die Papierformate in mm berechnet und die Rechteckmaße in das DIN-A-k-Format gesetzt.

Java

```
public class Rechteck {

// Attribute
   private int px,py,qx,qy;
   private String farbe;
   int[] dinA_laenge;
   int[] dinA_breite;

// Methoden
   public Rechteck() {
      px = 0; py = 0;
      qx = 10; qy = 10;
      farbe = "rot";
   }

   public void farbeSetzen(
                String neueFarbe) {
      farbe = neueFarbe;
   }

   public void verschieben(
                  int dx, int dy) {
      px = px + dx; py = py + dy;
      qx = qx + dx; qy = qy + dy;
   }

   public int laenge() {
      return Math.abs(qx - px);
   }

   public int breite() {
      return Math.abs(qy - py);
   }

   public int umfang() {
      return 2*(breite() + laenge());
   }

   public int flaecheninhalt() {
      return laenge()*breite();
   }

   public void istQuadrat() {
      if (laenge() == breite()) {
        System.out.print("Quadrat");
      }
      else {
        System.out.print("Kein
                        Quadrat");
      }
   }

   public void dinA_Setzen(int k) {
     int[] dinA_laenge = new int[k+1];
     int[] dinA_breite = new int[k+1];
     dinA_laenge[0] = 1189;
     dinA_breite[0] = 841;
     for (int i = 1; i < k + 1; i++){
       dinA_laenge[i] =
                  dinA_breite[i-1];
       dinA_breite[i] =
                 dinA_laenge[i-1]/2;
     }
     qx = px + dinA_laenge[k];
     qy = py + dinA_breite[k];
   }
}
```

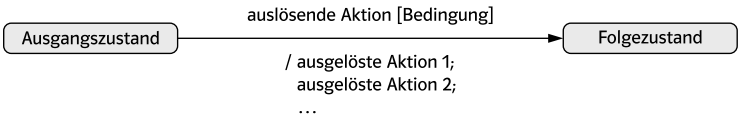


**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

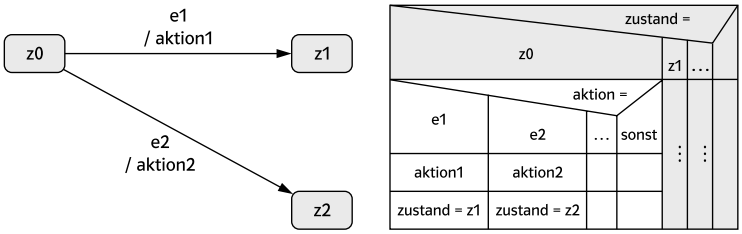
**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

**Endliche Automaten und allgemeine Zustandsdiagramme**

Endliche Automaten werden durch Zustandsdiagramme mit einer endlichen Anzahl von Zuständen beschrieben, an deren Übergängen neben auslösenden auch ausgelöste Aktionen stehen können. Jedoch können damit nicht alle mithilfe von Algorithmen simulierbaren Systeme beschrieben werden. Dazu müssen die Zustandsdiagramme noch um die Möglichkeit erweitert werden, für Übergänge eine Bedingung anzugeben, unter der sie stattfinden sollen.

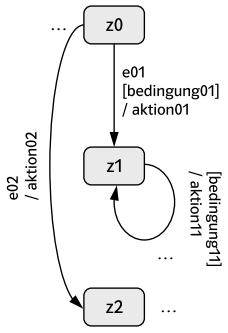



Die Zustandsübergänge implementiert man am besten mittels Fallunterscheidung über die jeweiligen Ausgangszustände, innerhalb derer jeweils weitere Fallunterscheidungen über die verschiedenen auslösenden Aktionen (z. B. Eingaben) geschachtelt werden. In den einzelnen Fällen dieser inneren Fallunterscheidung werden dann die Zustandswechsel sowie die ausgelösten Aktionen durch entsprechende Anweisungen vorgenommen.




Übergangsbedingungen werden implementiert, indem man für die entsprechenden Zustandsänderungen bedingte Anweisungen verwendet.

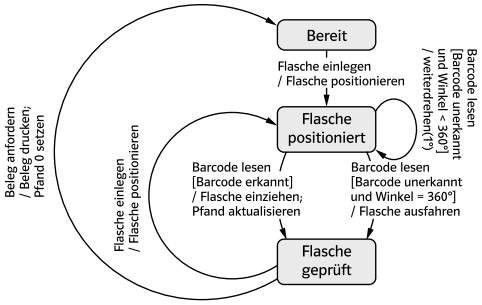
Zur Simulation einer Folge von Zustandswechseln umschließt eine Wiederholungsstruktur die mehrfache Fallunterscheidung über die Zustände (siehe oben).



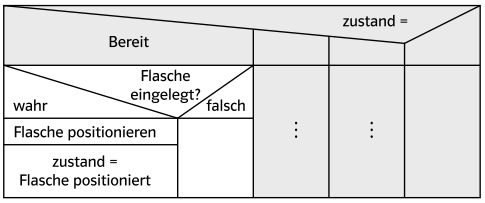

Java

```
switch (zustand) {
 case '0': { // Zustand z0
    switch (aktion) {
       case 1: { // Aktion e01
          if (bedingung01) {
             aktion01();
             zustand = 1;}
          } break; ...
    } // Ende switch (aktion)
 } break;
 case '1': { // Zustand z1
    while (bedingung11) {aktion11();}
 } break; ...}
...} // Ende switch (zustand)
```

*Bei einem einfachen Automaten zur Rückgabe von Pfandflaschen wird zunächst im Zustand* Bereit *die Leerflasche in eine Öffnung gelegt. Diese Aktion löst beim Übergang in den Zustand* Flasche positioniert *das Einfahren in die korrekte Leseposition aus.*




*Der folgende Ausschnitt des Struktogramms zeigt die Implementierung dieses Zustandswechsels.*




*Die Flasche wird dann im Zustand* Flasche positioniert *(codiert mit 'F') so lange gedreht, bis der Barcode erkannt wird.*

Java

```
switch (zustand) {...
   case 'F': {
      while (!barcodeErkannt() &&
                (winkel < 360)) {
         weiterdrehen(1);}
   } break; ...
}
```

*Je nachdem, ob der Barcode erkannt worden ist oder nicht, werden beim Übergang in den Zustand* Flasche geprüft *die Aktionen* Flasche einziehen, Pfand aktualisieren *bzw.* Flasche ausfahren *ausgelöst. Schließlich sorgt die auslösende Aktion* Beleg anfordern *u. a. für den Ausdruck des Belegs.*



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

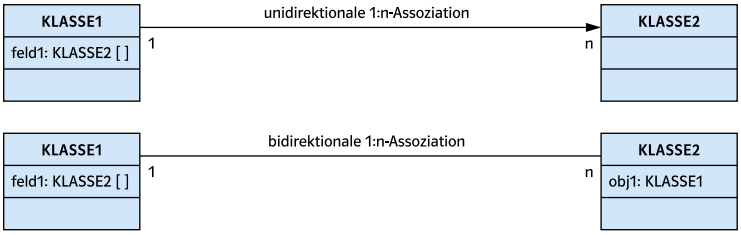
**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

**Klassenbeziehungen**
Beziehungen zwischen Klassen ermöglichen eine Interaktion zwischen den entsprechenden Objekten.
Klassenbeziehungen werden unterschieden nach
(1) ihrer Multiplizität: 1:1, 1:n bzw. n:m,
(2) ihrer Richtung: unidirektional, bidirektional.

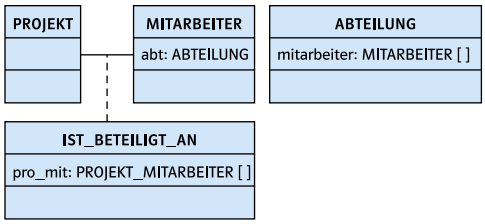
*Das folgende Klassendiagramm zeigt eine einfache Situation in einer Firma.*


**Implementierung von Klassenbeziehungen**
In einer Klasse, die eine 1:1-Beziehung zu einer anderen Klasse herstellt, legt man ein Attribut oder eine Variable für eine Referenz auf ein Objekt der anderen Klasse fest.
Ist die Beziehung bidirektional, so erhält jede Klasse ein Attribut für die Speicherung der Referenz des Objektes der anderen Klasse.

Bei einer 1:n-Beziehung wird in der Klasse auf der Seite mit Multiplizität 1 ein Feld aus Referenzen auf Objekte der anderen Klasse deklariert.



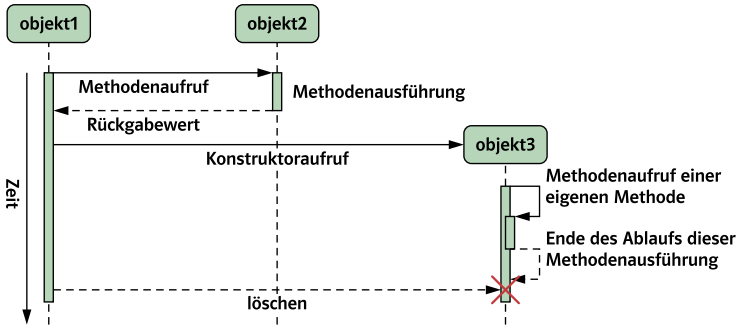

*Diese Beziehungen kann man z. B. mit den in den folgenden Klassenkarten eingetragenen Attributen implementieren. Die n:m-Beziehung* ist_beteiligt_an *zwischen PROJEKT und MITARBEITER wird dabei über eine Assoziationsklasse realisiert.*

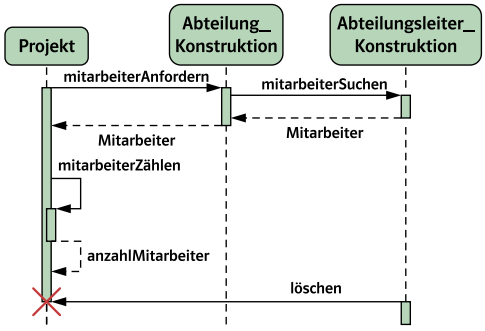



Eine Möglichkeit, n:m-Beziehungen zu realisieren, bietet die Definition einer eigenen Assoziationsklasse, in der ein Feld definiert wird, dessen Objekte je ein Attribut für die Referenzierung eines der an einer Beziehung beteiligten Objekte enthalten.

**Sequenzdiagramm**
Sequenzdiagramme dienen der Darstellung von Interaktionen zwischen Objekten. Jedoch werden nur bestimmte Ausschnitte des Systemablaufs beschrieben.

*Das Sequenzdiagramm zeigt den zeitlichen Ablauf für die Zuordnung eines Mitarbeiters der Konstruktionsabteilung zu einem Projekt und das Löschen des Projektes nach Fertigstellung.*





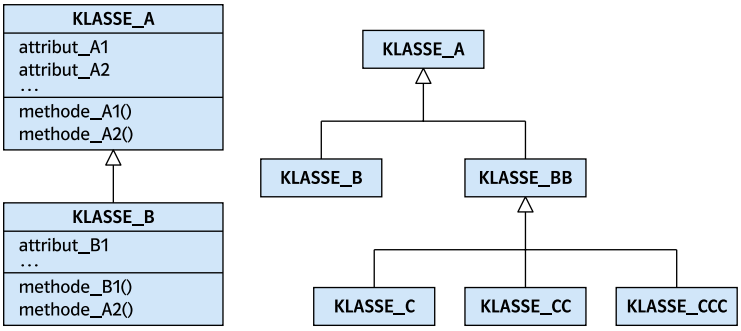

**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Grundwissen – Generalisierung

**Ober- und Unterklassen**
Klassen mit gemeinsamen Merkmalen können (als Unterklassen) zu einer allgemeineren, d.h. abstrakteren Klasse (Oberklasse) generalisiert werden. Die Unterklassen stellen dabei Spezialisierungen der Oberklasse dar.
Die Unterklassen erben alle in der Oberklasse deklarierten Attribute und Methoden. Soll eine Methode einer Oberklasse in einer Unterklasse anders implementiert werden, muss sie dort neu definiert, also überschrieben werden.
Durch fortgeführte Generalisierung bzw. Spezialisierung kann die Klassenbeziehung zu einer baumartigen Klassenhierarchie ausgebaut werden.

Nachfolgend ist KLASSE_A Oberklasse von KLASSE_B. Die Objekte von KLASSE_B erben alle Attribute und Methoden von KLASSE_A. *methode_A2()* wird dann jedoch in KLASSE_B überschrieben.

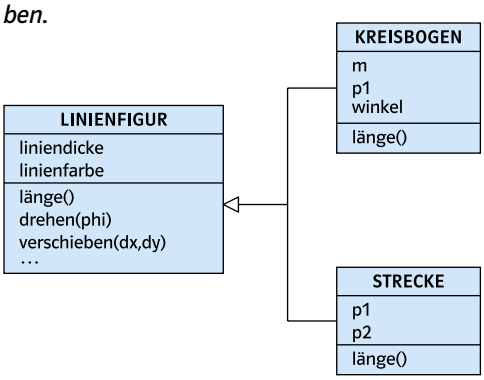



**Polymorphe Methoden**
Wenn eine Methode einer Oberklasse in ihren verschiedenen Unterklassen unterschiedlich überschrieben wird, sieht der Aufruf dieser Methoden zwar identisch aus, die Ausführung ist jedoch jeweils verschieden. Unten ist dies bei der Methode *methode_A2()* der Fall, die in KLASSE_B bzw. KLASSE_BB jeweils (eventuell unterschiedlich) überschrieben wird. Beim Aufruf über *objB. methode_A2()* bzw. *objBB.methode_A2()* wird dann die jeweilige Implementierung in KLASSE_B bzw. KLASSE_BB gestartet, was eventuell auch zu unterschiedlichen Ergebnissen führen kann.

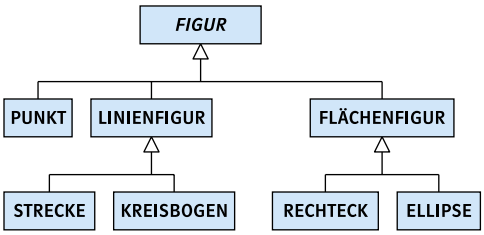
Java

```
public class Klasse_B extends Klasse_A {
   public void methode_A2() {...} // Überschreibung
}
public class Klasse_BB extends Klasse_A {
   public void methode_A2() {...} // Überschreibung
}
public class Test {
   objB = new Klasse_B; objBB = new Klasse_BB;
   public void testMethode() {
      objB.methode_A2(); objBB.methode_A2();
   }
}
```

*Kreisbögen und Strecken gehören zu den Linienfiguren. Gemeinsame Attribute sind Liniendicke und Linienfarbe. Die Methode* länge *ist in beiden Unterklassen überschrieben.*




*Rechtecke und Ellipsen sind Flächenfiguren. Diese gehören neben Linienfiguren und Punkten allgemein zur Klasse FIGUR.*




*Der Flächeninhalt einer aus verschiedenen Flächenfiguren zusammengesetzten Figurengruppe lässt sich beispielsweise über einen gemeinsamen Methodenaufruf* `flaeche()` *und der anschließenden Summenbildung berechnen.*

Java

```
public class Figurengruppe {
   Figur[] Fig = new Figur[10];
// ...
public Figurengruppe() {
   Fig[0] = new Rechteck();
   Fig[1] = new Ellipse();
   // ...
   public double gesamtflaeche() {
      double erg = 0;
      for (int i = 0; i < 10; i++) {
         erg = erg + Fig[i].flaeche();
      }
      return erg;
   }
}
```



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Grundwissen – Listen

## Felder

Bei Feldern ist die maximale Anzahl von aufzunehmenden Objekten von Anfang an festgelegt (Fig. 1). Dafür ist ein direktes Ansprechen eines bestimmten Elementes über einen Index möglich.

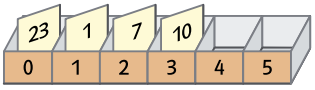



Fig. 1

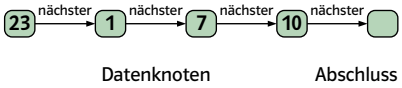



Fig. 2

## Listen

Einfach verkettete Listen entstehen durch Aneinanderreihen von Objekten, die Knoten genannt werden. Sie können jederzeit um neue Knoten erweitert werden. Jeder Knoten enthält ein Attribut für eine Referenz auf einen weiteren Knoten, nämlich den jeweils nächsten (Fig. 2). Man spricht daher von einer rekursiven Klassendefinition. Für die Implementierung von Listen verwendet man häufig als Entwurfsmuster das Kompositum (hervorgehobener Teilbereich in Fig. 3). Hierbei unterscheidet man zwei verschiedene Arten von Knoten: Datenknoten (mit Inhalt und genau einem Listenelement als Nachfolger) und Abschluss (ohne Nachfolger und meist ohne Inhalt). Jede verkettete Liste enthält beliebig viele Datenknoten und am Ende einen Abschluss.

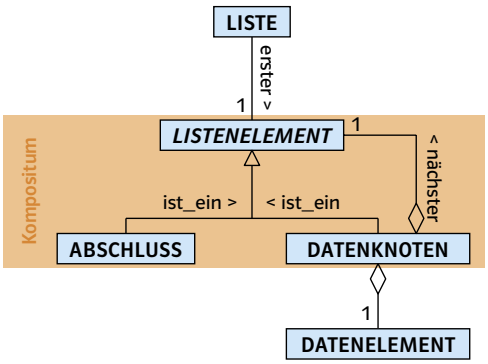



Fig. 3

## Warteschlangen und Stapel

Man unterscheidet zwei wichtige Arten von Listen, je nachdem wie die Elemente hinzugefügt bzw. entfernt werden.
Bei einer Warteschlange wird dasjenige Element als nächstes entfernt, welches bereits am längsten in der Liste „gewartet“ hat (FIFO-Prinzip: First In First Out). Bei einem Stapel verlässt das zuletzt hinzugekommene Element zuerst die Liste (LIFO-Prinzip: Last In First Out).

## Rekursive Funktionen

Funktionen, die in ihrem Funktionsterm mindestens einmal den eigenen Funktionsbezeichner enthalten, heißen rekursiv. Ein Funktionsaufruf endet nur dann nach einer endlichen Anzahl von Berechnungsschritten, wenn die Abbruchbedingung erfüllt ist.

Java

```
class Liste {
   // einziges Attribut ist eine
   // Referenz auf den ersten Knoten
   private Listenelement erster;

   public Liste() {
      erster = new Abschluss();
   }

   public void anfuegen(int zahl) {
      erster = erster.add(zahl);
   }
   public int entnehmen() {
      int old = erster.datenGeben();
      erster = erster.nextGeben();
      return old;
   }
   public int allesAddieren {
      return erster.summeGeben();
   }
   ...
}

abstract class Listenelement {
   public abstract int datenGeben();
   public abstract int summeGeben();
   ...
   }
}

class Datenknoten extends Listen-
         element {
   // jeder Knoten hat einen Inhalt
   // und eine Referenz auf den nächsten
   private Listenelement next;
   private int inhalt;

   public Datenknoten(Listen-
         element n, int i) {
      next = n;
      inhalt = i;
   }

   public Datenknoten add(int i) {
      next = next.add(i);
      return this;
   }
   public int datenGeben {
      return inhalt;
   }
   public int summeGeben() {
      return inhalt + next.summeGeben();
   }
   ...
}

class Abschluss extends Listen-
         element {
   public Datenknoten add(int i) {
      return new Datenknoten(this, i)
   }
   public int summeGeben() {
      return 0;
   }
   ...
}
```



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

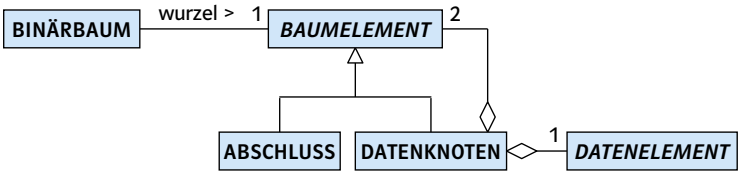
**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

**(Binär-)Bäume**
Kann ein Knoten einer (einfach) verketteten Liste mehrere Nachfolger haben, so entsteht ein Baum. Dabei wird genau ein Element innerhalb der Baumstruktur nicht referenziert. Dieses Element heißt **Wurzel**. Ausgehend von der Wurzel lässt sich jedes weitere Objekt der Baumstruktur durch schrittweises Folgen der Referenzen auf genau einem Weg erreichen.
Jeder Knoten hat genau einen **Elternknoten**, von dem er referenziert wird, kann aber selbst mehrere **Kindknoten** referenzieren.
Hat jeder Knoten höchstens zwei Kindknoten, so spricht man von einem Binärbaum.

**Traversieren eines Binärbaumes**
Ein Binärbaum kann auf drei verschiedenen Wegen durchlaufen (traversiert) werden:
Preorder: Knoten, linker Teilbaum, rechter Teilbaum,
Inorder: linker Teilbaum, Knoten, rechter Teilbaum,
Postorder: linker Teilbaum, rechter Teilbaum, Knoten.

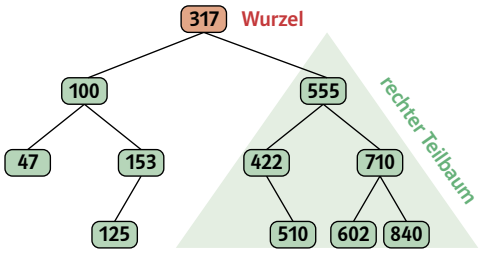
**Geordnete Binärbäume**
Ein Binärbaum ist geordnet bezüglich eines bestimmten Attributes (Schlüsselattribut), wenn für jeden Knoten
(1) jeder Schlüsselwert des linken Teilbaumes kleiner,
(2) jeder Schlüsselwert des rechten Teilbaumes größer
ist als der aktuelle Knoten.

**Implementierung eines Binärbaumes**
Ein Binärbaum wird vollkommen analog zur Liste implementiert mit dem Unterschied, dass es für jeden Knoten zwei Nachfolger gibt: „links“ und „rechts“.




**Einfügen eines Elementes in einen geordneten Binärbaum**
Das neue Datenelement wird so lange (rekursiv) weitergeleitet, bis ein Abschluss erreicht ist und dort an dessen Stelle eingefügt. Beim „Durchreichen“ wird das neue Element jeweils nach links oder rechts weitergegeben, je nachdem, ob der Schlüsselwert größer oder kleiner als der Schlüsselwert des aktuellen Knotens ist.
Die Reihenfolge des Einfügens entscheidet über die Struktur des Baumes.
Das Suchen nach einem Element erfolgt auf die gleiche Weise.

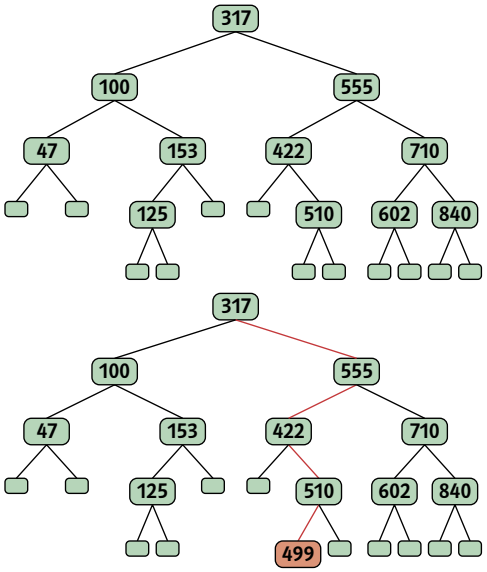
*Die Kundendaten eines Unternehmens werden in einer Baumstruktur gespeichert. Dabei sollen die abgebildeten Zahlen für die Kundennummern stehen.*




*Die Knoten mit den Schlüsseln 47 und 153 sind die Kindknoten vom Knoten mit dem Schlüssel 100, dieser hat wiederum die Wurzel (317) als Elternknoten.*
*Knoten 555 hat als linken Teilbaum die Knoten 422 und 510, wobei 422 die Wurzel dieses Teilbaumes ist.*

*Preorder-Durchlauf: 317 – 100 – 47 – 153 – 125 – 555 – 422 – 510 – 710 – 602 – 840*
*Inorder-Durchlauf: 47 – 100 – 125 – 153 – 317 – 422 – 510 – 555 – 602 – 710 – 840*
*Postorder-Durchlauf: 47 – 125 – 153 – 100 – 510 – 422 – 602 – 840 – 710 – 555 – 317*

*Es handelt sich um einen geordneten Binärbaum. Das Einfügen eines neuen Elementes mit der Schlüsselnummer 499 veranschaulichen folgende Grafiken.*






**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Grundwissen – Graphen

## Graphen

Ein Graph besteht aus einer Menge von Knoten und einer Menge von Kanten. Durch Kanten werden jeweils zwei Knoten miteinander verbunden; diese sind dann benachbart oder Nachbarknoten. Bei gerichteten Kanten unterscheidet man bei den Nachbarknoten zwischen Vorgänger- und Nachfolgerknoten. Bei bewerteten Graphen wird den Kanten ein Gewicht zugeordnet. Insbesondere sind Bäume und Listen spezielle Graphen.

## Wege durch Graphen

Eine Folge von aufeinanderfolgenden Kanten stellt einen Pfad dar. Gibt es einen Pfad von einem Knoten zu einem zweiten Knoten, so ist der zweite vom ersten aus erreichbar. In ungerichteten zusammenhängenden Graphen ist jeder Knoten von jedem anderen aus erreichbar. Ein Zyklus ist ein Pfad, der in dem Knoten endet, von dem aus er begonnen hat.

## Repräsentation von Graphen

Einen Graphen kann man mithilfe einer Adjazenzmatrix repräsentieren. Bei unbewerteten Graphen lauten die Einträge dabei „wahr“ oder „falsch“, je nachdem, ob eine Kante zwischen den entsprechenden Knoten vorhanden ist oder nicht. Im Falle von bewerteten Graphen werden die Kantengewichte in die Zellen der Matrix eingetragen.

## Suche in Graphen

Es gibt verschiedene Verfahren, alle Knoten eines Graphen nacheinander zu bearbeiten. Bei der Tiefensuche werden ausgehend von einem Startknoten s alle von s aus erreichbaren Knoten untersucht. Dabei wird zunächst in die Tiefe gegangen, d. h., es wird ein Nachbar- bzw. Nachfolgerknoten eines bereits erreichten Knotens weiter verfolgt, bevor die anderen Nachbarn bzw. Nachfolger dieses Knotens besucht werden. Bei der Umsetzung des Tiefensuche-Algorithmus sind die Knoten während der Bearbeitung geeignet zu markieren, um zu verhindern, dass man in einen Zyklus gerät. Startet man bei dem Graphen aus Fig. 1 mit dem Knoten A, so ist eine Tiefensuche-Abarbeitung der Knoten in der Reihenfolge A, C, B und D möglich (Fig. 2, 3 und 4). Der Knoten E wird nicht besucht, da er vom Startknoten A aus nicht erreichbar ist.

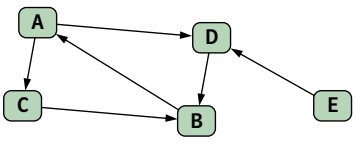


Fig. 1

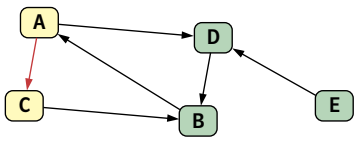


Fig. 2

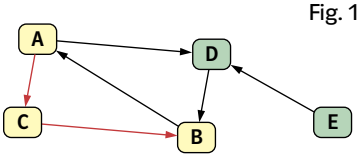


Fig. 3

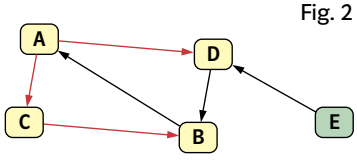
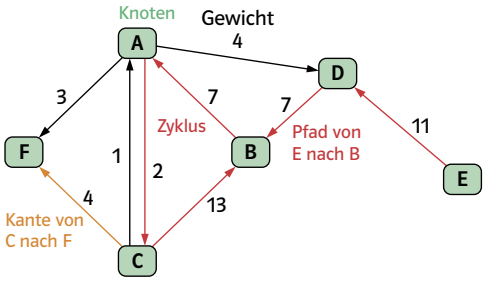


Fig. 4




*Die Abbildung zeigt einen bewerteten, gerichteten Graphen mit Zyklen. Dieser wird durch folgende Adjazenzmatrix repräsentiert:*

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | | | 2 | 4 | | 3 |
| B | 7 | | | | | |
| C | 1 | 13 | | | | 4 |
| D | | 7 | | | | |
| E | | | | 11 | | |
| F | | | | | | |

*Beispielsweise zeigt der Eintrag 7 in der Zelle der Zeile B und der Spalte A an, dass es von B nach A eine Kante mit dem Gewicht 7 gibt.*

**Java**

```
class Graph {
   private Knoten[] knotenliste;
   private int[][] adjazenzmatrix;
   private int anzahl;
   ...
   public void tiefensuche(int start-
            Index) {
      // Initialisierung
      tiefensucheKnoten(startIndex);
   }
   private void tiefensucheKnoten(int
            vIndex) {
      // Knoten mit vIndex als besucht
      // markieren
      ...
      for (int i = 0; i < anzahl; i++) {
         if (adjazenzmatrix[vIndex][i]
             &&!knotenliste[i].markie-
                  rungGeben()) {
            tiefensucheKnoten(i);
         } // Ende if
      } // Ende for
   }
}

class Knoten {
   private Datenelement inhalt;
   private boolean markierung;
   ...
}
```



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Grundwissen – Kooperative Arbeitsabläufe

**Projekte**
Projekte sind Vorhaben, deren klare, nachprüfbare Zielvorgabe in einem verbindlich festgelegten Zeitrahmen erreicht werden soll. Finanzielle, materielle und personelle Ressourcen sind begrenzt. Der Projektverlauf wird ständig überprüft, gesteuert und dokumentiert.

**Phasen eines Projektes**
Am Ende der Definitionsphase wird ein Projektauftrag mit Sachziel, Kostenziel und Terminziel formuliert sowie ein Pflichtenheft erstellt. Der Projektleiter bzw. die Projektleiterin wird ernannt, Arbeitsgruppen mit festen Zuständigkeiten werden eingeteilt und es wird ein Meilensteinplan erstellt.
Während der Planungsphase entsteht ein verfeinerter Projektstrukturplan. Der vermutliche Aufwand bezüglich Kosten, Zeit und Ressourcenbedarf wird abgeschätzt. Schließlich wird ein genauer Zeitplan erstellt.
In der Phase der Durchführung vergleicht der Projektleiter bzw. die Projektleiterin laufend den tatsächlichen Fortschritt mit der Planung und greift nötigenfalls steuernd ein. Die einzelnen Arbeiten werden von den Mitarbeitern in einem Projekttagebuch dokumentiert.
Zum Projektabschluss wird der Verlauf noch einmal rückblickend analysiert. Die Erfahrungen werden in einem Abschlussbericht festgehalten.

**Projektauftrag**

Projekttitel

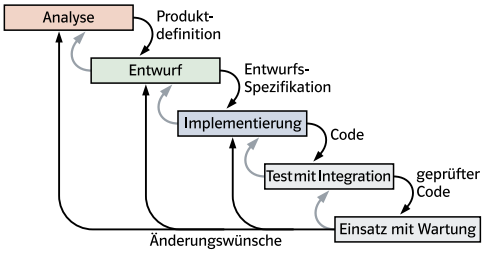
Projektgegenstand
1. Ausgangssituation
2. Ziele
3. Untersuchungsbereich
4. wichtige Rahmenbedingungen
5. Termine/Meilensteine
6. Projektaufbauorganisation

Projektleitung:
Projektteam:

Unterschriften der Auftragspartner

| | |
|---|---|
| für | für |
| Ort, Datum | Ort, Datum |
| Auftraggeber | Auftragnehmer |

**Wasserfallmodell**
Die Durchführung von Softwareprojekten unterteilt man nach dem Wasserfallmodell in folgende Phasen: Analyse, Entwurf, Implementierung, Test mit Integration sowie Einsatz mit Wartung. Diese Phasen werden nach diesem Modell in der genannten Reihenfolge und ohne Überschneidung durchlaufen. Nach jeder Phase kann jedoch wieder zu einer der vorausgehenden zurückgekehrt werden.




**Aufwandsschätzung**
Zur Schätzung des Zeit- und Kostenaufwands werden informelle Methoden wie Analogie, Schätzklausur, Aufwand pro Einheit und Prozentsatzmethode, aber auch formelbasierte Verfahren wie COCOMO herangezogen.

**Gegenseitige Abstimmung**
Werden Betriebsmittel von nebenläufigen Vorgängen gemeinsam genutzt, muss dies unter wechselseitigem Ausschluss erfolgen. Dieser kann durch Semaphore oder wie in *Java* durch Monitore gesichert werden. Eine Verklemmung (deadlock) entsteht, wenn nebenläufige Prozesse wechselweise Betriebsmittel belegen, auf die andere warten.

| SEMAPHOR |
|---|
| zähler<br>warteschlange |
| down()<br>up() |



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

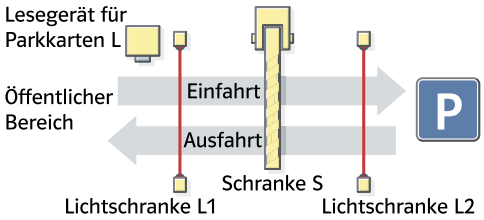
**Modellierungstechniken**
Für eine eindeutige und verständliche Beschreibung eines realen Systems verwendet man Modelle. Jedes Modell beinhaltet für seine Sichtweise auf das System die wesentlichen Eigenschaften. Funktionale und objektorientierte Modelle spiegeln die grundsätzlichen Strukturen des Systems wieder. In ihnen wird erfasst, wie die einzelnen Komponenten des Systems zusammenhängen. Mit funktionalen Modellen wird die Aufteilung des Systems in Teilsysteme beschrieben, die als Black Box aufgefasst werden. Objektorientierte Modelle erfassen den Zustand des Systems (Objektdiagramme) oder stellen die Abhängigkeiten zwischen den Komponenten dar (Klassendiagramme).
Will man den Ablauf oder die Kommunikation der einzelnen Komponenten darstellen, so verwendet man Zustands- oder Szenariomodelle. Diese zeigen das Zusammenspiel der einzelnen Komponenten zur Erledigung der Aufgaben.
Daneben werden Algorithmen als Struktogramme dargestellt.

**Softwaremuster**
Bei der Erstellung von Software versucht man aus Effizienzgründen, bewährte Softwarestrukturen wiederzuverwenden. Diese bezeichnet man als Entwurfsmuster.
Für die Erstellung hierarchischer Strukturen hat sich das Kompositum (Fig. 1) als günstig erwiesen.
Soll das Programm eine Benutzerschnittstelle in Form einer grafischen Oberfläche besitzen, so hat sich das Architekturmuster Model-View-Cotroller (MVC, Fig. 2) bewährt. Dieses trennt strikt das eigentliche Modell von der Anzeige (View) und der Steuerung (Controller). Zur Umsetzung in eine Programmiersprache steht dafür das Entwurfsmuster Beobachter zur Verfügung.

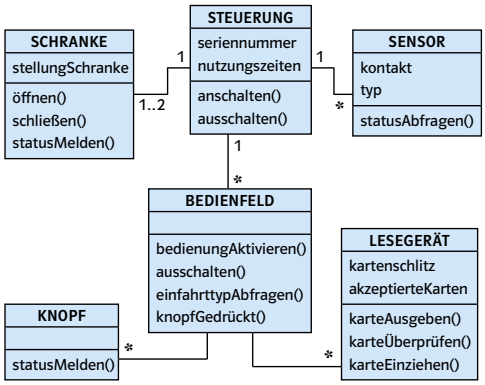
**Softwarequalität**
Bei der Erstellung von Software können Fehler passieren. Diese können sowohl im Modell als auch im Quellcode vorkommen. Will man diese Fehler aufspüren, so muss man das System validieren und verifizieren. Bei der Validierung wird überprüft, ob das richtige Produkt erstellt wird. Bei der Verifikation wird getestet, ob das Produkt richtig erstellt wird.
Für die Überprüfung stehen verschiedene Hilfsmittel zur Verfügung. Fehler im Modell wie auch im Programm lassen sich durch Softwareinspektion finden. Fehler im Quellcode werden meist durch Tests gefunden. Dabei wird das Programm mit Eingabedaten ausgeführt und das Ergebnis mit dem erwarteten Ergebnis verglichen. Dafür stehen auch automatisierte Testsysteme zur Verfügung, in *Java* zum Beispiel das *JUnit*-Framework.

**Dokumentation von Software**
Für die Benutzung einer Software bzw. eines Softwaresystems muss eine Beschreibung (Dokumentation) erstellt werden.

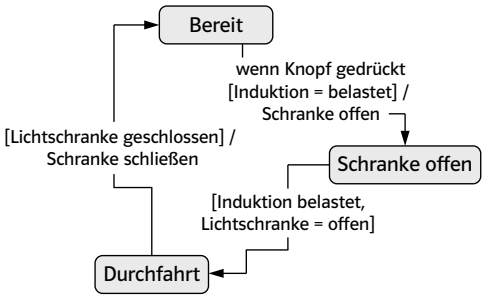
*Ausschnitt für die Modellierung eines Systems am Beispiel einer Schranke:*
*Hier ist die reale Situation zu sehen:*




*Das zugehörige Klassenmodell:*




*Ein mögliches Zustandsdiagramm:*




*Softwaremuster:*

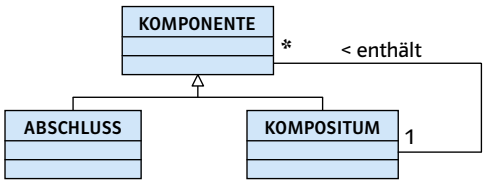



Fig. 1

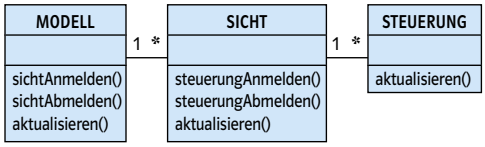



Fig. 2



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Rückblick – Formale Sprachen

**Formale Sprachen**
Wie natürlichen liegt auch formalen Sprachen ein Alphabet *Σ*, also eine endliche Menge von Zeichen, zugrunde. Jede beliebige Aneinanderreihung dieser Zeichen nennt man ein Wort. Eine Teilmenge aller derart erzeugten Wörter bildet eine formale Sprache *L*.

**Grammatik einer formalen Sprache**
Viele formale Sprachen können durch Angabe von Regeln zur Erzeugung aller zulässigen Wörter definiert werden. Dazu wird meist eine Grammatik angegeben. Diese umfasst
- eine endliche Menge *Σ* aus Terminalsymbolen, das Alphabet der Sprache,
- eine endliche Menge *V* aus Nichtterminalsymbolen als Hilfsvariablen für die Produktionsregeln (dabei darf kein Symbol zugleich zu *Σ* und zu *V* gehören),
- eine Startvariable *S*, die in der Menge *V* enthalten ist,
- eine Menge von Produktionsregeln der Form *w1 → w2*, wobei *w1* ein Element der Menge *V* und *w2* eine Kombination aus Nichtterminalen und/oder Terminalen ist.

Die von einer Grammatik *G* erzeugte formale Sprache *L(G)* besteht aus allen Wörtern, die aus der Startvariablen *S* unter Anwendung der Produktionen von *G* abgeleitet werden können und ausschließlich aus Terminalen bestehen. Die Reihenfolge der angewandten Regeln spielt dabei keine Rolle. Ein Wort kann unter Umständen mehrere Ableitungen haben.
Die Produktionsregeln einer Grammatik können auch durch Syntaxdiagramme oder in der (erweiterten) Backus-Naur-Form (EBNF) angegeben werden.

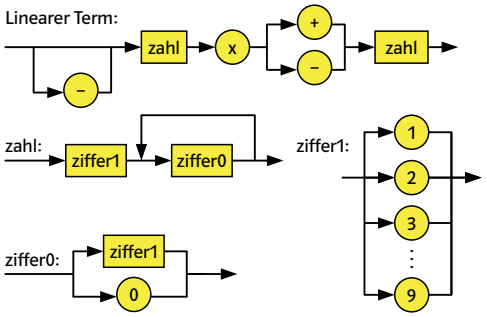
**Reguläre Sprachen und endliche Automaten**
Sprachen, bei denen für alle Regeln *w1 → w2* gilt, dass *w1* ein Nichtterminalsymbol und *w2* entweder ein Terminalzeichen oder ein Terminalzeichen gefolgt von einem Nichtterminalzeichen ist, heißen reguläre Sprachen. Diese können von einem endlichen Automaten dargestellt werden.
Ein deterministischer endlicher Automat (DEA) besteht aus
- einer endlichen Menge *Z* von Zuständen (abgerundetes Rechteck),
- einem endlichen Eingabealphabet *Σ*,
- einem Startzustand *z0 ∈ Z* (markiert durch einen Pfeil),
- einer Menge *E* von Endzuständen mit *E ⊂ Z* (doppelt umrandet),
- einer Übergangsfunktion *δ*, die jedem möglichen Paar aus Zustand und Eingabezeichen den Folgezustand zuordnet.

Befindet sich der Automat nach Abarbeitung sämtlicher Zeichen des Wortes in einem Endzustand, so ist dieses Wort vom Automaten akzeptiert (erkannt) und gehört zur entsprechenden regulären Sprache, andernfalls nicht. Endliche Automaten können oft mithilfe einer mehrfachen Fallunterscheidung implementiert werden.

*Mithilfe des Alphabetes Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, x, +, −} lassen sich beispielsweise folgende Wörter bilden:*
*'234+710−1', '7x−23', aber auch '4x3x−−+xx5'.*
*Eine mögliche Teilmenge* L *der durch diesen Zeichensatz darstellbaren Wörter ist die formale Sprache der Termdarstellung aller linearen Ausdrücke der Form mx+t.*

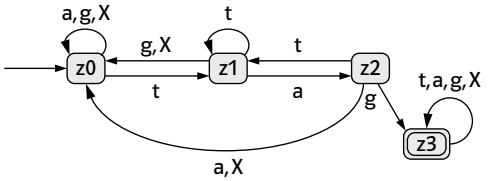
*Die Grammatik von* L *besteht aus* Σ, *den Nichtterminalen* {<L>, <z0>, <z1>, <PM>, <ziffer0>, <ziffer1>}, *dem Startsymbol* <L> *und folgenden Produktionsregeln:*
<L> → <z1>'x'<PM><z0>
<z0> → <ziffer0> | <ziffer1><z0>
<z1> → <ziffer1> | '−'<ziffer1> |
<ziffer1><z0> | '−'<ziffer1><z0>
<PM> → '+' | '−'
<ziffer0> → '0' | <ziffer1>
<ziffer1> → '1' | '2' | '3' | '4' | … | '9'

*Ein mögliches Syntaxdiagramm:*




*In EBNF:*
L = ['−'] zahl 'x' PM zahl;
zahl = ziffer0 {ziffer1};
PM = '+' | '−';
ziffer0 = '0' | ziffer1;
ziffer1 = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

*Folgender endlicher Automat erkennt, ob ein Wort die Zeichenfolge 'tag' enthält. X steht hierbei für alle anderen Zeichen.*






**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Rückblick – Kommunikation und Synchronisation von Prozessen

**Nebenläufige Prozesse**
Bestehen zwischen verschiedenen Arbeitsschritten (Prozessen) keine Abhängigkeiten, so können diese gleichzeitig (parallel) ausgeführt werden. Diese Prozesse werden dann als nebenläufig bezeichnet. Benutzen zwei oder mehrere nebenläufige Prozesse gemeinsame Ressourcen, so muss der Zugriff auf diese synchronisiert werden. Die einzelnen Arbeitsschritte, in denen ein Prozess auf die gemeinsame Ressource zugreift, werden als kritischer Abschnitt bezeichnet. Bei der Synchronisierung wird sichergestellt, dass diese Ressource immer sinnvoll verwendet wird und der Arbeitsschritt immer ein eindeutiges Resultat liefert.

**Monitore**
Für die Umsetzung der Synchronisierung von nebenläufigen Prozessen gibt es neben Semaphoren das Monitorkonzept. Ein Monitor besitzt sowohl innere Datenstrukturen als auch Methoden, über die nur ein Zugriff möglich ist. Zu jeder Zeit kann immer nur ein Prozess eine Methode eines Monitors aufrufen. Andere Prozesse müssen so lange warten, bis der Monitor wieder frei ist. Zur Verwaltung von Prozessen besitzt ein Monitor noch zwei weitere Methoden: zum Blockieren aktiver Prozesse und zum Aufwecken wartender Prozesse. Ein Blockieren eines aktiven Prozesses kann notwendig sein, wenn dieser aufgrund des inneren Zustands des Monitors seine Arbeit nicht fortsetzen kann und so auf die Arbeit eines anderen Prozesses warten muss.

**Protokolle und Netzwerkschichten**
Kommunikationsvorgänge zwischen Computersystemen erfolgen über Protokolle. Ein Protokoll regelt sowohl das Format der ausgetauschten Daten als auch den genauen Ablauf eines Kommunikationsvorgangs. Um die Komplexität für den gesamten Nachrichtenaustausch zu verringern, wird eine Kombination aus mehreren Protokollen verwendet. Die einzelnen Protokolle werden in einer hierarchischen Struktur (Schichten) angeordnet. Jedes Protokoll bietet der darüberliegenden Schicht einen speziellen Dienst an und verfügt dafür über eine festgelegte Schnittstelle. Bei einem Kommunikationsvorgang tauschen immer zwei Partner auf derselben Schicht Daten aus. Diese beiden Partner müssen nicht das eigentliche Ziel bzw. die eigentliche Quelle sein.

**Netzwerktopologien**
Die Art, wie Kommunikationsteilnehmer miteinander verbunden sind, wird als (Netzwerk-)Topologie bezeichnet. Mögliche Formen sind Busse, Ringe oder Sterne (Fig. 3).
Bei einem Bus hängen alle Teilnehmer an einem gemeinsamen Strang, auf den sie gemeinsam zugreifen. Der Ring unterscheidet sich hierzu dadurch, dass die Verbindungen zwischen einzelnen Teilnehmern verlaufen. Beim Stern sind die einzelnen Kommunikationspartner mit einer zentralen Vermittlungseinheit verbunden.

*Auf einem Computer können fast alle Programme Daten auf dem Drucker ausgeben. Die Verwaltung des Druckers übernimmt ein Druckerspooler (Fig. 1).*

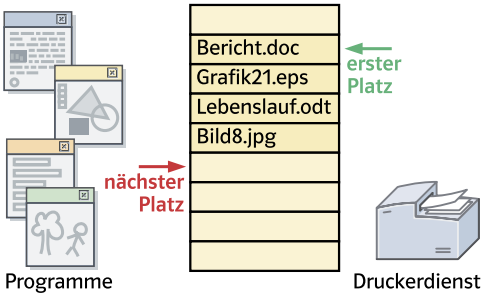


Fig. 1

*Fig. 2 zeigt einen möglichen Ablauf, bei dem Programme ihre Daten an den Druckerdienst übergeben und dieser die Dateien dann ausdruckt.*

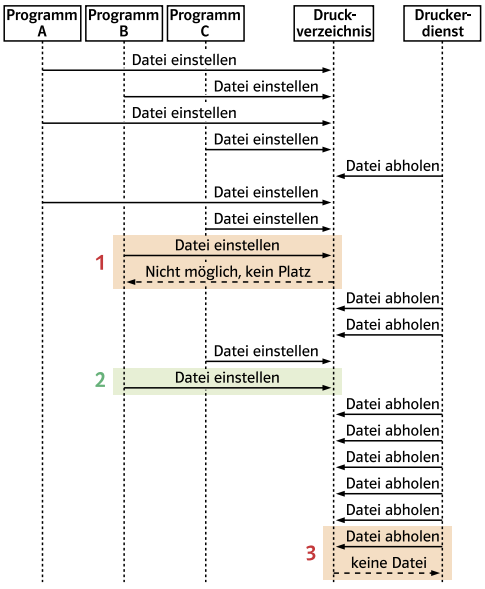


Fig. 2

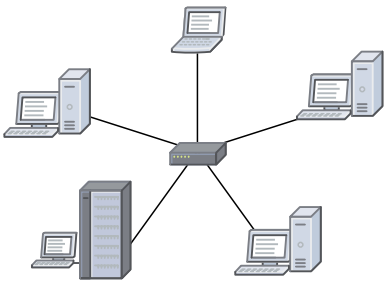


Fig. 3



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Rückblick – Funktionsweise eines Rechners

**Von-Neumann-Rechner**
Das theoretische Modell eines universellen, programmierbaren digitalen Rechners besteht aus Steuerwerk, Rechenwerk, Speicherwerk und Ein-/Ausgabewerk. Die Werke sind über ein getaktetes Bus-System miteinander verbunden. Daten und Befehle sind in binärer Form in den fortlaufend nummerierten Zellen des Speicherwerks abgelegt.

**Registermaschine**
Eine theoretische Registermaschine besteht aus einem Programm im Speicher, aus Registern als Speicherzellen für Daten sowie einem Befehlszähler, der die Speicheradresse des nächsten zu bearbeitenden Befehls enthält. Der Akkumulator ist ein spezielles Register zur Speicherung des letzten Berechnungsergebnisses.
Zur Programmierung verwendet man *Assembler*-Sprachen mit sehr kleinem Befehlsvorrat. Dieser umfasst

- Transportbefehle (`LOAD`, `DLOAD`, `STORE`),
- arithmetische Befehle (`ADD`, `SUB`, `MULT` …),
- Sprungbefehle mit und ohne Bedingung (`JGE`, `JUMP` …),
- logische Verknüpfungen (`AND`, `OR`, Negierung, Invertierung …).

**Reale Maschinen**
Reale Mikroprozessoren rechnen ebenfalls in Registern. In der Regel sind hierbei die Register der Operanden und des Ergebnisses frei wählbar.
Die gesamte Abarbeitung eines Maschinenbefehls (der Befehlszyklus) besteht dabei grob aus zwei Phasen: Befehl holen und Befehl ausführen.

- In der Befehlsholphase erfolgt der Speicherzugriff auf die vom Befehlszähler angezeigte Adresse. Der Befehl wird in das Befehlsregister des Prozessors geladen und der Befehlszähler aktualisiert.
- Der Befehl wird decodiert. Anschließend werden die notwendigen Operanden ermittelt und aus dem Speicher geladen. Der Befehl wird ausgeführt. Nötigenfalls werden Ergebnisse im Speicher abgelegt.

**Algorithmische Strukturelemente**
Algorithmen, die bedingte Anweisungen, Alternativen oder Wiederholungen enthalten, können allein mithilfe von Vergleichen und Sprungbefehlen implementiert werden.
Bei bedingten Anweisungen und Alternativen wird eine Sequenz übersprungen.
Bei Wiederholungen erfolgt ein Rücksprung zur erneuten Prüfung der Bedingung.

*Das folgende* Assembler*-Programm berechnet die dritte Potenz von 8.*

**Assembler**

```
1:  DLOAD 8        --Basis 8
2:  STORE 1        --Potenz p in R1
3:  STORE 2        --Basis b in R2
4:  DLOAD 1
5:  STORE 3        --Konstante 1 in R3
6:  DLOAD 3
7:  STORE 4        --Exponent e = 3 in R4
8:  JEQ 21         --falls e = 0
9:  SUB 3
10: JEQ 19         --falls e = 1
11: STORE 4        --e = e - 1
12: LOAD 1
13: MULT 2
14: STORE 1        --p = p * b
15: LOAD 4
16: SUB 3
17: STORE 4        --e = e - 1
18: JGT 12         --falls e > 0
19: LOAD 1         --Ergebnis in A
20: JUMP 22
21: LOAD 3         --1 in A für e = 0
22: END
```

*Die folgende Tabelle zeigt, wie der Akkumulator und die einzelnen Register belegt werden:*

| | A | R1 | R2 | R3 | R4 | BZ |
|---|---|---|---|---|---|---|
| | | | | | | 1 |
| 1: DLOAD 8 | 8 | | | | | 2 |
| 2: STORE 1 | | 8 | | | | 3 |
| 3: STORE 2 | | | 8 | | | 4 |
| 4: DLOAD 1 | 1 | | | | | 5 |
| 5: STORE 3 | | | | 1 | | 6 |
| 6: DLOAD 3 | 3 | | | | | 7 |
| 7: STORE 4 | | | | | 3 | 8 |
| 8: JEQ 21 | | | | | | 9 |
| 9: SUB 3 | 2 | | | | | 10 |
| 10: JEQ 19 | | | | | | 11 |
| 11: STORE 4 | | | | | 2 | 12 |
| 12: LOAD 1 | 8 | | | | | 13 |
| 13: MULT 2 | 64 | | | | | 14 |
| 14: STORE 1 | | 64 | | | | 15 |
| 15: LOAD 4 | 2 | | | | | 16 |
| 16: SUB 3 | 1 | | | | | 17 |
| 17: STORE 4 | | | | | 1 | 18 |
| 18: JGT 12 | | | | | | 12 |
| 12: LOAD 1 | 64 | | | | | 13 |
| 13: MULT 2 | 512 | | | | | 14 |
| 14: STORE 1 | | 512 | | | | 15 |
| 15: LOAD 4 | 1 | | | | | 16 |
| 16: SUB 3 | 0 | | | | | 17 |
| 17: STORE 4 | | | | | 0 | 18 |
| 18: JGT 12 | | | | | | 19 |
| 19: LOAD 1 | 512 | | | | | 20 |
| 20: JUMP 22 | | | | | | 22 |
| 22: END | | | | | | 23 |



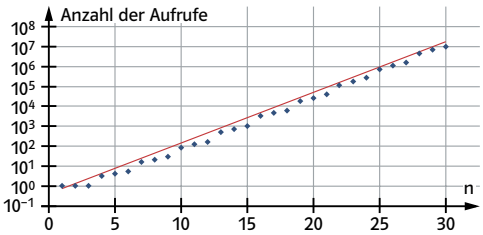
**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9

# Rückblick – Grenzen der Berechenbarkeit

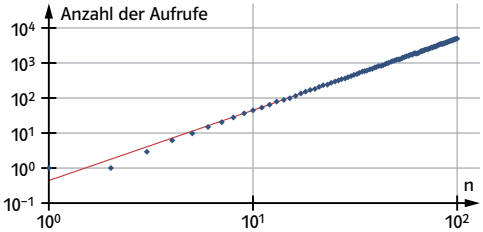
**Exponentielle Laufzeit**
Algorithmen, deren asymptotisches Laufzeitverhalten durch eine Exponentialfunktion beschrieben wird, führen bei der konkreten Berechnung von Funktionswerten sehr schnell zu unvertretbar hohen Laufzeiten. Bei exponentiellem Laufzeitverhalten ergibt die halblogarithmische Auftragung von experimentell ermittelten Laufzeiten eine Gerade. Ist durch die halblogarithmische Auftragung das asymptotisch exponentielle Verhalten gezeigt, so lässt sich aus den experimentellen Daten der funktionale Zusammenhang zwischen den Funktionsparametern und der Laufzeit ableiten.

*Die Funktion binomialZählen(int n, int m) zählt die Anzahl der rekursiven Aufrufe bei der Berechnung der Binomialfunktion. Auswertung für den Spezialfall $m = \frac{n}{3}$ mit der daraus ermittelten Ausgleichskurve (rot):*




**Polynomiale Laufzeit**
Algorithmen, deren asymptotisches Laufzeitverhalten durch ein Polynom beschrieben wird, sind – im Gegensatz zu Algorithmen mit exponentiellem Laufzeitverhalten – für praktische Zwecke einsetzbar. Polynomiales Laufzeitverhalten lässt sich durch eine experimentelle Laufzeitanalyse verbunden mit einer doppeltlogarithmischen Auftragung zeigen, da sich in diesem Fall eine Gerade ergibt. Der Grad des Polynoms, d.h. der maximale Exponent des Polynoms, folgt aus der Steigung der Geraden.

*Eine iterative Implementierung der Binomialkoeffizienten, beispielsweise mithilfe des Pascal'schen Dreiecks, weist polynomiales Laufzeitverhalten auf. Auswertung:*




**Symmetrische Verschlüsselung**
Bei symmetrischer Verschlüsselung müssen der Empfänger und der Sender einer verschlüsselten Nachricht denselben Schlüssel haben. Das große Problem bei symmetrischer Verschlüsselung ist die Schlüsselübergabe, die ebenfalls auf geheimem Weg erfolgen muss.

*Das nahe liegendste Beispiel einer symmetrischen Verschlüsselung ist ein einfacher Türschlüssel: Die Tür kann nur mit einem baugleichen Schlüssel verschlossen (d.h. verschlüsselt) und wieder geöffnet (d.h. entschlüsselt) werden.*

**Asymmetrische Verschlüsselung**
Bei asymmetrischer Verschlüsselung wird das Problem der Schlüsselübergabe durch einen öffentlichen Schlüssel, mit dem die Nachricht verschlüsselt wird, und einen privaten Schlüssel zur Entschlüsselung gelöst. Der öffentliche Schlüssel ist dabei jedem zugänglich, da grundsätzlich jeder eine Nachricht verschlüsseln darf. Der private Schlüssel zur Entschlüsselung ist geheim.

*Das „Zuklatschen" einer Tür mit Schnappschloss kann als Anwendung des öffentlichen Schlüssels angesehen werden (die Tür kann von jedem „zugeklatscht" werden). Das Aufschließen der Tür ist jedoch nur mit einem passenden privaten (und natürlich geheimen) Schlüssel möglich.*

**Berechenbarkeit und Halteproblem**
Die Existenz nichtberechenbarer Funktionen bedeutet, dass es mathematische Fragen gibt, die trotz ihrer Wohldefiniertheit (Existenz und Eindeutigkeit ist gesichert) algorithmisch nicht gelöst werden können. Ein wichtiges derartiges Problem ist das Halteproblem, d.h. die Frage nach einem allgemeinen Verfahren, um zu entscheiden, ob ein beliebiger Algorithmus terminiert.

*Ein einfaches nichtentscheidbares Problem ist bereits die Frage, welche von zwei reellen Zahlen die größere ist. Bezeichnen $a_n$ und $b_n$ jeweils die n-te Dezimalstelle der beiden Dezimalzahlen, so kann die Stelle $n_{max}$, bis zu der $a_n = b_n$ für alle $n < n_{max}$ gilt, beliebig groß sein. Das heißt, die Entscheidung, welche Zahl die größere ist, kann sich beliebig „hinauszögern".*



**Autoren:** Peter Hubwieser, Patrick Löffler, Petra Schwaiger, Matthias Spohrer, Markus Steinert, Siglinde Voß, Ferdinand Winhard
**Gestaltung:** Andrea Eckhardt
**Illustration:** Uwe Alfer, Andrea Eckhardt

**Seite aus:** Informatik 5 SB
ISBN: 978-3-12-731068-9