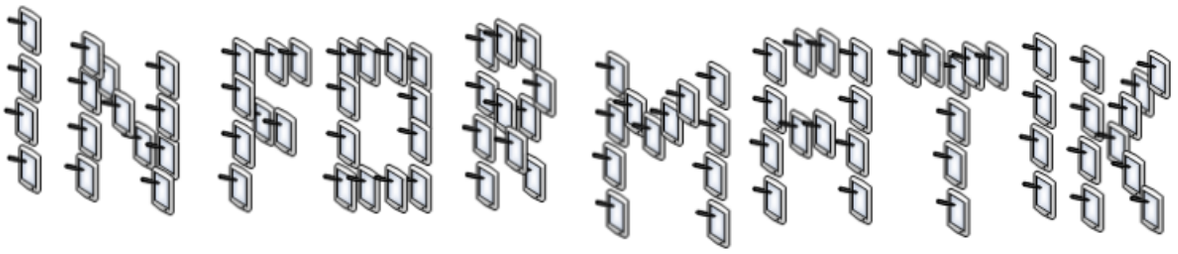


Informatik

Objektorientierte Programmierung

Datenbanken

Theoretische Informatik

Prolog und künstliche Intelligenz

Alexandra Horn, September 2014

Inhaltsverzeichnis

Mein Dank geht an alle Schüler, die vor dem Druck des Skriptes geholfen haben, Fehler im Skript zu finden.

Objektorientierte Programmierung

Objektorientierte Modellierung

Kurzer Überblick über die Entwicklung der Programmiersprachen

Im Laufe der Geschichte der Informatik gab es Hunderte von Programmiersprachen, die sich grundsätzlich jedoch in vier Programmierparadigmen zusammenfassen lassen:

1. die prozedurale Programmierung (z. B. Pascal, C)
2. die objektorientierte Programmierung (z. B. Delphi, Java, C++)
3. die funktionale Programmierung (z. B. LISP, LOGO, Miranda)
4. die wissensbasierte (auch logische) Programmierung (z. B. Prolog)

Die beiden erstgenannten Arten der Programmierung werden auch als imperative Sprachen bezeichnet, weil in ihnen ein Programm als Befehlsfolge aufgefasst wird. Die beiden letztgenannten Sprachen gelten hingegen als **deklarativ**, da in ihn ein Programm als berechenbare Funktion interpretiert wird.

<table>
<tr><th colspan="2">imperativ</th><th colspan="2">deklarativ</th></tr>
<tr><th>prozedural</th><th>objektorientiert</th><th>funktional</th><th>logisch</th></tr>
<tr><td>- Zerlegung in Unteralgorithmen (Prozeduren)
- Aufruf der Prozeduren im „Hauptprogramm meist am Ende des Quellcodes</td><td>- Programm besteht aus Objekten mit Attributen und Methoden
- Klassen sind Baupläne gleichförmiger Objekte
- Vererbung</td><td>- Eindeutige Abbildung der Eingabemenge auf die Ausgabemenge (mathematisch: Funktionsbegriff)
- Selbstaufruf möglich</td><td>- Sammlung von Fakten und Regeln
- Schlussfolgerungen führen zu neuen Erkenntnissen</td></tr>
<tr><td>Beispiel: Berechnung von Pi als Turbo-Pascal-Programm</td><td>Hallo Welt-Beispiel mit C++</td><td>Bespiel: Berechnung von 123 - 4.5 * 67.8 mit LISP</td><td>Beispiel: Verwandtschaftsverhältnisse in Prolog</td></tr>
<tr><td><pre>PROGRAM Pi_Berechnung_Archimedes;
USES dos,crt;
VAR m,n,s,u,p : real;
    ch   : char;
BEGIN
  ClrScr;
  WriteLn('Drücke fortlaufend die
Leertaste!');WriteLn;
  s:=1;
  m:=1;
  n:=6;
  p:=3;
  REPEAT
    WriteLn(m:3:0,' ',n:8:0,'   ',p:3:10);
    m:=m+1;
    n:=2*n;
    s:=sqrt(2-sqrt(4-s*s));
    u:=n*s;
    p:=u/2;
    REPEAT
      ch:=readkey
    UNTIL ch=' ';
  UNTIL m=21;
END.</pre></td><td><pre>#include

int main {
  cout << "Hello World\n";
  return 0;
}</pre></td><td><pre>(- 123 (* 4.5 67.8))</pre></td><td><pre>familie(ingrid, fred, anna-w).

familie(ingrid, fred, elsa-w).

familie(ingrid, fred, marcus).

familie(miriam,'?', mary-w).

familie(anna, eric, thea-w).

familie(mary, '?', james-m).

eltern(M,V,_):-
familie(M,V,_).</pre> [1]</td></tr>
</table>

[1] Tabelle vgl. (Dr. Engelmann, 2006, S. 41) und Inhalt der zugehörigen CD

Die Anforderungen an Computerprogramme haben sich im Laufe der Zeit verändert. Zu Beginn standen eine gute *Speicher- und Zeiteffizienz* im Vordergrund. Die Entwicklung besserer Hardware ließ dann auch komplexere Programme zu, die jedoch aufgrund der Programmiermethoden immer fehlerhafter wurden. Die Softwarekrise 1965 führte schließlich dazu, dass die Qualitätsmerkmale *Zuverlässigkeit, Korrektheit und Robustheit* verstärkt in den Fokus gerückt wurden. Diese Qualitätsmerkmale sollten vor allem mit der **strukturierten Programmierung** erreicht werden, die auf der Basis des Top-Down-Prinzips und der Methode der schrittweisen Verfeinerung agiert. Schließlich stellte man Anfang der 70er Jahre fest, dass die *Wartbarkeit* als Qualitätsmerkmal ergänzt werden musste. Seit Beginn der Neunzigerjahre hat sich die objektorientierte Programmierung (OOP) in der Praxis mehr und mehr durchgesetzt. Vor allem in Zeiten des Internets und der Verbreitung von Software in allen Lebensbereichen bietet die OOP große **Vorteile**:

- **Wiederverwendbarkeit von schon programmierten Elementen**
- **Aufteilung in überschaubare Einzelteile (Modularisierung)**
- **Erweiterung durch Schnittstellen**

Das objektorientierte Paradigma teilt sich in zwei miteinander verzahnte Bereiche auf: die objektorientierte Modellierung und die eigentliche objektorientierte Programmierung.

Kunden, die heute ein Programm in Auftrag geben, erwarten neben einer eingängigen leicht verständlichen Bedienbarkeit auch, dass das Programm schnell auf neuere Entwicklungen angepasst werden kann und selbstverständlich fehlerfrei arbeitet. Die althergebrachte Vorgehensweise zunächst zu programmieren und anschließend die Fehler des Programms zu suchen und zu beheben, hat sich jedoch als ungünstig erwiesen, so dass man sich eine bessere Vorgehensweise in der Softwareentwicklung suchen musste. Hieraus hat sich ein neuer Forschungszweig entwickelt: Software Engineering.

Software Engineering

Software Engineering beschäftigt sich mit der systematischen Entwicklung von großen Softwareprojekten, d. h. man untersucht Methoden und Techniken, die die Planung und Umsetzung einer Softwareidee bis zum fertigen Produkt und sogar darüber hinaus begleiten können, so dass sowohl die Kosten im Budgetrahmen bleiben, das Softwareproduktes eine hohe Qualität aufweist und der Zeitrahmen eingehalten werden kann.

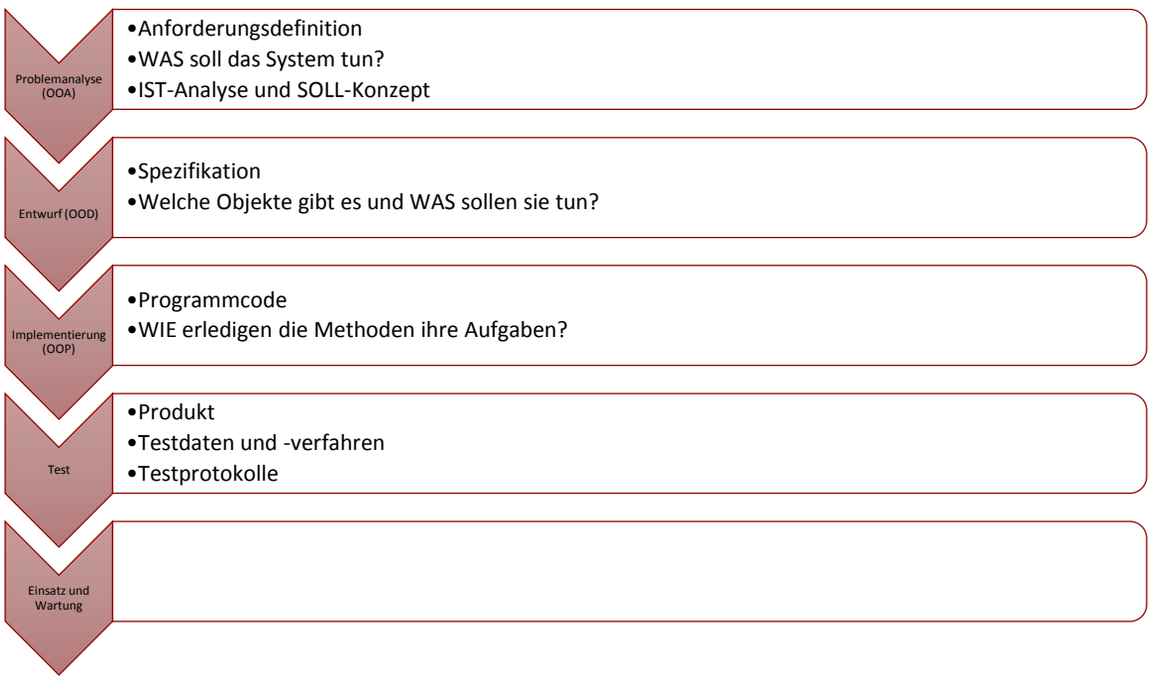
Folgende Fehlerschwerpunkte konnten zunächst ausgemacht werden:

- Die Kommunikation zwischen Kunde und Programmierer stellte sich als Manko heraus, so dass eigentlich gute Software beim Kunden aber schlecht ankam, weil der sich ein ganz anderes Programm mit anderen Möglichkeiten vorgestellt hatte.
- Die Fehlerbehebung bei fertigen Programmen führte häufig zu komplexen Umstrukturierungen, durch die neue teilweise nicht mehr überschaubare Fehler hinzukamen.
- Viele Programme wurden nur unzureichend getestet, so dass die eigentlichen Probleme dann erst in der Praxis auftauchten.

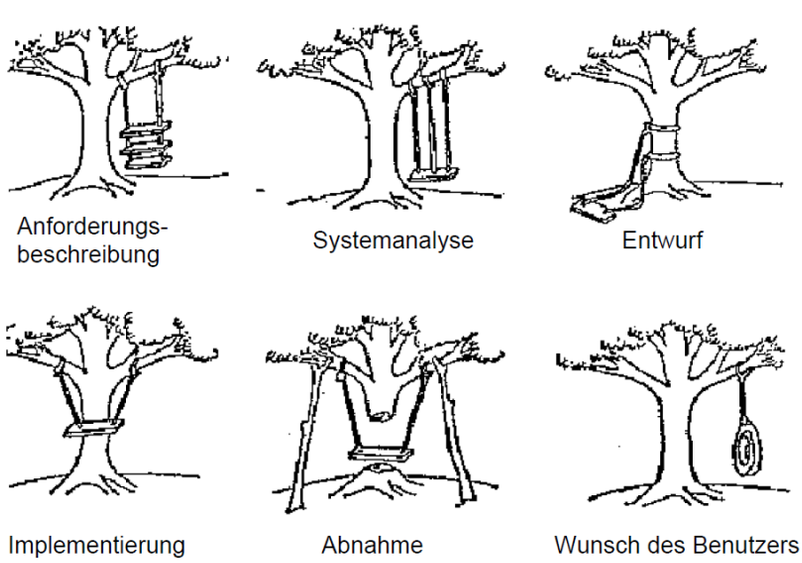
Die Konsequenz aus diesen Fehlerquellen lautet: Alle Beteiligten muss zu jedem Zeitpunkt der Entwicklung des Programms klar sein, an welcher Stelle man sich befindet und wer welche Aufgabe zu erfüllen hat. In der Vergangenheit wurden zahlreiche Modelle für solche Entwicklungsprozesse entwickelt.

Phasen der Programmentwicklung

1. Modell:

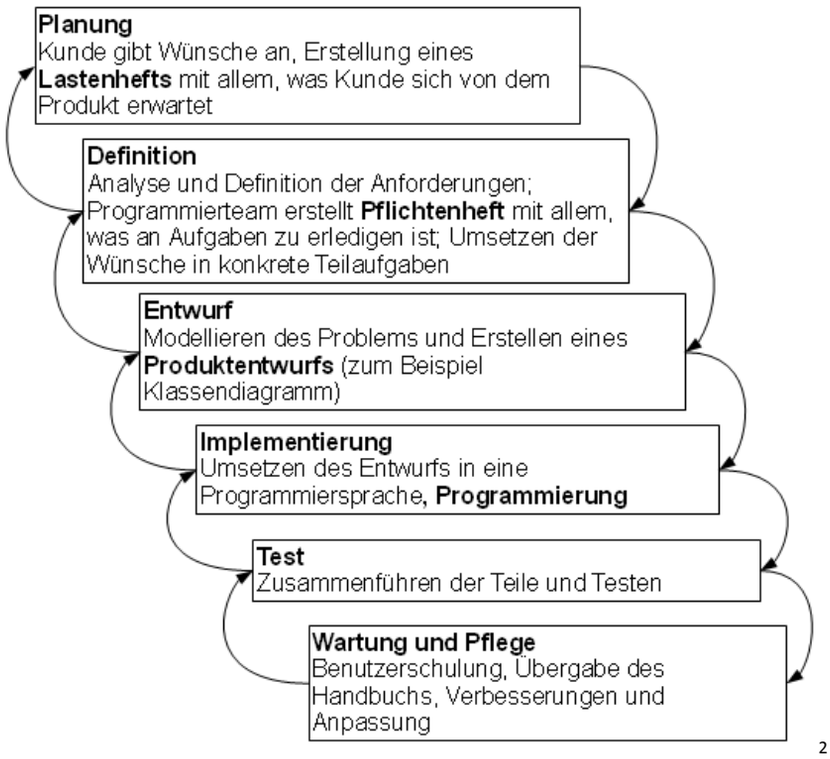


2. Modell:



3. Modell: Das Wasserfallmodell

Eine klassische Variante stellt dabei das Wasserfallmodell dar, wobei in diesem Modell jede untere Phase erst durchlaufen werden kann, wenn die darüber stehende Phase durchlaufen wurde.



[2]

Aufgaben:

1. Mit welcher Phase des Wasserfallmodells versucht man die oben genannten Fehlerschwerpunkte in den Griff zu bekommen?
2. In welchen Phasen ist der Einfluss des Auftraggebers besonders hoch?

Für viele Phasen gibt es nun unterstützende Werkzeuge, z.B. das Lastenheft bzw. Pflichtenheft oder UML-Diagramme in der Modellierung. Betrachten wir zunächst die Entwicklung eines Programmes aus der Sicht eines Kunden.

Das Lasten- und Pflichtenheft

Häufig ergibt sich aus dem Alltag in einem Betrieb die Idee, dass bestimmte Abläufe mithilfe eines Programms vereinfacht werden sollen. Der Auftraggeber hat also einerseits konkrete Vorstellungen, wie er ein Problem gerne gelöst hätte, andererseits fehlt ihm die Kenntnis über die Möglichkeiten, in welcher Art und Weise das Problem später am Computer gelöst werden könnte. Die Programmierer haben andererseits keine Kenntnisse über die genauen Abläufe im Firmenalltag. Um Klarheit über die Vorstellungen aller Beteiligten zu bekommen, dienen die beiden Hefte.

Im Lastenheft werden während der Planungsphase die Wünsche des Auftraggebers möglichst genau formuliert, u. a. muss zunächst geklärt werden, mit welchem Betriebssystem die Firma arbeitet, ob bestimmte Bedingungen durch den Gesetzgeber vorgegeben werden oder Anforderungen bzgl. einer Zertifizierung an den Auftragnehmer gestellt werden. Das Lastenheft macht deutlich, WAS erwartet wird. Im Pflichtenheft nimmt der Auftragnehmer nun Stellung zu den aufgeführten Anforderungen und legt dar, in welcher Art und Weise er die Wünsche des Kunden mit welchen Mitteln umsetzen möchte. Damit sowohl Auftraggeber als auch Auftragnehmer mit dem Heft des anderen gut zurechtkommen, empfiehlt

[2] (Rau, 2014) am 7.9.2014

es sich, dass sich beide zuvor auf annähern die gleiche Struktur verständigen. Helmut Banzer schlug dazu 2001 den folgenden Aufbau vor, der heute am häufigsten zum Einsatz kommt.

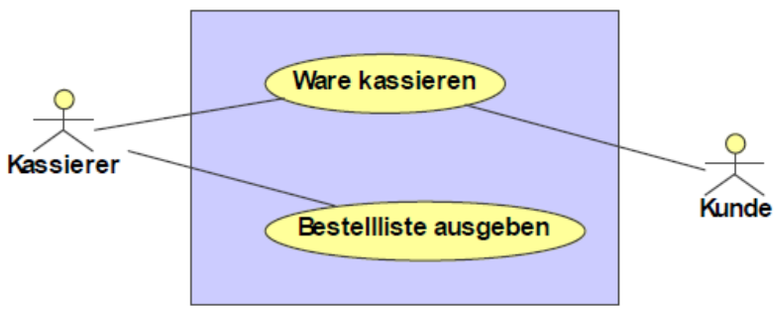
Aufbau eines Lasten- bzw. Pflichtenheftes

1. **Zielbestimmung**:
   Welche Musskriterien sind für das Produkt unabdingbar?
   Welche Sollkriterien werden angestrebt?
   Welche Kannkriterien sollten angestrebt werden?
   Was muss es nicht leisten?
   Welche Abgrenzungskriterien gelten für das Produkt?
2. **Produkteinsatz:**
   Für welche Anwendungsbereiche und welche Zielgruppen ist das Produkt vorgesehen? Unter welchen Betriebsbedingungen wird das Produkt laufen?
3. **Produktübersicht:**
   Überblick über Produktumgebung, kann auch grafisch sein, z. B. Anwendungsfalldiagramm
4. **Produktfunktionen**:
   Welche typischen Arbeitsabläufe sind mit dem Produkt durchzuführen? Welche Schnittstellen zu anderen Produkten, Geräten bzw. Personen soll es geben?
5. **Produktdaten**:
   Welche Hauptdaten sind langfristig aus Benutzersicht zu speichern?
6. **Produktleistungen**:
   An welchen Hauptfunktionen oder Hauptdaten werden Leistungsanforderungen bzgl. Zeit oder Genauigkeit bestellt?
7. **Qualitätsanforderungen**:
   Welche Qualitätsstandards muss das Produkt bzgl. Zuverlässigkeit, Effizienz,… erfüllen?
8. **Benutzeroberfläche**:
   Wie soll die Benutzeroberfläche aussehen?
   Welche Zugriffsrechte gibt es?
9. **Nichtfunktionale Anforderungen**:
   Aussehen?
   Handhabung?
   Wartung?
   Plattformunabhängigkeit?
10. **Technische Produktumgebung**:
    Welche Softwareumgebung läuft auf dem System?
    Welche relevanten Hardwaredaten besitzt das System?
    Welche Produktschnittstellen gibt es?
11. **Spezielle Anforderungen an die Entwicklungsumgebung:**
12. **Gliederung in Teilprodukte** (falls möglich)
13. **Ergänzungen**[3]

Die Punkte 8 bis 12 sind eher nur im Pflichtenheft zu finden.

[3] Vgl. (Preckel, 2012, S. 15)

Betrachten wir ein Beispiel: Am Schulkiosk soll eine neue Registrierkasse angeschafft werden. Für diese Kasse muss eine Software entwickelt werden, die anhand der verkauften Waren eine Bestellliste mit den neu benötigten Waren erzeugen kann. Das zugehörige Lastenheft könnte so aussehen:

<table>
<tr><td colspan="4">Lastenheft: Registrierkasse Schulkiosk</td></tr>
<tr><td>Zielbestimmung</td><td colspan="3">Die Registrierkassensoftware soll die verkauften Waren erfassen und aus diesen Daten eine Bestellliste mit den neu benötigten Waren generieren.</td></tr>
<tr><td>Produkteinsatz</td><td colspan="3">Die erstellte Software soll in die Registrierkassensoftware integriert werden und auf die dort hinterlegten Stammdaten (Artikelnummern etc.) zugreifen können. Die Software soll leicht von den Schülerinnen und Schülern einer Schülerfirma bedienbar sein.</td></tr>
<tr><td>Produktübersicht</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="2">Produktfunktionen</td><td>/LF10/</td><td>Prozess:
Akteur:
Beschreibung:</td><td>Ware kassieren
Kassierer
Die verkaufte Ware wird in der Registrierkasse erfasst und der zu zahlende Betrag vom Kunden bezahlt.</td></tr>
<tr><td>/LF20/</td><td>Prozess:
Akteur:
Beschreibung:</td><td>Bestellliste ausgeben
Kassierer
Die verkaufte Ware wird nach Artikelnummern zusammengefasst und als Bestellliste ausgegeben.</td></tr>
<tr><td>Produktdaten</td><td colspan="3">Artikeldaten mit Lieferant, vorhandener Menge und verkaufter Menge</td></tr>
<tr><td>Produktleistungen</td><td colspan="3">Alle Prozesse müssen zügig und fehlerlos funktionieren.</td></tr>
<tr><td>Qualitätsanforderungen</td><td colspan="3">hohe Zuverlässigkeit, sehr gute Bedienbarkeit auch für ungeübte Arbeitskräfte</td></tr>
<tr><td>Benutzeroberfläche</td><td colspan="3">Die Benutzeroberfläche soll kontrastreich und „idiotensicher“ sein.</td></tr>
</table>

[4]

Aufgaben:

3. Für den Weihnachtsmarkt der Schule plant ein Kurs die Erstellung eines digitalen Adventskalenders, der auf CD gebrannt und dort verkauft werden soll. Ausgehend von der Startseite soll man jeden Tag ein „Türchen“ (Link) öffnen können. Unter anderen sollen dort Geschichten, Bastelideen, Spiele, etc. versteckt sein. Natürlich muss gewährleistet sein, dass man nur „Tag“ bis zum aktuellen Datum öffnen kann. Erstelle ein Lastenheft für diese Adventskalender-CD.

4. Ein Bekannter möchte ein Programm in Auftrag geben, leider weiß er nicht, an welcher Stelle er die nötigen Eingaben im Lastenheft tätigen soll. Hilf ihm beim Ausfüllen des Lastenheftes, indem du die Kategorien jeweils zuweist.
   a. Alle Prozesse müssen jederzeit online ausgeführt werden können.
   b. Bedienbarkeit auch bei kaltem oder feuchtem Wetter

[4] Zusatzmaterial: (Preckel, 2012, S. 17)

c. Touch-Bedienoberfläche
d. Für eine kilometergenaue Abrechnung der Fahrradkuriere soll jeder Fahrer mit einem eigenen Gerät ausgestattet werden.
e. Drahtlose Übertragung
f. Kassieren des Fahrers beim Kunden
g. Sehr gute Bedienbarkeit auch für Aushilfsfahrer
h. Datenbank der angebotenen Leistungen, aktuellen Aufträgen und Rechnungen

5. Ein Kino plant, einen Online-Reservierungsservice anzubieten. Kinogänger sollen im Internet das aktuelle Kinoprogramm abrufen und Reservierungen für bestimmte Vorstellungen vornehmen können. Im Kino können dann an der Kasse die Karten abgeholt werden, wenn ein entsprechender Code genannt werden kann. Erstellen Sie ein Lastenheft für das Online-Kinoreservierungssystem.

Objektorientierte Modellierung

Nach der Planungs- und Definitionsphase folgt nun die Entwurfsphase, in der sich nun über Zusammenhänge und Umsetzungsmöglichkeiten Gedanken machen muss. Hier bietet UML eine gute Unterstützung. Unified Modeling Language, ist eine „Sprache“ zur Veranschaulichung von Klassen, Objekten und deren Zusammenspiel. Die UML ist in erster Linie eine einheitliche Notation für die Modellierungs- und Designprozesse, sie stellt keine Entwicklungsmethode dar und wird seit 1997 angewendet. Diese Entwurfsphase ist identisch mit der objektorientierten Modellierung, wenn man das Softwareprojekt mit einer objektorientierten Programmiersprache verwirklichen will.

Im Prinzip geht die objektorientierte Modellierung auf ein grundlegendes Erkenntnisprinzip der Menschen zurück, welches von dem Philosophen Jostein Gaarder folgendermaßen zusammengefasst worden ist:

„Ich sehe ein Pferd, dann sehe ich noch ein Pferd – dann noch eins. Die Pferde sind nicht ganz gleich, aber es gibt etwas, das allen Pferden gemeinsam ist, und das, was allen Pferden gemeinsam ist, ist die ‚Form‘ des Pferdes. Was unterschiedlich oder individuell ist, gehört zum ‚Stoff‘ des Pferds.“

Hengst Fury

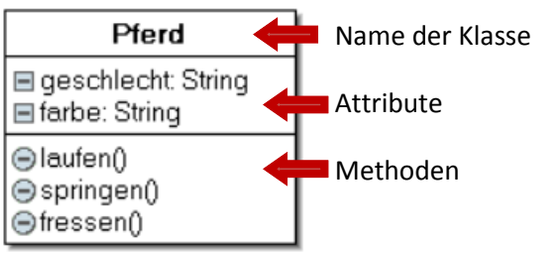
Stute Jacqueline

„An der „Form“ können schon kleine Kinder ein Pferd erkennen, genauso wie sie eine Katze, ein Auto oder einen Baum erkennen können. Erkennen sie über die Form eine Wespe, können sie direkt auf deren gefährlichen Stachel schließen. Mit der Form sind auch direkt bestimmte Fähigkeiten verknüpft, so kann das Pferd aufgrund seiner Form laufen, springen und vieles mehr.

Der „Stoff“, von dem Gaarder spricht, sind die individuellen Ausprägungen des Pferdes, wie z. B. seine Farbe, sein Geschlecht oder seine Schnelligkeit. Ein individuelles Pferd konkretisiert sich somit erst durch die Ausprägung bestimmter Merkmale der Form.“ (Kempe & Tepaße, Informatik 1; Softwareentwicklung mit Greenfoot und BlueJ, 2010, S. 15)

Was hat das nun mit Informatik zu tun? Dazu müssen wir die Begriffe von „Stoff“ und „Form“ informatisch definieren. „Die beiden Pferde Fury und Jaqueline sind informatisch gesprochen zwei Objekte der Klasse Pferd. Die grundlegenden Gemeinsamkeiten der beiden bzw. aller Pferdeobjekte werden in einem Bauplan, der sogenannten Klasse, abgebildet. Sie beinhalten die Attribute (Eigenschaften), die für die einzelnen Objekte konkrete Werte“ (Kempe & Tepaße, Informatik 1; Softwareentwicklung mit Greenfoot und BlueJ, 2010, S. 15) annehmen können, und Methoden (Fähigkeiten). Methoden stellen dabei Operationen dar, mit denen Objekte manipuliert werden können. Man unterscheidet zwischen Prozeduren und Funktionen. Über Prozeduren kann der Zustand von Objekten geändert werden und mit Funktionen lässt sich Auskunft über den Zustand von Objekten einholen.

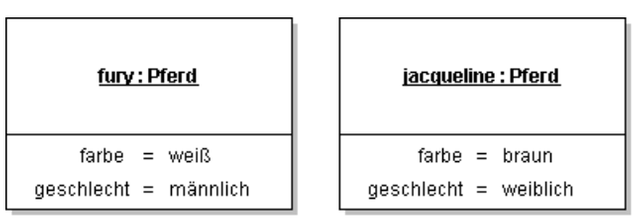
Die Klasse PFERD können wir vereinfacht als UML-Klassendiagramm darstellen:



Im UML-Klassendiagramm werden Klassen als Rechteck dargestellt. Der Klassenname steht im Singular und zentriert im Kopf des Rechtecks. Darunter – durch einen Strich abgetrennt – werden die Attribute linksbündig aufgeführt. Ganz unten stehen die Methoden der Klasse. Diese werden in der Kurzform mit dem Klammerpaar gekennzeichnet, d. h. die Details der Parameterliste werden nicht angegeben.

Jedes Attribut kann mit seinem Namen, z. B. geschlecht, der Schutzklasse (- private, # protected oder + public) und dem Datentyp, z .B. String charakterisiert werden. Methoden erhalten ebenfalls einen Namen und eine Schutzklasse. Zudem kann in den runden Klammern noch eine Parameterliste übergeben werden und bei Funktionen muss der Datentyp des Rückgabewertes ebenfalls angegeben werden. Auf die näheren Bedeutungen werden wir später noch zurückkommen. Während der Klassenname in Java großgeschrieben wird, werden alle anderen Namen kleingeschrieben.

Ein Objekt dagegen verkörpert die konkrete Umsetzung der Klasse, es ist ein Exemplar der Klasse. Am Beispiel der Pferde würde das bedeuten, dass es sich um ein ganz bestimmtes Pferd handelt. Beispielsweise um einen weißen Hengst oder eine braune Stute. Das sind dann Objekte der Klasse Pferd.



Im UML-Objektdiagramm werden Objekte ebenfalls als Rechteck dargestellt, allerdings wird der Objektname unterstrichen. Bei Bedarf kann dem Objektname der Klassenname folgen, der durch einen Doppelpunkt vom Objektnamen getrennt wird. Bei Objekten gibt man nur die Attribute mit den belegten Werten an. Die Methoden werden nicht angegeben, da sie für alle Objekte einer Klasse gleich sind.

Die oben bereits benannten Vorteile der OOM lassen sich hier nun konkretisieren:

- **Wiederverwertbarkeit**: Eine einmal programmierte Klasse Pferd ermöglicht es, unendlich viele Objekte der Klasse Pferd zu erzeugen. Man muss lediglich die Attribute mit konkreten Werten belegen.

- **Aufteilung in überschaubare Einzelteile**: Ein Pferderennen kann programmiert werden, indem Klassen wie Pferd, Reiter etc. einzeln programmiert werden.

- **Erweiterungsmöglichkeiten**: Sollte zu dem Pferderennen ein Wettbüro hinzukommen, müssen lediglich die neuen Klassen Wettbüro und Wette erstellt werden.[5]

Die wichtigsten Bestandteile eines objektorientierten Programms sind also Objekte und Klassen. Objekte sind dabei Elemente, die in einem Anwendungsfall von Bedeutung sind. So stellt die Bilanz des Monats November in einem Finanzbuchhaltungsprogramm ein Objekt dar. Bei einem Auftragsverwaltungsprogramm wird man es mit verschiedenen Kunden und Aufträgen zu tun haben. Aus Sicht des Benutzers stellen Objekte bestimmte Dienstleistungen und Informationen zur Verfügung. Aus Sicht des Programmierers sind Objekte Funktionseinheiten eines Programms, die zusammenarbeiten, um eine gewünschte Funktionalität zur Verfügung zu stellen.

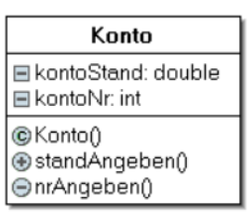
Eine Klasse entsteht durch die Abstraktion von den Details gleichartiger Objekte und beschreibt die Eigenschaften und Struktur einer Menge nahezu gleicher Objekte. Die Objekte sind Exemplare dieser gemeinsamen Klasse. Klassen besitzen einen Mechanismus, um neue Objekte zu erzeugen (Konstruktor). Jedes erzeugte Objekt gehört genau einer Klasse an. Es gibt allerdings auch sogenannte abstrakte Klassen, von denen keine Objekte gebildet werden. Ebenso gibt es Klassenmethoden, die auch aufgerufen werden können, wenn kein Objekt der Klasse erschaffen worden ist. Die Methode main() ist eine solche Klassenmethode.

Man darf die Klasse nicht mit der Menge aller Objekte dieser Klasse verwechseln. Die Klasse ist eine Abstraktion, die Gemeinsamkeiten von Objekten und Regeln zu ihrer Erzeugung beschreibt. Eine Menge von Objekten ist dagegen einfach eine Ansammlung von Objekten.

Aufgaben:

1. In einer Obstgärtnerei werden auf Karteikarten zu den Apfelbäumen die Maximalhöhe, die Blütedauer und der Wasserbedarf pro Woche notiert. Des Weiteren interessiert den Betrieb, dass der Baum erst blühen und dann Obst tragen kann. Erstelle ein Klassendiagramm der Klasse Apfelbaum.

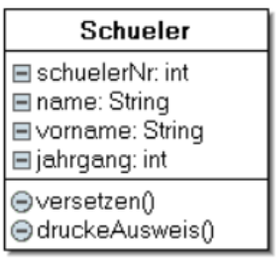
2. Gegeben ist das nebenstehende UML-Diagramm. Erkläre anhand dieses Diagramms die Begriffe Klasse, Objekt, Attribut und Methode mit eigenen Worten. Grenze insbesondere die Begriffe Klasse und Objekt voneinander ab.



[5] Vgl. (Kempe & Tepaße, Informatik 1; Softwareentwicklung mit Greenfoot und BlueJ, 2010, S. 15f)

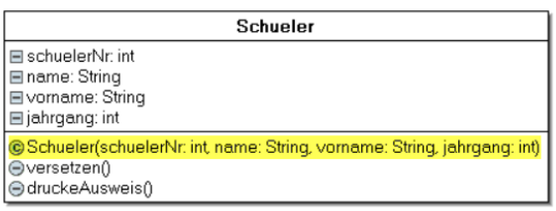
Programmierung von Klassen und Objekten in Java

Der Java-Editor hat im UML-Menü einen Befehl zum Anlegen neuer Klassen. Diese stellen sich dann wie im Beispiel dar:




```
1  public class Schueler {
2
3    // Anfang Attribute
4    private int schuelerNr;
5    private String name;
6    private String vorname;
7    private int jahrgang;
8    // Ende Attribute
9
10   // Anfang Methoden
11   private void versetzen() {
12
13   }
14
15   private void druckeAusweis() {
16
17   }
18
19   // Ende Methoden
20 } // end of Schüler
21
```

Um Objekte erstellen zu können, benötigt man einen **Konstruktor**, wobei eine Klasse auch mehrere Konstruktoren besitzen kann. Wir benötigen hier nur einen, den wir unserer Klasse jetzt noch hinzufügen:




```
8   // Ende Attribute
9
10  private Schueler(int schuelerNr, String name, String vorname, int jahrgang) {
11    this.schuelerNr = schuelerNr;
12    this.name = name;
13    this.vorname = vorname;
14    this.jahrgang = jahrgang;
15  }
16
17  // Anfang Methoden
```

Nun kann man Schülerobjekte erzeugen, hierzu erstellen wir ein einfaches Konsolenprogramm SchuelerObj.java. Im Beispiel werden zwei Schüler erzeugt. Schüler 1 wird zunächst als Variable deklariert und anschließend mit new erzeugt. Schüler zwei wird als Variable deklariert und gleichzeitig erzeugt. Beide Vorgehensweisen sind möglich. Man erkennt an diesem Verfahren, dass eine Klasse einem Datentyp entspricht. Von Datentypen kann man Variablen deklarieren. Da es sich jedoch bei einer Klasse nicht um einen einfachen Datentyp handelt (wie int oder double) müssen die Werte mittels new erzeugt werden. Damit wir in unserem Konsolenprogramm überprüfen können, ob die Objekte auch erzeugt wurden, müssen wir in die Methode druckeAusweis() der Klasse schreiben. Die Attributwerte wurden alle als privat deklariert (am – vor dem Attributnamen zu erkennen), so dass wir nicht direkt auf die Attributwerte zugreifen können.

```
public class SchuelerObj {

 public static void main(String[] args) {
   Schueler s1;
   s1 = new Schueler(123,"Meier","Frida",9);
   Schueler s2 = new Schueler(124,"Freund","Paul",5);

   System.out.println("Es wurden zwei Schüler angelegt.");
   s1.druckeAusweis();
   s2.druckeAusweis();
 }
}
```

```
public void druckeAusweis() {
   System.out.println("SCHÜLERAUSWEIS");
   System.out.println("Schülernummer: " + this.Schuelernummer);

}
```

Aufgaben:

3. Ergänze noch drei weitere Schüler.
4. Ergänze die Methode druckeAusweis(), so dass auch der Name, der Vorname und der Jahrgang ausgegeben werden.
5. Schreibe die Methode versetze(). Überprüfe, ob deine Methode korrekt arbeitet.

Die objektorientierte Analyse (OOA) - Fachkonzept

Programmieren ist nicht Selbstzweck. Ziel eines Programmierers sollte immer sein, dass ein zukünftiger Benutzer seine Aufgaben effizienter erledigen kann. Daher ist eine genaue Analyse der aktuellen Situation und der gewünschten Neuerungen nötig, um die Anforderungen an das zu entwickelnde Programm zu erfassen. Traditionell werden die Wünsche eines Auftraggebers in einem Lasten- und Pflichtenheft dokumentiert. Als Standardnotation für die objektorientierte Modellierung hat sich UML durchgesetzt. Das entstehende OOA-Modell ist dann die Grundlage für den darauffolgenden Entwurf und schließlich für die Programmierung. Während der Erstellung des OOA-Modells soll die spätere Implementierung völlig ausgeklammert werden, die verwendete Programmiersprache spielt keine Rolle, denn bei der objektorientierten Analyse wird nicht beschrieben, wie ein Objekt auf der Benutzeroberfläche dargestellt oder gespeichert werden soll. Das Augenmerk soll nur auf die Objekte der realen Welt und deren Bedeutung für das Problem gerichtet werden. Es wäre schließlich keinem Anwender geholfen, wenn er ein tolles Programm erhält, welches aber nicht seinen Anforderungen gerecht wird. Ziel der objektorientierten Analyse ist es also, das zu realisierende Problem zu verstehen und in einem OOA-Modell zu beschreiben. Das OOA-Modell bildet dann die fachliche Lösung des zu realisierenden Systems. Man nennt es daher auch Fachkonzept. Jede Klasse, die im OOA-Modell auftaucht, ist eine **Fachklasse**.

Zusammenfassung:

Bei der objektorientierten Analyse wird ein Modell erstellt, welches

- konsistent, vollständig, eindeutig und realisierbar ist
- Aspekte der Implementierung bewusst ausklammert (Annahme über “perfekte Technologie”)
- Festlegt, was das System tun soll, aber noch nicht wie es realisiert werden soll

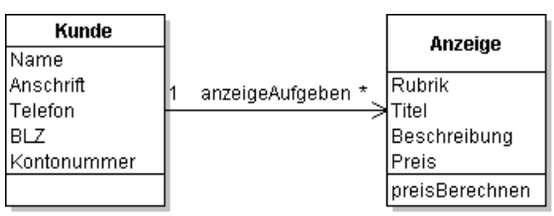
Wie erstellt man ein OOA-Modell?

Der Auftraggeber erklärt meist an Beispielen, welche Anforderungen er an das zu erstellende System hat. Die Aussagen des Auftraggebers können dabei unvollständig, widersprüchlich oder unklar sein. Meist werden allgemeine Forderungen und Details vermischt, so dass der Systemanalytiker aus dem bunten Informationsmix ein schlüssiges Modell erstellen muss. Es ist seine Aufgabe sich dem Auftraggeber gegenüber verständlich zu machen, nicht umgekehrt. Dabei muss er in der Lage sein vom Konkreten zum Abstrakten zu denken. Er muss also die für das Problem wichtigen von den unwichtigen Dingen unterscheiden. Dies nennt man Abstraktion.

Beispiel:
Eine Tageszeitung benötigt eine Inserenten- und Kleinanzeigen-Verwaltung. Die Systemanalytikerin Fr. Müller führt folgendes Interview mit dem Auftraggeber Hr. Neuhaus.

Fr. Müller: Welche Aufgaben wollen Sie mit dem Inserenten-und Anzeigenprogramm erledigen?
Hr. Neuhaus: Unsere Sachbearbeiterinnen nehmen die Anzeigenwünsche der Inserenten in der Regel per Telefon auf und notieren die Daten des Inserenten und den Anzeigentext in einem Textsystem. Das Textdokument wird dann sowohl an die Setzerei als auch an die Buchhaltung weitergegeben. Wenn ein Inserent nach einiger Zeit erneut eine Anzeige aufgibt, muss er seine persönlichen Angaben alle noch einmal machen. Das kostet Zeit und verärgert unsere Kunden.
Fr. Müller: Welche Daten werden über jeden Inserenten und jede Anzeige erfasst?
Hr. Neuhaus: Von jedem Inserenten werden Namen, Anschrift, Telefonnummer, Bankleitzahl und Kontonummer erfasst. Für jede Kleinanzeige sind folgende Angaben notwendig: Rubrik, Titel, Beschreibung und Preis.
Fr. Müller: Sollen in dem neuen Softwareprogramm alle Kleinanzeigen, die ein Inserent im Laufe der Zeit aufgibt, diesem Inserenten zugeordnet werden?
Hr. Neuhaus: Ja, das wäre praktisch, da wir dann wissen, wer ein häufiger Kunde ist und ihm einen Rabatt anbieten können.
Fr. Müller: Wie berechnen Sie im Moment den Preis für eine Kleinanzeige?
Hr. Neuhaus: Die Wortanzahl bestimmt den Preis; der sollte in Zukunft automatisch berechnet werden.

Fr. Müller erstellt daraufhin folgendes OOA-Modell:



Um dieses Modell zu erstellen, musste Fr. Müller von den konkreten Angaben des Hr. Neuhaus abstrahieren. Obwohl es zunächst nahe liegen würde, eine Klasse Inserent vorzusehen, entscheidet sich Fr. Müller für die allgemeinere (und daher bessere) Lösung eine Klasse Kunde, um später auch Abonnements verwalten zu können. Zudem muss Fr. Müller anhand der Äußerungen erkennen, welche Attribute und Methoden die beiden Klassen besitzen müssen und in welche Beziehung die beiden Klassen zueinander stehen.

Assoziation – Die kennt-Beziehung zwischen Klassen

Zwischen den Objekten von Klassen können konkrete Beziehungen bestehen. Beispielsweise kann der Kunde Tom Sommer ein Inserat aufgeben. Es ist aber laut Hr. Neuhaus auch möglich, dass ein Kunde mehrere Inserate aufgibt, so dass die Kundin Sonja Walder vier Kleinanzeigen aufgegeben hat. Diese Objektbeziehungen werden durch Assoziationen modelliert. So wie Objekte Exemplare von Klassen sind, sind Objektbeziehungen Exemplare einer Assoziation.

In UML-Diagrammen werden Assoziationen als Strecken zwischen Klassen dargestellt. Die Strecke wird mit einem **Beziehungsnamen** (im Beispiel: anzeigeAufgeben) beschriftet. Dieser sollte die Beziehung inhaltlich beschreiben.

Häufig kommen auch **gerichtete** Assoziationen vor, so dass man einen Pfeil für die Orientierung hinzufügt. Sie sind dann nur in eine Richtung navigierbar. Assoziationen, bei denen nur das eine der beiden beteiligten Objekte auf die Attribute oder Methoden des anderen Objekts zugreift, werden als **unidirektional** bezeichnet. Findet die Nutzung der Assoziation in beide Richtungen statt, so spricht man von **bidirektional**.

Für jede Assoziation muss dann noch die **Multiplizität,** häufig auch **Kardinalität** genannt, festgelegt werden. Die Multiplizität einer Assoziation gibt an, mit wie vielen Objekten der gegenüberliegenden Klasse ein Objekt assoziiert sein kann. Dabei steht eine bestimmt Zahl für die entsprechende Anzahl, * steht für beliebig viele. Es können auch Bereiche, z. B. 0 .. 3 angegeben werden, so dass die Anzahl zwischen 0 und 3 liegt. Ist das Minimum 0, so ist die Beziehung optional. Für unser Beispiel bedeutet dies: Jeder Kunde kann beliebig viele Anzeigen aufgeben. Eine Anzeige kann immer nur von einem bestimmten Kunden aufgegeben werden.

Eine Assoziation beschreibt immer eine bestimmte Art von Kommunikation zwischen zwei Objekten, für die es folgende Möglichkeiten gibt:

- Ein Objekt **nutzt Daten** (Attributwerte) eines anderen Objektes.
- Ein Objekt **ruft Methoden** eines anderen Objektes **auf**.

Da die Attribute gewöhnlich privat deklariert sind, kann die direkte Nutzung von Attributwerten anderer Objekte nur stattfinden, wenn beide Objekte derselben Klasse angehören. In allen anderen Fällen kann die Kommunikation nur über den Aufruf spezieller Lese- oder Schreibmethoden ausgeführt werden.

Aufgaben:

6. Modelliere mit UML Autos als Klasse mit den Attributen Kfz-Kennzeichen, Kilometerstand, Tankvolumen, Kraftstoffverbrauch, Kraftstoffmenge, den Methoden tanken(Menge) und fahren(Strecke), sowie einen Konstruktor erzeuge(Kennzeichen, Tankvolumen, Verbrauch).

7. Es soll ein Lagerverwaltungsprogramm erstellt werden.
    a. Versuche die Lagerhalle so weit wie möglich zu abstrahieren. Überlege, welche Eigenschaften du wirklich benötigst: Die Lagerhalle soll lediglich dazu dienen, Waren aufnehmen und verwalten zu können. In dieser Lagerhalle sollen Möbel gelagert werden. Die Lagerhalle steht in der Hauptstraße 211 und hat ein Flachdach. In dem roten Gebäude können maximal 4000 Möbelstücke gelagert werden. Insgesamt können 5 LKW zeitgleich be- und entladen werden. Der aktuelle Bestand beträgt 3456 Möbelstücke.

b. Überlege dir nun, welche Attribute die Klasse Moebelstück benötigt. Denke dabei einerseits natürlich an die Eigenschaften von Möbeln, aber versuche, die Möbel auch etwas allgemeiner (als einzulagernde Ware) zu betrachten. Stelle dir einen Tisch und einen Stuhl vor: Was verbindet die beiden, welche Gemeinsamkeiten haben sie, die für Möbel von Bedeutung sind. Erstelle nun ein OOA-Modell für das Lagerverwaltungsprogramm.

8. Überlege dir, wie du die folgenden Angaben in einem Klassendiagramm umsetzen kannst. Gib auch die Multiplizitäten der Assoziationen an. Handelt es sich um gerichtete Assoziationen?
   a. Der Schüler Fabian hat einen Führerschein und fährt ein Auto mit dem Kennzeichen KB – F 1.
   b. Der Kunde Hans Winter hat bei der Wildunger Bank zwei Konten mit den Nummern 19465 und 80233. Angelika Paulus ist bei der Hinterwälder Bank Kundin. Sie hat dort die Kontonummer 343434.
   c. Jan und Kai spielen in derselben Mannschaft Fußball. Heute haben sie ein Spiel gegen die Mannschaft aus dem Nachbarort. Paul hat sich fest vorgenommen mehr Tore zu schießen als Kai reinlässt.

9. Was bezeichnet man als Modellierung?
   - Den Quelltext für unsere Klassenschreiben.
   - Unwichtige Eigenschaften realer Gegenstände auszusortieren.
   - Abstraktes Wissen über Gegenstände mit Hilfe von Klassen auszudrücken.

10. Worauf kommt es bei der Abstraktion ganz besonders an?
    - Möglichst viele Eigenschaften des realen Objektes zu übernehmen.
    - Die relevantesten Eigenschaften eines realen Objektes zu übernehmen.
    - Bei der Auswahl der Eigenschaften den Zweck des Programms nie aus den Augen zu verlieren.

Produkte der objektorientierten Analyse

- **Pflichtenheft**: Hier wird der Leistungsumfang beschrieben.
  Das Pflichtenheft beschreibt die erwartete Leistung aus Sicht des Auftraggebers. Es handelt sich um eine textuelle Beschreibung dessen, was das zu realisierende System leisten soll.
- **OOA-Modell**: Beschreibung der fachlichen Lösung (Fachkonzept)
  Es handelt sich um die fachliche Lösung des zu realisierenden Systems und besteht mindestens aus dem Klassendiagramm. Weitere Diagrammtypen können helfen, das Fachkonzept besser darzustellen.

Welche Diagrammtypen sind auch noch hilfreich?

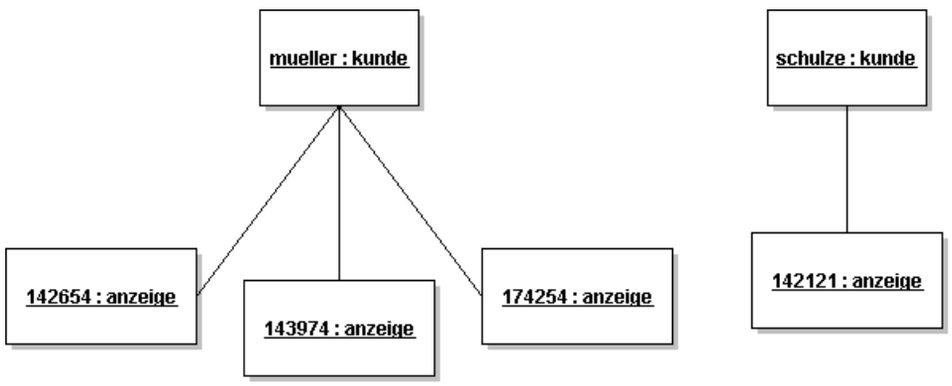
Neben dem Klassendiagramm in Form eines UML-Diagramms, kann es hilfreich sein, wenn man weitere Diagramme erstellt, um die Wirklichkeit auf die wesentlichen Bereiches des Problems herunter zu brechen.

Objektdiagramme[6]

Die Beziehungsstruktur einer Menge von Objekten kann man sehr übersichtlich durch ein Objektdiagramm darstellen. Dabei werden die Objekte in Form von Objektkarten (die oft auch nur den Objektbezeichner

[6] Objektdiagramme und Sequenzdiagramme lassen sich mit Violet (kostenloses Java-Programm) oder Microsoft Visio gut darstellen.

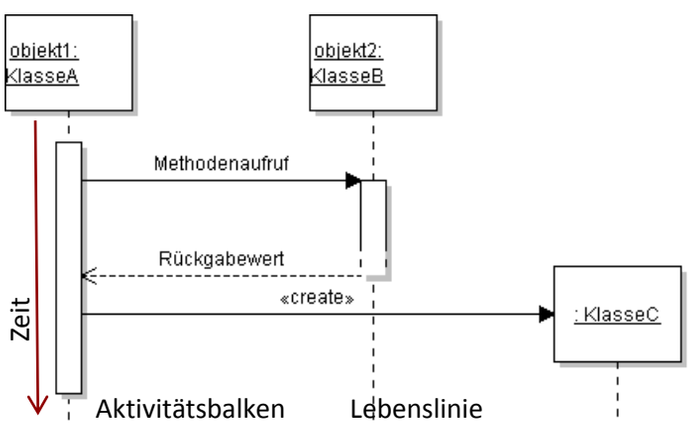
enthalten) dargestellt und die Beziehungen als Verbindungslinien dazwischen, die ggf. genau ein Objekt mit genau einem anderen Objekt verbinden. Falls es im Objektdiagramm eine Beziehung zwischen Objekten der gleichen Klasse gibt (ist hier nicht der Fall), dann nennt man eine solche Beziehung rekursiv. Im Klassendiagramm muss dann eine Verbindung dieser Klasse zu sich selbst eingezeichnet werden.

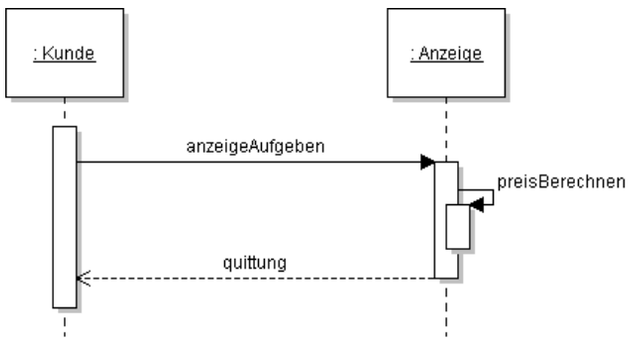


Sequenzdiagramme

Sequenzdiagramme dienen der Beschreibung von Interaktionen zwischen Objekten innerhalb eines bestimmten Anwendungsfalls. Sie zeigen den zeitlichen Ablauf bzw. die Reihenfolge von gegenseitigen Methodenaufrufen und –ausführungen. Obwohl Sequenzdiagramme üblicherweise nur typische Interaktionen zwischen Objekten darstellen und damit nur einen Ausschnitt des Systemablaufs zeigen, eignen sie sich gut zur Überprüfung eines Klassenmodells sowie zur Entwicklung der Struktur von Methoden.

In Sequenzdiagrammen werden in der Kopfzeile Objekte aufgereiht, von denen nach unten jeweils eine Zeitachse, auch Lebenslinie genannt, führt. Von einer solchen Zeitachse ausgehend zeichnet man für jeden Methodenaufruf einen Pfeil von aufrufenden zum aufgerufenen Objekt. Nach dem Ende der Ausführung der Methode wird ein Pfeil in der Gegenrichtung eingetragen und evtl. mit dem Rückgabewert der Methode versehen. Ist das Objekt aktiv, so wird dies durch ein Rechteck auf der Zeitachse kenntlich gemacht. Die Länge des Rechtecks zeigt an, wie lange die Aktivierung dauert. Falls nur interessant ist, zu welcher Klasse Objekte gehören, kann man ggf. die Objektbezeichner weglassen. Trifft ein horizontaler Pfeil auf den Kopf eines Objektes, so wird eine Konstruktormethode aufgerufen: ein Objekt wird neu erzeugt.





Für unser Inserenten- und Anzeigenverwaltungsprogramm kommen wir zunächst zu folgendem Sequendiagramm. Man erkennt, dass sich neue Fragen ergeben, z. B. ob die Rechnungsstellung in dem Programm mit verwaltet werden soll. Diese Fragen müssen mit dem Auftraggeber neu geklärt werden, ggf. müssen die Diagramme angepasst werden.

Zustandsdiagramm

Zur Beschreibung von Zustandsfolgen verwendet man Zustandsdiagramme. Der Zustand eines Objektes lässt sich durch seine aktuellen Attributwerte beschreiben. Ein Objekt kann damit im Laufe der Zeit viele verschiedene Zustände annehmen. Zustandsdiagramme spielen in der Automatentheorie eine große Rolle und werden daher im Lehrgang „Theoretische Informatik“ ausführlich besprochen.

Aufgaben:

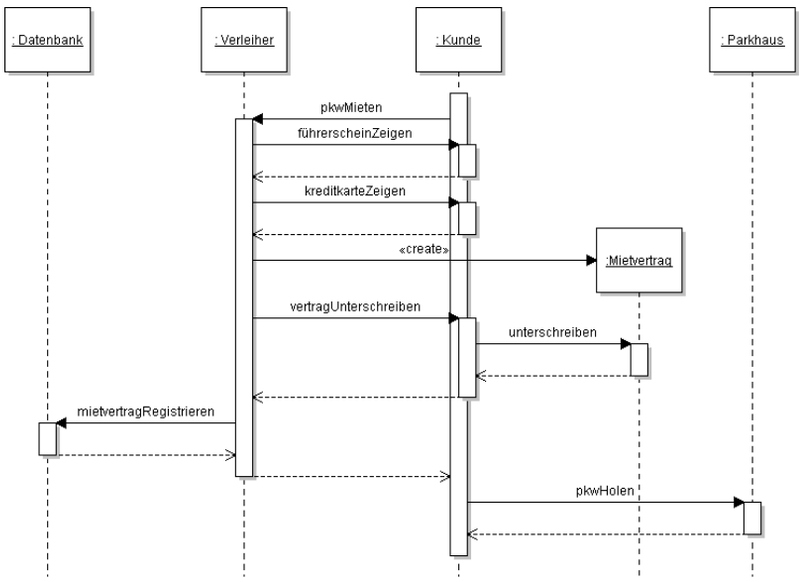
11. Majestrix ist der Chef des widerspenstigen gallischen Dorfes, zu dessen Bewohnern unter anderem die Krieger Asterix und Obelix sowie der Druide Miraculix gehören. Eine spezielle Rolle nimmt der Barde Troubadix ein, dessen künstlerisches Talent von seinen Stammesgenossen bekanntlich in keinster Weise gewürdigt wird. Nicht weniger wichtig sind die Frauen des Dorfes, wie beispielsweise Gutemine, die streitbare Gattin des Majestrix, oder Gelatine, die mit Orthopädix, dem Wirt des dörflichen Gasthofs „Zur frischen Brise“, verehelicht ist.

    a. Stelle in einem Objektdiagramm die genannten Persönlichkeiten des gallischen Dorfes und ihre Beziehungen zueinander dar. Es steht dir natürlich frei, weitere Charaktere deiner Wahl hinzuzufügen. Im Internet ist umfangreiches Zusatzmaterial zu finden.

    b. Erstelle aus dem Objektdiagramm ein Klassendiagramm, das die Gesellschafts-struktur des gallischen Dorfes beschreibt.

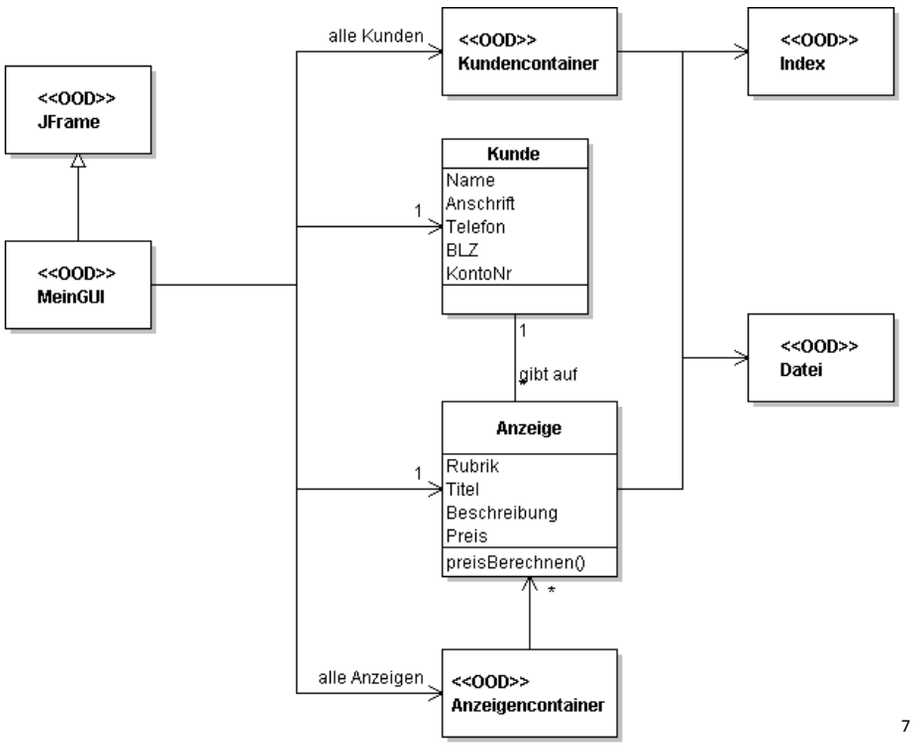
12. Das folgende Sequenzdiagramm zeigt den Ablauf beim Mieten eines Fahrzeuges.
    a. Gib diesen Ablauf in natürlicher Sprache wieder.
    b. Wie könnte das Klassendiagramm, das dem Ablauf zugrunde liegt, aufgebaut sein?
    c. Welche Methoden müssen die Klassen auf jeden Fall enthalten?
    d. Der Kunde wird mit seinem Mietwagen in einen Unfall verwickelt. Dabei spielt sich folgendes Szenario ab: *Der Kunde meldet dem Verleiher den Schaden und gibt den beschädigten Wagen der Verleihfirma zurück. Die Verleihfirma ihrerseits gibt die Schadensmeldung der zuständigen Versicherung weiter, die wiederum von dem Kunden einen Unfallbericht einholt. Da sich herausstellt, dass der Kunde am Unfall nicht schuld ist, übernimmt die Versicherung die Reparaturkosten, und die Verleihfirma gibt die Reparatur bei einer Werkstatt in Auftrag.*

       Erweitere das Sequenzdiagramm, so dass der geschilderte Ablauf dargestellt wird.



Das Objektorientierte Design (OOD) – Der Entwurf

Nachdem die objektorientierte Analyse abgeschlossen ist, muss nun im nächsten Schritt überlegt werden, wie man die OOA konkret in der gewählten Programmiersprache umsetzen kann. Aufgabe des Entwurfs ist es, die Anwendung auf einer Plattform unter den geforderten technischen Randbedingungen zu realisieren. Das OOD-Modell wird unter den Gesichtspunkten der Effizienz und Standardisierung konzipiert. Nun muss also vom Abstrakten zum Konkreten hin gedacht werden, dabei erhöht sich meist die Anzahl der Klassen um das Drei- bis Vierfache. Im OOD-Modell werden vor allem Klassen für die Oberflächengestaltung und die dauerhafte Datenspeicherung sowie Verwaltungsklassen für die Objekte hinzugefügt. Wesentliches Entwurfsziel ist die **Trennung von Fachkonzept, Benutzeroberfläche und Datenhaltung.** Es wird die so genannte Drei-Schicht-Architektur angestrebt. Für unsere Anzeigenverwaltung ergibt sich folgendes OOD:



[7]

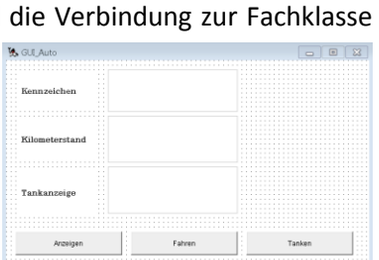
Die Darstellung ist trotz des Klassenzuwachses immer noch vereinfacht. Später wird man allein für die Benutzeroberfläche wesentlich mehr Klassen benötigen.

Das Design, also das Konzept für die Programmierung, muss die Möglichkeiten der gewählten Programmiersprache berücksichtigen. Hierzu gehören die unterschiedlichen Möglichkeiten der Darstellung und der Datenspeicherung (welche Klassenbibliotheken stehen zur Verfügung für die Darstellung sowie Datenbanken), sowie die Mittel der Programmiersprache (Einfach- oder Mehrfachvererbung).

Die Erstellung des OOD-Modells vor der Programmierung sollte auf keinen Fall vernachlässigt werden, da man sonst bei der Programmierung den Überblick über die Klassen und ihre Zusammenhänge verliert. Weitere Vorteile eines guten Entwurfs liegen in der Wiederverwendbarkeit und der Möglichkeit veränderte Anforderungen in den Entwurf und somit die Programmierung einfließen zu lassen. Um diese Vorteile nutzen zu können, sollten die einzelnen Klassen möglichst einfach sein, d. h. komplexere Klassen sollten zerlegt werden, und die einzelnen Klassen sollten abgeschlossen sein, d. h. sie sollten möglichst wenig von den anderen Klassen wissen. Hierauf werden wir später noch eingehen.

Aufgaben:

13. Für die objektorientierte Analyse aus Aufgabe 6 (Autos) soll nun ein passendes GUI-Formular erstellt werden. Deklariere ein GUI_Auto. Über dieses Auto wird die Verbindung zur Fachklasse Auto hergestellt.



[77] Die Bedeutung der Pfeilspitze von MeinGui zu JFrame wird später erläutert.

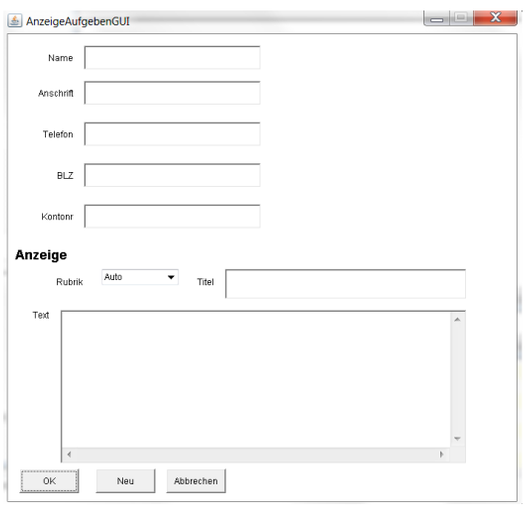
Objektorientierte Programmierung (OOP)

Das OOD-Modell wird nun mit einer objektorientierten Programmiersprache (in unserem Fall Java) implementiert. **Um die Trennung zwischen dem Fachkonzept und der Benutzeroberfläche deutlich zu machen, sollte man sämtliche Ausgabeanweisungen aus den Fachklassen entfernen. So kann man die Benutzeroberfläche ändern, ohne die Funktion der Ausgabe innerhalb jeder Fachklasse neu programmieren zu müssen. Die Fachklassen sollen nur das modellieren und realisieren, was fachlich von Bedeutung ist.** Der Anwender des Programms soll nur über die Benutzeroberfläche agieren können. Hierzu benötigen wir eine GUI-Klasse (siehe OOD). Damit die GUI-Klasse nun die Fachklasse kennt, muss im Variablenbereich der GUI-Klasse eine Anweisung zum Erzeugen eines Fachobjekts ergänzt werden.

```
private Button bu_Abbrech
private Kunde neukunde;
private Anzeige inserat;
// Ende Attribute
```

1. Schritt: Wir benötigen für unser Inserenten- und Anzeigenverwaltungsprogramm die beiden Fachklassen Kunde und Anzeige sowie eine Benutzeroberfläche.

2. Schritt: Die GUI-Klasse muss Kunden und Anzeigen erzeugen können. Da die Attribute von Objekten immer privat sind (geheim), kann die GUI-Klasse nicht direkt auf die Attribute zugreifen. Die GUI-Klasse muss mit dem Konstruktor der Fachklassen arbeiten.




```
public void bu_OK_ActionPerformed(ActionEvent evt) {
    String Name, Add,Tnr,BLZ,Knr;      //für Kunden notwendig
    String Rubrik,Titel,Beschreibung; //für Anzeige notwendig
    Name = tFKundenname.getText();
    Add = tFAnschrift.getText();
    Tnr = tFTelefon.getText();         //einlesen der Eingaben für Kunden
    BLZ = tFBlz.getText();
    Knr = tFKontonr.getText();
    neukunde = new Kunde(Name,Add,Tnr,BLZ,Knr);    //erzeugen des Neukunden
    Rubrik = chRubrik.getSelectedItem();
    Titel = tFTitel.getText();              //einlesen der Eingaben für Anzeige
    Beschreibung = tAText.getText();
    inserat = new Anzeige(Rubrik,Titel,Beschreibung);//erzeugen der neuen Anzeige
} // end of bu OK ActionPerformed
```

3. Schritt: Die Assoziationen zwischen den Fachklassen müssen implementiert werden.
   a. **Die unidirektionale Assoziation**:
      Da es sich bei der Assoziation anzeigeAufgeben um eine gerichtete Assoziation handelt, muss die Klasse Kunde (der Nutzer der Verbindung) um ein Attribut Inserat vom Typ Anzeige ergänzt werden. Um auf eine Anzeige zugreifen zu können, benötigt man die drei folgenden Methoden:

Herstellen einer Beziehung

Abfragen einer Beziehung

Aufheben einer Beziehung

```
// Anfang Methoden
public void setAnzeige(Anzeige einInserat) {
    Inserat = einInserat;
}
public Anzeige getAnzeige(){
  return Inserat;
}
public void removeAnzeige(){
  Inserat = null;
}
// Ende Methoden
```

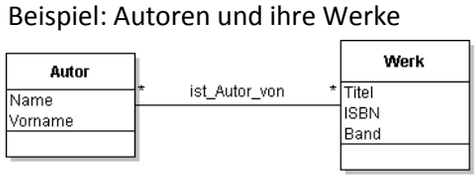
Man hat sich mit den Java-Beans-Konventionen[8] darauf geeinigt, dass die Methoden Namen der Form setXxxx und getXxxx bekommen. Sie haben auch die Aufgabe zu kontrollieren, dass keine ungültigen Datenwerte verwendet werden.

b. **Die bidirektionale Assoziation**:

Die Implementierung ist etwas aufwändiger. Es gibt zwei Möglichkeiten:

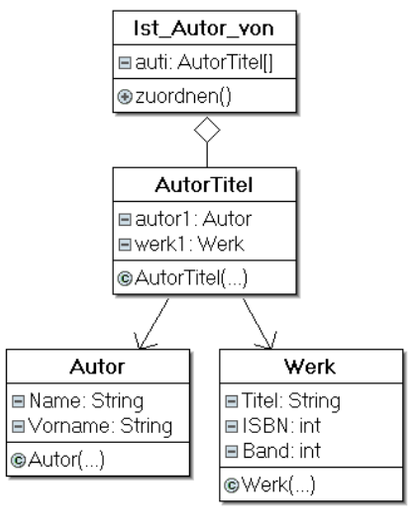
- Man löst die Beziehung in zwei getrennte Assoziationen auf und implementiert diese wie in a. beschrieben.
- Man realisiert die bidirektionale Assoziation mit Hilfe einer speziellen Assoziationsklasse. Im Wesentlichen besteht ein solches „Assoziationsobjekt“ aus einem Feld, dessen Elemente (eigentlich Objekte) jeweils eine Zuordnung der Assoziation darstellen, indem sie je eine Referenz auf die beiden einander zugeordneten Objekte enthalten.[9]

  Beispiel: Autoren und ihre Werke



Die Assoziation ist_Autor_von ist bidirektional. Es gibt Autoren, die mehrere Bücher geschrieben haben, z. B. Goethe. Es gibt aber auch Bücher, die von mehreren Autoren geschrieben wurden, z. B. Lehrbücher.

Wir benötigen also die beiden Fachklassen Autor und Werk, eine Assoziationskasse Ist_Autor_von und eine Klasse AutorTitel.



[8] Der Name bedeutet eigentlich Kaffeebohne und meint hier den Grundstoff, aus dem alle guten Java-Programme sind.

[9] Hilfe! Ich verstehe nur Bahnhof! Macht nichts. Jetzt kommt ein Beispiel.

Die Bedeutung des Beziehungssymbols werden wir im Laufe dieses Schuljahres noch kennen lernen.

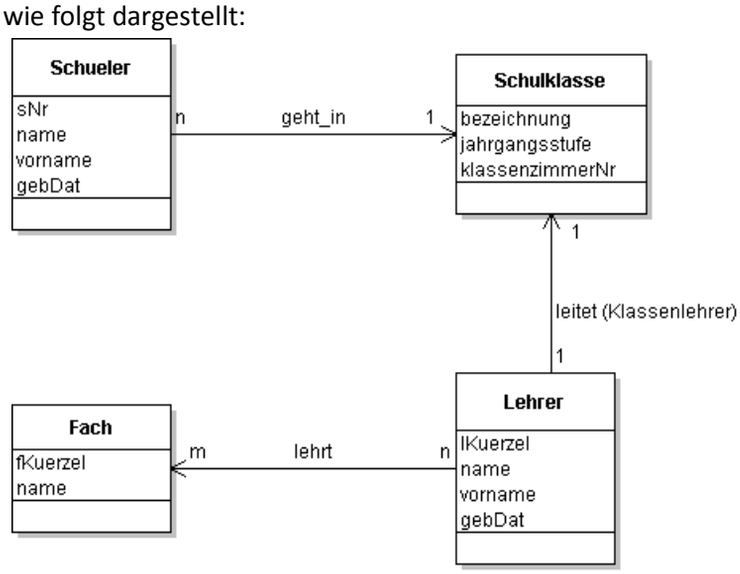
```
public class Ist_Autor_von {

  // Anfang Attribute
  private AutorTitel[] auti = new AutorTitel[100];
  // Ende Attribute

  // Anfang Methoden
  public void zuordnen() {
    auti[0] = new AutorTitel(new Autor("Goethe","Johann Wolfgang"), new Werk("Faust",342));
    auti[1] = new AutorTitel(new Autor("Goethe","Johann Wolfgang"), new Werk("Die Leiden des...",937));
    auti[2] = new AutorTitel(new Autor("Lindgren","Astrid"), new Werk("Pippi Langstrumpf",437));
  }

  // Ende Methoden
} // end of Ist_Autor_von
```

```
public class AutorTitel {

  // Anfang Attribute
  private Autor autor1;
  private Werk werk1;
  // Ende Attribute

  public AutorTitel(Autor autor1, Werk werk1) {
    this.autor1 = autor1;
    this.werk1 = werk1;
  }

  // Anfang Methoden
  // Ende Methoden
} // end of AutorTitel
```

Aufgaben:

4. Die Autoaufgabe als Programm:
   a. Implementiere die Fachklasse Auto und die zugehörige GUI (siehe Aufgabe 6 und 9) sowie die notwendigen Methoden.
   b. Lass dir im JavaEditor das UML-Diagramm anzeigen. Was stellst du fest?

5. Die Beziehungen zwischen den Schülern einer Schulklasse, den Lehrern und den einzelnen Fächern werden wie folgt dargestellt:



a) Ergänze die Beziehung „unterrichtet“ (Lehrer unterrichtet eine Schulklasse).
b) Erweitere das Klassendiagramm um die Klasse Schule und die dazugehörigen Assoziationen. Gib die Klassen, die zur Implementierung der einzelnen Assoziationen

notwendig sind, an. Weshalb könnte es sinnvoll sein, auch die Beziehung zwischen Schülern und der Schulklasse durch eine Assoziationsklasse zu implementieren?

c) Weshalb ist es sinnvoll, eine m:n-Assoziation zwischen den Klassen Schueler und Fach einzuführen? Ergänze das Klassendiagramm entsprechend.

d) Die Implementierung der Schulbeziehungen sollen nun arbeitsteilig durchgeführt werden. Suche dir ein oder zwei Mitschüler, mit denen du gemeinsam die Schulbeziehungen programmieren möchtest. Teilt euch die notwendigen Arbeiten auf:
   - Die fünf Klassen Schule, Schueler, Schulklasse, Lehrer und Fach sollen arbeitsteilig programmiert werden.
   - Erzeugt eine hinreichende Anzahl von Objekten der Klassen Schueler, Schulklasse, Lehrer und Fach in jeweils einer Methode der Klasse Schule. Auch hier könnt ihr arbeitsteilig vorgehen.
   - Implementiert nun die Assoziationen arbeitsteilig.
   - Nun fehlen noch ein paar Methoden der Klasse Schule:
     - eine Methode, um die Daten eines bestimmten Schülers auszugeben
     - eine Methode, um alle Schüler einer bestimmten Schulklasse zu ermitteln
     - eine Methode, um alle Fächer, die in einer bestimmten Klasse unterrichtet werden, zu bestimmen
     - ZUSATZ: eine Methode, um einen bestimmten Schüler zu löschen. Dabei soll sichergestellt werden, dass alle Assoziationen, an denen dieser Schüler beteiligt ist, gelöscht werden.

e) Wie muss das Klassendiagramm erweitert werden, damit die Endnote der Schüler in den einzelnen Fächern erfasst werden können?

f) Programmiert die Erweiterung zur Erfassung der Endnoten aus Teilaufgabe e).

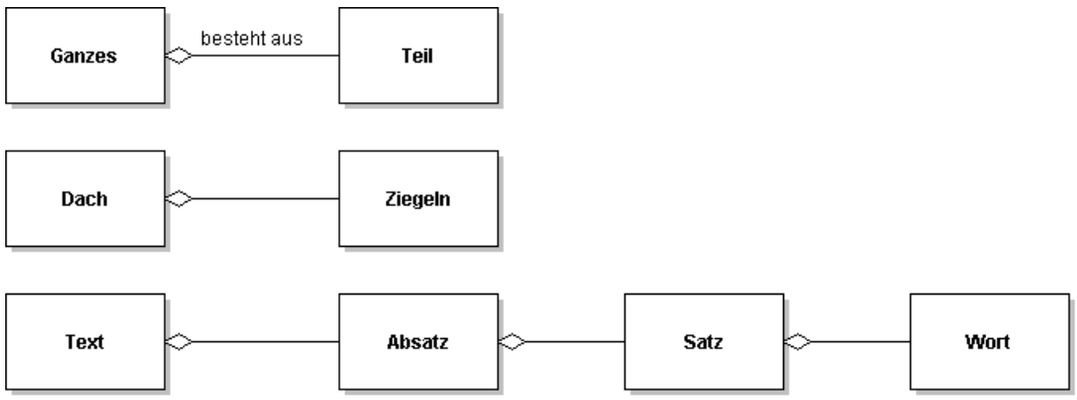
g) Ergänzt in der Klasse Schule die folgenden Methoden:
   - eine Methode, um alle Endnoten eines bestimmten Schülers auszugeben
   - eine Methode, um den Zeugnisdurchschnitt eines bestimmten Schülers zu berechnen
   - eine Methode, um den Durchschnitt einer bestimmten Klasse in einem bestimmten Fach zu berechnen.

Die Aggregation

Ein Spezialfall der Assoziation ist die Aggregation. Sie wird auch als „Enthält“-Beziehung oder „hat“-Beziehung bezeichnet, da sie ausdrückt, dass eine Klasse eine andere Klasse enthält. Unter einer Aggregation versteht man also eine Zusammensetzung eines Objektes aus einer Menge von Einzelteilen. So besteht ein Dach aus Ziegeln und ein Text aus Absätzen, wobei jeder Absatz wiederum aus einzelnen Sätzen besteht.

In der UML-Darstellung wird die Aggregatklasse mit einer Raute versehen. Die Raute symbolisiert das Behälterobjekt, in dem die Teile gesammelt werden. Navigierbar muss diese Beziehung immer von Ganzen zum Teil sein. Auf die Pfeilspitze wird verzichtet.

Die Aggregatklasse muss bei der Programmierung um ein Attribut ergänzt werden, in dem mehrere Objekte gespeichert werden können. Hierzu bietet sich in Java ein Feld (oder ein Vektor) an.



Auch eine Klasse besteht aus mehreren Schülern, so dass wir auch hier eine Aggregation haben. (siehe Aufgabe 15).

Wenn ein Aggregat nur dann existiert, wenn mindestens ein Teil vorhanden ist, so spricht man streng genommen von einer **Komposition**. Kompositionen werden im UML-Diagramm durch eine ausgefüllte Raute dargestellt.

Ein Buch ohne Seiten gibt es nicht!

Es ist nicht immer einfach zu entscheiden, ob eine Aggregation oder Komposition vorliegt. Für die Modellierung und Implementierung spielt der Unterschied auch keine so große Rolle. Im Landes-abitur wird daher auf die Komposition verzichtet. Betrachten wir das Verhältnis zwischen Motor und Auto. Das Fachkonzept könnte diese Beziehung als Komposition darstellen. Denkt man aber an den Austauschmotor wird deutlich, dass Motoren nicht unbedingt existenzabhängig vom Auto sind. Es lässt sich darüber streiten, ob die Beziehung eine Aggregation oder Komposition ist.

Kapselung

Bisher haben wir hingenommen, dass bestimmte Datenfelder, Methoden und auch Klassen als public bzw. private deklariert wurden. Klassen werden meist als public deklariert. Bisher haben wir alle Attribute als privat und alle Methoden als public deklariert. Ist das sinnvoll? Welche Bedeutung haben diese Zugriffsmodifikatoren?

Zugriffsmodifikator private und public

Wird ein Attribut als privat markiert, so können nur Objekte der gleichen Klasse seinen Wert direkt lesen oder verändern. Eine private Methode kann nur von Objekten der gleichen Klasse aufgerufen werden. Auf öffentliche (public) Attribute und Methoden kann jedoch jedes Objekt zugreifen und somit Attributwerte lesen und verändern und Methoden aufrufen. Die Menge der öffentlichen Attribute einer Klasse bezeichnet man auch als Schnittstelle der Klasse. Die Schnittstelle einer Klasse beschreibt also alle Komponenten, die ihre Objekte zur Benutzung durch Objekte anderer Klassen zur Verfügung stellen.

Es wird daher dringend empfohlen, bei jeder Klasse genau zu überlegen, welche Datenfelder und Methoden „von außen“ direkt verändert werden müssen und welche nur innerhalb der Klasse benötigt werden. Alles, was nicht zwingend von außen verwendet werden muss, sollte auch nur im Inneren

veränderbar sein, d. h. privat. Diese Vorgehensweise nennt man Kapselung (oder auch Data Hiding). Die Kapselung bietet folgende Vorteile:

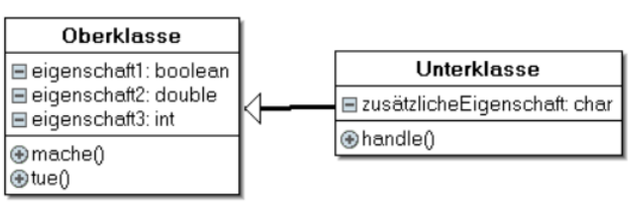
- Sicherheit, dass nur gültige Attributwerte vorhanden sind
- Einfache und klare Schnittstellen für die Verwendung dieser Klassen bzw. ihrer Objekte
- Weitreichende Unabhängigkeit der internen Programmierung einzelner Klassen von der Programmierung anderer Klassen
- Leichtere Wartung und Optimierung
- Weitgehende Vermeidung von Programmierfehlern beim Zusammenspiel der verschiedenen Klassen

Vererbung

> „Was du bist, hängt von drei Faktoren ab: Was du geerbt hast, was deine Umgebung aus dir machte und was du in freier Wahl aus deiner Umgebung und deinem Erbe gemacht hast.“ (Aldous Huxley 1894 – 1963, britischer Schriftsteller)

Den Begriff Vererbung kennst du aus der Biologie (Stammbaum von Artverwandten) oder aus dem täglichen Leben (Familienstammbaum). Die Idee der Vererbung nutzt man auch in objektorientierten Programmiersprachen. Bei allen Arten der Vererbung ist das zugrundeliegende Prinzip, dass bestimmte Eigenschaften von jemand an jemand anderen vererbt werden, z. B. vererbt ein Vater seine Haarfarbe an seine Tochter. Wesentliches Element der Objektorientierung sind die Klassen mit ihren Attributen und Methoden, die vererbt werden können. So wie der Vater an seine Tochter Eigenschaften vererbt, so vererbt die Oberklasse an ihre Unterklasse Attribute und Methoden. Der Unterschied zur biologischen Vererbung liegt darin, dass Unterklassen in der Regel nur von einer Oberklasse erben. Dabei werden grundsätzlich erst mal alle Eigenschaften und Methoden vererbt.

Im UML-Diagramm wird die Vererbung dargestellt, indem von der Unterklasse zur Oberklasse ein durchgezogener Pfeil mit unausgefüllter, dreieckiger Spitze gezogen wird. Der Pfeil zeigt auf die Oberklasse.



Die Unterklasse[10] übernimmt sämtliche Attribute und Methoden der Oberklasse[11]. Diese müssen im UML-Diagramm nicht noch einmal aufgeführt werden. Man sieht es ja an der Beziehungsart. Die Unterklasse kann aber noch zusätzliche Eigenschaften und weitere Methoden besitzen. Die Bildung einer Unterklasse nennt man **Spezialisierung**. Fasst man mehrere Klasse mit gleichen Attributen und Methoden zu einer

[10] Manchmal auch Subklasse genannt.
[11] Manchmal auch Superklasse genannt.

allgemeineren Oberklasse zusammen, nennt man diesen Vorgang **Generalisierung**[12]. Durch diese Vorgehensweise kann man eine hierarchische Klassenstruktur erhalten.

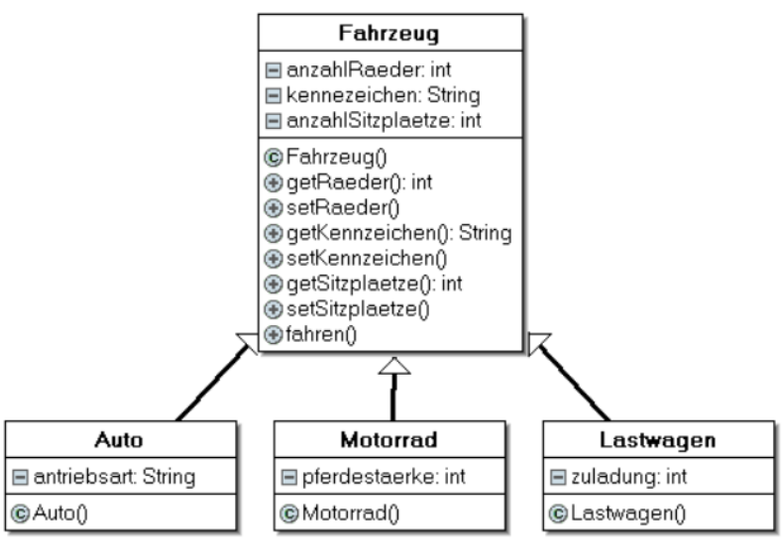
Während man bei der Erstellung des Fachkonzeptes, also bei der OOA, den Begriff Generalisierung verwendet, benutzt man in den Bereichen des OOD und OOP eher den Begriff Vererbung. Es gilt:

- Jedes Attribut der Oberklasse wird an die Unterklasse vererbt. Der Attributwert wird nicht vererbt.
- Alle Methoden, die auf Objekte der Oberklasse anwendbar sind, sind auch auf die Objekte der Unterklasse anwendbar.

Spezialisierung von Klassen

Im Allgemeinen kann man immer dann auf die Möglichkeit der Vererbung zurückgreifen, wenn man umgangssprachlich formulieren kann, dass „A ein B ist“. Man spricht daher auch von einer „ist“-Beziehung.

Beispiel: Ein Auto ist ein Fahrzeug. Ein Motorrad ist ein Fahrzeug. Ein Lastwagen ist ein Fahrzeug. Hier ist Fahrzeug der Oberbegriff, also die Oberklasse und die anderen Fahrzeugarten stellen jeweils eine Unterklasse dar.



Implementierung einer Spezialisierung

Um im Programmcode festzulegen, von welcher Klasse geerbt wird, verwendet man das **Schlüsselwort extends**.

Die **Konstruktoren** werden nicht vererbt, so dass sie neu in den Unterklassen definiert werden müssen. Der Konstruktor der Unterklasse muss zunächst den Konstruktor der Oberklasse aufrufen, dies geschieht mit super(..). Über die Art und Anzahl der Parameter wird festgelegt, welcher Konstruktor der Oberklasse aufgerufen wird. Erfolgt in einem Konstruktor kein Aufruf von super(..) so fügt der Compiler automatisch und damit implizit einen Aufruf des standardmäßigen Konstruktor super() ohne Argumente ein. Dies geschieht zum Beispiel bei allen Klassen an der Spitze der Klassenhierarchie.

[12] Generalisierung bedeutet Verallgemeinerung.

```
public class auto extends Fahrzeug {

  // Anfang Attribute
  private String antriebsart;
  // Ende Attribute

  public auto(int raeder,String kennzeichen, int sitzplaetze, String antriebsart) {
    super(raeder,kennzeichen,sitzplaetze);  //Konstruktor der Oberklasse aufrufen
    raeder = 4;
    this.antriebsart = antriebsart;
  }
```

Die Anweisung super(raeder, kennzeichen, sitzplaetze) ruft den Konstruktor der Oberklasse Fahrzeug mit den angegebenen Parametern auf. Sie ist gleichbedeutend mit dem Aufruf fahrzeug(raeder,kennzeichen,sitzplaetze).

Das **Schlüsselwort super** ohne Klammern liefert eine Referenz auf ein Objekt der Oberklasse Fahrzeug zurück. Mit super.fahren() kann z. B. von einem Objekt der Klasse Fahrrad aus die Methode fahren der Oberklasse Fahrzeug aufgerufen werden. Mit dem **Schlüsselwort this** kann man explizit auf die Attribute und Methoden der eigenen Klasse zugreifen.

Zugriffsmodifikator protected: Private Attribute bzw. Methoden können trotz Vererbung nicht von Objekten einer Unterklasse benutzt werden. Durch die Deklaration als *geschützt* mit dem Zugriffsmodifikator protected kann man dieses Problem beheben, ohne dass man mit public den Zugriff auch für alle anderen Klassen öffnen müsste.

Abstrakte Klassen

Wenn man eine Hierarchie von vererbten Klassen aufbaut, kann es sein, dass von der Oberklasse selbst gar keine Objekte erzeugt werden sollen, d. h. die Klasse selbst dient nur als Vorlage für andere Klassen. Sie definieren über normale Methoden eine Grundfunktionalität und durch abstrakte Methoden eine einheitliche Schnittstelle, die abgeleitete Klassen implementieren müssen.

Die Fahrzeuge, die erzeugt werden, sind immer von einer bestimmten Art, so dass sie nicht ganz allgemein mithilfe der Klasse Fahrzeuge erzeugt werden. Die Klasse Fahrzeuge könnte daher auch als abstrakte Klasse vorgesehen werden. Diese Entscheidung muss im Vorfeld getroffen werden, damit bereits in der Modellierungsphase auf diesen Aspekt eingegangen werden kann. Im oberen Entwurf ist die Klasse Fahrzeuge nicht abstrakt modelliert.

Welche Auswirkungen hätte eine abstrakte Klasse Fahrzeuge? Wie würde sich diese Entscheidung auf die Implementierung auswirken?

- Die Klasse Fahrzeug müsste als abstrakt deklariert werden. (siehe Abb.)
- Von der abstrakten Klasse Fahrzeuge können keine Objekte erzeugt werden.

```
abstract class Fahrzeug {

  // Anfang Attribute
  protected int raeder;
  protected String kennzeichen;
```

- Abstrakte Klassen müssen als solche gekennzeichnet werden, wenn sie eine abstrakte Methode enthält.

- Abstrakte Methoden enthalten nur die Methodenköpfe, d. h. es ist festgelegt, welche Parameter mit der Methode übergeben werden müssen, ob ein Wert zurückgegeben wird und ggf. welchem Datentyp der Rückgabewert entspricht. Allerdings ist keine Funktionalität gegeben.

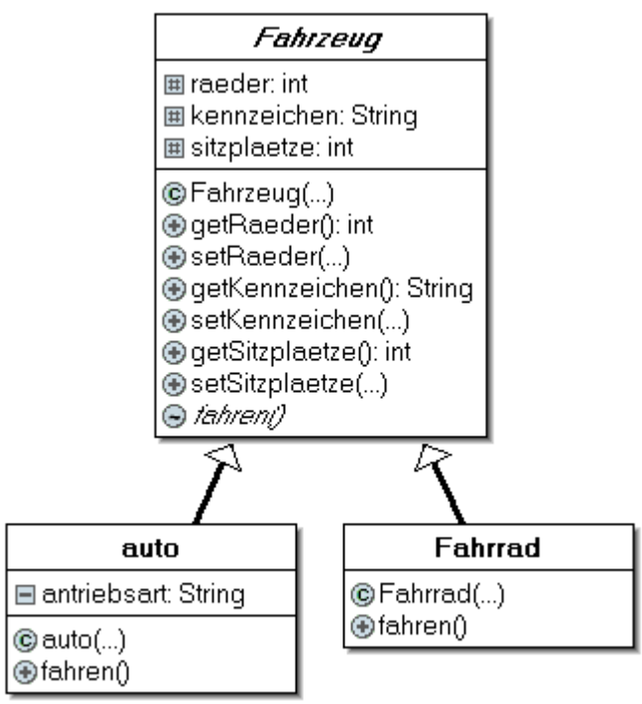
- Die Klasse Fahrzeuge würde weiterhin die konkreten get- und set-Methoden behalten. Die Methode fahren() wird abstrakt deklariert. Der Methodenrumpf ist leer.

```
abstract void fahren();
```

- In allen abgeleiteten Klassen (Unterklassen) muss die Methode fahren implementiert werden. Die Methode wurde abstrakt deklariert, weil beim Fahren eines Autos z. B. Benzin oder Diesel verbraucht wird, beim Fahrradfahren werden jedoch Kalorien verbrannt.

```
public void fahren(){
  //hier fehlt noch der Quelltext
}
```

- Im UML-Diagramm wirden die abstrakten Methoden und der abstrakte Klassenname kursiv geschrieben.



Eine abstrakte Klasse kann abstrakte Methoden anbieten. Sofern eine Unterklasse nicht selbst abstrakt sein soll, müssen diese Methoden überschrieben werden. Eine abstrakte Klasse kann nicht instanziiert werden.

Vor- und Nachteile der Vererbung

Wo liegt nun aber der Vorteil der Spezialisierung. Man könnte doch auch einfach alle Fahrzeugklasse getrennt voneinander erstellen. Unser Programm würde dann aus zahlreichen Klassen bestehen, die viele gemeinsame Attribute und Methoden aufweisen. Da eine neue Klasse recht leicht zu erstellen ist, stellt dies zunächst kein Problem dar. Wie sieht es aber aus, wenn wir an den vorhandenen Klassen etwas ändern wollen? Wir wollen allen Klassen das Attribut preis hinzufügen. Wenn wir die Klassen alle unabhängig voneinander erstellt haben, dann müssen wir nun in jeder einzelnen Klasse das Attribut preis hinzufügen.

Dies ist einerseits recht mühsam und kann zu Tippfehlern führen und andererseits müssen wir aufpassen, dass wir in allen Klassen die gewünschte Änderung durchführen. Evtl. müssen auch noch Änderungen in Methoden vorgenommen werden. Hat man die verschiedenen Fahrzeugklassen allerdings mithilfe der Vererbungsstrategie erstellt, so kann man das Attribut preis in der Oberklasse Fahrzeuge einfügen.

Man sollte die Vererbung nicht überbewerten. Vieles lässt sich mittels Assoziationen und Aggregationen modellieren. Vererbung sollte man wirklich nur dann einsetzen, wenn eine Generalisierungsstruktur tatsächlich vorliegt. Beispielsweise darf man eine Rechteck-Klasse nicht von einer Quadrat-Klasse ableiten. Zwar benötigt eine Quadrat-Klasse nur eine Seitenlänge, und die davon abgeleitete Rechteck-Klasse könnte diese Seitenlänge erben und eine weitere Seitenlänge ergänzen, aber ein Rechteck ist kein spezielles Quadrat! Deshalb sollte man sich immer vergewissern, ob es sich bei der Beziehung zwischen den Klassen tatsächlich um eine „ist“-Beziehung handelt.

Diese Überlegungen gehören in den Bereich der objektorientierten Analyse. Das Ziel ist mit Hilfe von Klassen die Realität möglichst genau abzubilden. Also kann man immer dann über Ober- und Unterklassen nachdenken, wenn man zwischen den realen Objekten eine „ist“-Beziehung erkennen kann.

Subtyping

Wie sieht es aus, wenn Objekte ganz anderer Klassen allgemein auf verschiedene vererbte Klassen zugreifen müssen. Muss man dann für jede einzelne Klasse eine extra Methode schreiben? Das wäre sehr umständlich. Durch Subtyping (Ersetzbarkeit) können Objekte einer Unterklasse immer dort eingesetzt werden, wo ein Objekt der Oberklasse erwartet wird. Umgekehrt funktioniert das selbstverständlich auch.

Ein Beispiel: Auf einem Parkplatz dürfen alle Arten von Fahrzeugen parken, dann genügt es, wenn die Klasse Parkplatz die Methode parken(Fahrzeug fNr) besitzt. Mit dieser Methode können jetzt Autos, Fahrräder, Motorräder und Lastwagen geparkt werden. Soll eine bestimmte Fahrzeugart ausgeschlossen werden, muss dies in der Methode entsprechend programmiert werden.

Vorteile der Vererbung:

- bessere Wartbarkeit
- weniger fehleranfällig
- keine Code-Duplizierung
- übersichtlicher und lesbarer Klassendiagramme

Polymorphismus

Manchmal kann es sein, dass die Methoden einer Oberklasse sich nicht im vollen Umfang auf die Methode der Unterklasse übertragen lassen. Dann wird der Methodenkopf beibehalten, aber der Methodenrumpf wird überschrieben, d. h. die Signatur der Methode bleibt die gleiche, die Anweisungen im Rumpf werden neu programmiert. Diesen Vorgang nennt man Polymorphie oder Überlagerung.

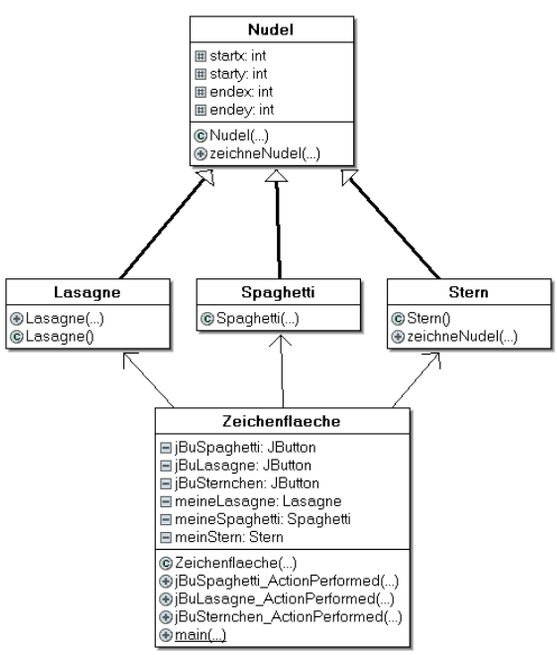
Diese Möglichkeit muss man zum Beispiel verwenden, wenn man die Klasse Fahrzeug nicht abstrakt deklariert hat, die Methode fahren() in der Klasse Fahrzeug so implementiert wurde, dass Benzin oder Diesel beim Fahren verbraucht wird. In der Klasse Fahrrad muss diese Methode nun überschrieben werden. Liegt nun ein Objekt meinDrahtesel vor und die Methode meinDrahtesel.fahren() wird aufgerufen, so wird die Methode der Klasse Fahrrad aufgerufen.

Aufgaben:

6. In einem Geschirrschrank können tiefe und flache Teller, Gläser, Kaffeebecher, Teetassen, Suppentassen, Unterteller für Tassen und Müslischalen gelagert werden. Alle Geschirrteile haben einen Durchmesser, eine Höhe sowie ein Gewicht. Sie unterscheiden sich durch ihre Funktion: Getränkeaufnahme, Nahrungsaufnahme, Sonstiges. Alle Geschirrteile lassen sich stapeln.
    a. Modelliere ein Entwurfsdiagramm für die verschiedenen Geschirrteile unter Berücksichtigung des Konzepts der Vererbung. Die oberste Oberklasse soll Geschirr sein.
    b. Erläutere das Konzept der Vererbung sowie dessen Vorteile.
    c. Überlege dir, in welcher Klasse du die get- und set-Methoden jeweils realisieren möchtest. Begründe deine Entscheidung.
    d. Implementiere die Klassen in Java. Vergiss nicht den Konstruktor.
    e. Gegeben ist die Klasse Geschirrschrank. Erläutere, warum mit der Methode public void einraeumen(Geschirr pgeschirr) dieser Klasse auch z. B. Teller eingeräumt werden können.
    f. Entwirf eine Benutzeroberfläche, in der angezeigt wird, wie viel Geschirr von der jeweiligen Sorte im Schrank steht. Zusätzlich soll Geschirr ein- und ausgeräumt werden können. Beachte, dass der Schrank eine gewisse Höhe hat.
    g. Schicke deine Lösung (von allen Teilaufgaben) per Mail an mich.

7. Ein objektorientiertes Programm zur Flächenberechnung von Rechtecken, Dreiecken und Kreisen soll aus den Klassen Flaeche, Dreieck, Rechteck, Kreis bestehen.
    a. Entwirf ein Klassendiagramm für die Flächenberechnung.
    b. Implementiere das Flächenberechnungsprogramm aus dem Klassendiagramm.

8. Das Nudelzeichenprogramm: Mit einem objektorientierten Programm sollen verschiedene Nudelsorten gezeichnet werden.
   Die Benutzeroberfläche soll drei Schaltflächen enthalten, so dass man zwischen Spaghetti, Lasagne und Sternchen auswählen kann. Zusätzlich muss eine Zeichenfläche vorhanden sein.
   Das Klassendiagramm sieht so aus:



Die Standardnudel hat immer einen Start- und einen Endpunkt mit x- und y-Wert und sie braucht eine Methode, um gezeichnet zu werden. Die Spaghetti entspricht ziemlich genau einer Standardnudel. Der Quelltext für Nudel lautet:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;


public class Nudel {

  // Anfang Attribute
  protected int startx;
  protected int starty;
  protected int endex;
  protected int endey;
  // Ende Attribute

  public Nudel(int sx, int sy, int ex, int ey) {
    this.startx = sx;
    this.starty = sy;
    this.endex = ex;
    this.endey = ey;
  }

  // Anfang Methoden
  public void zeichneNudel(Graphics g, String s) {
    g.setColor(Color.yellow);
    g.drawLine(startx,starty,endex,endey);
    g.drawString(s, startx, 20+endey);
  }

  // Ende Methoden
} // end of Nudel
```

Sortierverfahren

Eine gute Erklärung mit Veranschaulichung findet man auf: http://www.matheprisma.uni-wuppertal.de/Module/Sortieren/index.htm.

Wie sortieren eigentlich Menschen?

Ständig müssen wir im Alltag Dinge sortieren, z. B. die Post, beim Kartenspiel, die CDs im Regal, die Wäsche in den Schrank, welche Aufgaben muss ich zuerst erledigen,… Ein kleiner Praxistest soll dir zeigen, wie du welche Strategien du persönlich anwendest, wenn du etwas sortieren musst.

Praxistest Teil 1:

a) Du bekommst Länderkarten von mir. Wähle beliebig 10 Karten aus und lege sie unsortiert auf den Tisch. Sortiere die Karten entsprechend ihrer Fläche. Notiere dir, wie du beim Sortieren vorgegangen bist.

b) Sortiere nun die Karten nach der höchsten Erhebung. Ist dir noch etwas beim Sortieren aufgefallen?

c) Bildet Dreiergruppen. Führt euch jetzt gegenseitig euer Sortierverfahren vor. Beachtet dabei folgendes:

   a. Jeder übernimmt eine Rolle.
   b. Einer ist der **Sortierer**. Er sortiert nacheinander 5, 10 und 20 Karten nach der Einwohnerzahl.
   c. Einer ist der **Protokollant**. Beobachte und protokolliere genau, **wie** der Sortierer vorgeht. Folgende Fragen können dir helfen: Verfolgt der Sortierer eine bestimmte Strategie? Wie lange braucht er/sie, um eine bestimmte Karte an die richtige Position zu bringen? Was bereitet ggf. Probleme beim Sortieren?
   d. Die letzte Rolle ist der **Statistiker**: Stoppe die Zeit, die dein Mitschüler zum Sortieren braucht. Lege ein Diagramm Anzahl der Karten -> Zeit in Sekunden an und halte die Sortierzeiten darin fest. Beschreibe die Entwicklung der Sortierzeiten mit zunehmender Kartenanzahl.
   e. Tauscht die Rollen!

Wie sortiert ein Computer?

Für einen Computer sind die zu sortierenden Objekte Daten, die er mit keinem Vorwissen verknüpfen kann, so weiß der Computer nicht, dass Luxemburg bzgl. seiner Fläche recht klein ist und Deutschland im Vergleich ziemlich viele Einwohner hat. Der PC benötigt einen Algorithmus, in dem klare Anweisungen für den Sortiervorgang enthalten sind. Die einzelnen Objekte müssen irgendwo abgespeichert sein. Als Datenstruktur eignet sich z. B. ein Array. Dies hat zur Konsequenz, dass der Computer immer nur ein Objekt einsehen kann. Dies kann er dann mit dem Inhalt des Zwischenspeichers vergleichen. Pro Feld kann maximal ein Objekt gespeichert sein.

Praxistest Teil 2:

Nun soll das Sortieren eines Computers simuliert werden. Lege dazu 10 Länderkarten verdeckt wie in einer Arraystruktur auf den Tisch (alle Karten in einer Reihe). Um zu markieren, welcher Wert gerade in der Zwischenablage gespeichert ist, lege einen Stift parat. Entwickle nun unter Berücksichtigung der kommenden Regel ein Sortierverfahren, das von einem Computer ausgeführt werden könnte.

Regeln:

- Jede Karte liegt in einem Feld des Arrays.
- In jedem Feld liegt maximal eine Karte, d. h. die Karten dürfen nicht gestapelt werden.
- Wenn eine Karte gelesen wird, so wird sie umgedreht. Nun kann die Karte wieder verdeckt werden oder in die Zwischenablage kopiert werden. Dieser Vorgang wird dadurch kenntlich gemacht, dass der Stift auf die aufgedeckte Karte zeigt.
- Maximal kann nun noch eine weitere Karte aufgedeckt werden. Die zweite aufgedeckte Karte kann nun mit dem Inhalt der Zwischenablage verglichen werden.
- Wird der Karteninhalt nicht mehr gelesen, so wird die Karte wieder verdeckt.

Wie kann man nun zwei Karten tauschen?

Hierzu benötigt man ein weiteres Feld im Array. In dies Feld wird der Inhalt der aufgedeckten Karte, die nicht in der Zwischenablage ist, kopiert, d. h. man legt die aufgedeckte Karte in das zusätzliche Feld. Nun kann der Inhalt der Zwischenablage in das freigewordene Feld gelegt werden, d. h. die Karte, auf die der Stift zeigt, wird in das freigewordene Feld gelegt. Nun kann die andere Karte in das freie Feld gelegt werden. Was passiert mit der Zwischenablage?

Ergänze die untenstehende Tabelle, um die Unterschiede beim Sortieren zwischen Mensch und Maschine festzuhalten:

Computer	Mensch
kein Vorwissen	
	Intuition
	Gesamtüberblick über die zu sortierenden Daten
feste Datenstruktur	

Wer Ordnung hält, ist nur zu faul zum Suchen

Hinter diesem Satz verbirgt sich das informatische Problem, ob man durch vorheriges Sortieren tatsächlich Zeit beim Suchen spart, wenn man die Zeit des Sortierens berücksichtigt. Der Aufwand, Ordnung zu schaffen, muss dadurch gerechtfertigt werden, dass man daraus einen erhöhten Komfort ziehen kann. Dies kann mehr Platz oder schnelleres Auffinden von Dingen sein (Gallenbacher, 2008). Hat man bereits eine Ordnung geschaffen, so kann man dem Chaos entgegen wirken, indem man die neu hinzukommenden Objekte sofort an den richtigen Ort ablegt. Allerdings gibt es Situationen, in denen unsortierte Mengen vorliegen, so dass ein vollständiges Sortieren erforderlich ist.

Computer müssen häufig Zahlen, z. B. Geburtsdaten oder Versicherungsnummern der Größe nach sortieren. Ein zweites großes Feld des Sortierens umfasst das alphanummerische Sortieren von Namen, wobei dies für den Computer keinen Unterschied zu dem nummerischen Sortieren darstellt.

Damit der Computer überhaupt Daten in angemessener Zeit wiederfinden kann, ist das vorherige Sortieren notwendig.

Was bedeutet sortieren?

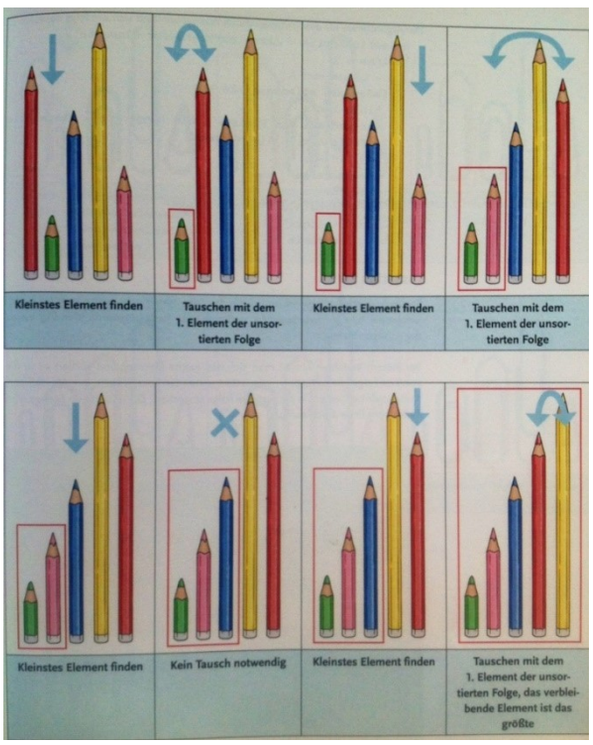
Beim Sortieren werden die Daten in eine logische Reihenfolge gebracht. Dazu benötigt man mindestens ein Kriterium, mit dessen Hilfe die Daten dann verglichen werden. Es ist möglich, dass es mehrere Kriterien für den Sortiervorgang gibt, z. B. zunächst nach Nachname, dann nach Vorname. Mehrere Kriterien schließen jedoch ein, dass sie eine Priorität haben, so dass genau festgelegt ist, welches Kriterium zunächst beachtet werden muss. Die meisten Sortierverfahren sind vergleichsbasiert, d. h. es werden immer paarweise zwei zu sortierende Objekte verglichen.

Beispiele für Sortierverfahren[13]

Bei allen Beispielen geht man von einer zufälligen unsortierten Menge aus, die in eine logische Ordnung gebracht werden soll, bei der der kleinste Wert an erster Position stehen soll.

Selection Sort/ Sortieren nach Auswahl

Beim Sortieren nach Auswahl wird zunächst der kleinste Wert gesucht. Das Objekt mit dem kleinsten Wert wird mit dem Objekt an der ersten Position vertauscht. Im nächsten Schritt wird das Objekt mit dem nächstkleineren Wert in der unsortierten Menge gesucht und mit dem Objekt am Anfang der unsortierten Menge getauscht. So baut sich eine sortierte Menge mit dem kleinsten Objekt zu Beginn auf.



Pseudocode:

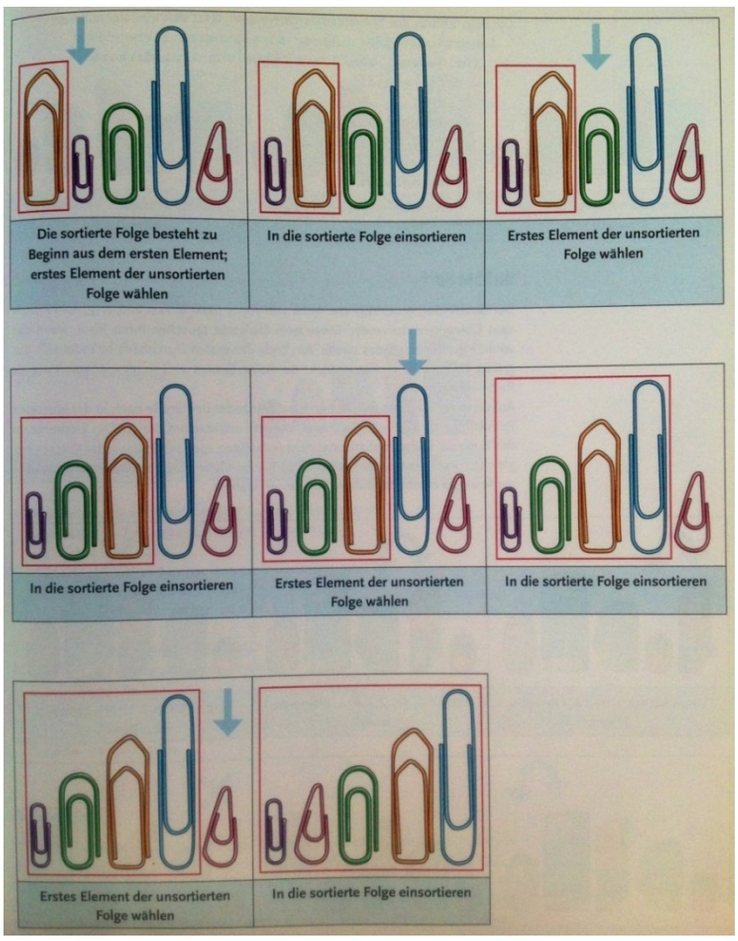
```
SelectionSort (A : Array sortierbarer Elemente)
  Links = 1
  n = Länge von A
  Wiederhole solange links < n
    min = links
    für jedes i von links+1 bis n wiederhole
      falls A[i] < A[min] dann
        min=i
      ende falls
    ende für
    vertausche A[min] und A[links]
    links = links+1
  ende wiederhole
```

Insertion Sort/ Sortieren durch Einfügen

Beim Sortieren durch Einfügen wird zunächst das zweite Objekt betrachtet und vor dem ersten Objekt einsortiert, wenn es kleiner ist, oder hinter dem ersten Objekt einsortiert, wenn es größer ist. Man erhält

[13] Die Bilder stammen aus: (Kempe & Löhr, Informatik 2, Modellierung, Datenstrukturen und Algorithmen, 2012)

zwei Teilmengen. Die erste Teilmenge besteht aus zwei sortierten Objekten, die zweite Teilmenge ist unsortiert. Nun wird das erste Objekt der unsortierten Teilmenge betrachtet und an der richtigen Position in der sortierten Teilmenge eingefügt.

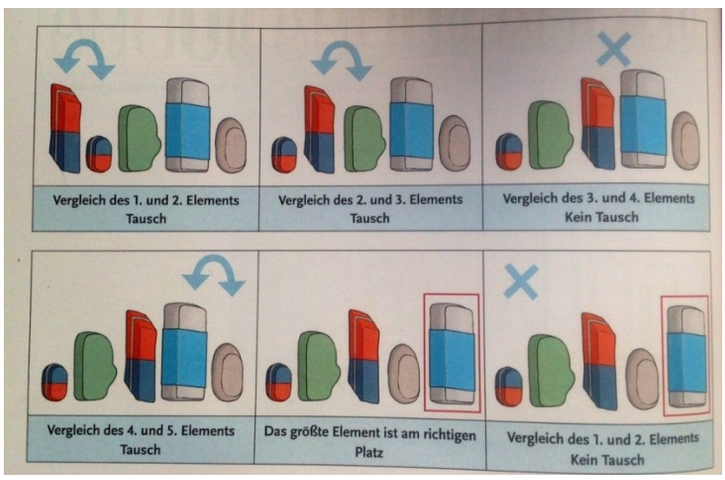


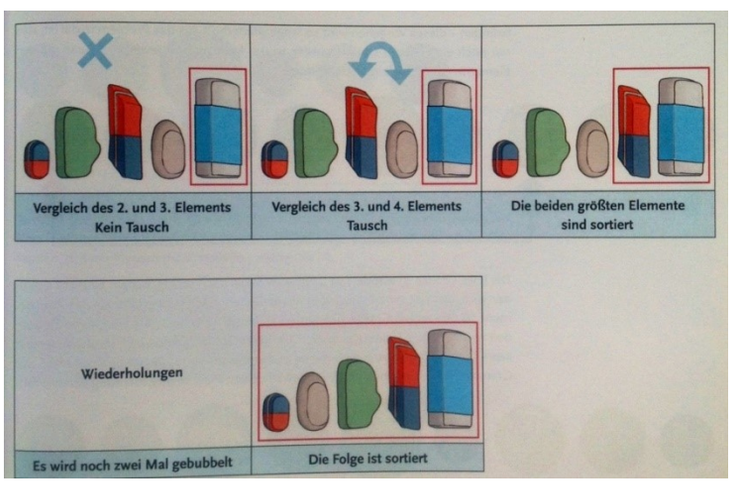
Pseudocode:

```
InsertionSort (A : Array sortierbarer Elemente)
  n = Länge von A
  für jedes i von 2 bis n wiederhole
    merke = A[i]
    j = i
    solange j > 1 und A[j-1]> merke
      A[j]=A[j-1]
      j = j-1
    ende solange
    A[j] = merke
  ende für
```

Bubble Sort/ Sortieren mit Blubberblasen

Die unsortierte Menge wird mehrmals von vorne nach hinten durchlaufen, wobei jeweils die beiden benachbarten Objekte miteinander verglichen und, falls erforderlich, getauscht werden. D. h. die Objekte tauschen ihre Plätze, wenn das vordere größer als das hintere ist. Am Ende des ersten Durchlaufs befindet sich das größte Objekt an letzter Stelle. Nach dem zweiten Durchlauf befindet sich das zweitgrößte Objekt an vorletzter Stelle. Das Sortierverfahren hat seinen Namen erhalten, weil man das Aufsteigen der großen Objekte vergleichen kann mit Luftblasen, die im Wasser aufsteigen. Auch dort steigen die größeren Blasen zuerst nach oben.





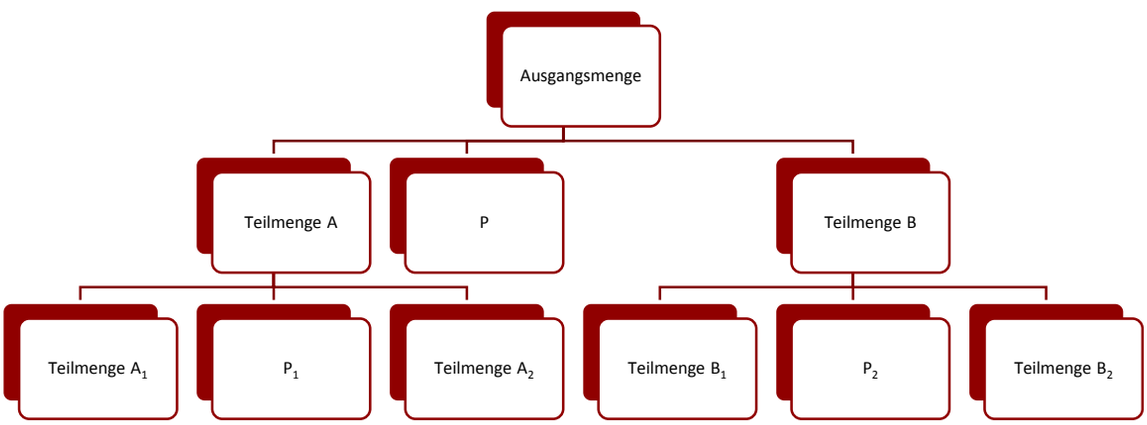
Pseudocode:

```
BubbleSort (A : Array sortierbarer Elemente)
  n = Länge von A
  wiederhole solange vertauschungen und n > 1
    vertauschungen  = falsch
    für jedes i von 0 bis n-2 wiederhole
      falls A[i] > A[i+1] dann
        vertausche A[i] mit A[i+1]
        vertauschungen = wahr
```

```
    ende falls
  ende für
  n = n-1
ende wiederhole
```

QuickSort

Zunächst wird ein Vergleichselement (Pivotelement) P frei wählbar bestimmt. Dann wird die unsortierte Menge mit Hilfe des Vergleichselementes in zwei Teilmengen aufgeteilt. Die Teilmenge A beinhaltet alle Elemente, die kleiner als P sind. Die Teilmenge B beinhaltet alle Elemente, die größer als P sind. Man erhält zwei unsortierte Teilmengen, aber alle Elemente befinden sich schon mal im richtigen Bereich bzgl. P. Nach dem gleichen Prinzip werden nun die beiden Teilmengen A und B unterteilt. Das Verfahren wird sooft wiederholt, bis jede Teilmenge aus einem einzigen Element besteht.



Pseudocode:

```
QuickSort (A : Array sortierbarer Elemente)
  Wähle das mittlere Element der Folge als Pivotelement (x)
  n = Länge von A
  i = 0
  j = n-1
  wiederhole solange i < j
    suche von links das erste Element A[i] mit A[i]≥ x
    suche von rechts das erste Element A[j] mit A[j] ≤ x
    falls i < j dann
       vertausche A[i] mit A[j]
       i = i+1
       j = j-1
    ende falls
    falls linker Teil noch mehr als ein Element hat
      quicksort(linker Teil)  //rekursiver Aufruf
    ende falls
    falls rechter Teil noch mehr als ein Element hat
      qicksort(rechter Teil) //rekursiver Aufruf
    ende falls
  ende wiederhole
```

Dividera et impera

Das Prinzip ein Problem (hier: Sortieren einer großen Menge) in mehrere kleine Probleme (hier: Sortieren von Teilmengen) aufzuteilen, nennt man „Dividera et impera“ (teile und herrsche). Die „Teile-und-herrsche“-Methode zerlegt ein Gesamtproblem, das man mit einem Algorithmus lösen will, in immer kleinere Teilprobleme, bis schließlich nur noch elementare Grundstrukturen wie Sequenzen und andere Kontrollstrukturen zur Lösung des Problems übrig bleiben. Dieses Grundprinzip stammt nicht aus der Informatik, sondern wurde von König Ludwig XI. von Frankreich im 15. Jahrhundert als Maxime ausgegeben. Dabei handelte es sich um eine Militärstrategie: den Gegner entzweien, um ihn dann leichter zu beherrschen.

Wichtig ist, dass die Teilprobleme unabhängig voneinander gelöst werden können, denn sonst müssten die Teilprobleme miteinander kommunizieren oder evtl. untereinander auf Lösungen warten. Wie gut das Problem mithilfe dieses Sortierverfahrens gelöst werden kann, hängt vorrangig von der gewählten Aufteilung, d. h. der Pivotelemente ab.

Die Problemgröße

Ein wichtiger Vorgang bei der Lösung von Aufgaben aus der Informationstechnik ist das Bestimmen der sogenannten Problemgröße. Sie stellt ein Maß für die Schwierigkeit dar, dazu gehört auch der Umfang des jeweiligen Problems. Häufig ist offensichtlich, welche Problemgröße vorliegt, aber manchmal lohnt sich auch in zweiter Blick. Die Anzahl der zu sortierenden Objekte ist sicherlich ein wichtiger Hinweis auf den Umfang des Sortierproblems, aber auch die Größe der zu sortierenden Zahlen kann die Problemgröße maßgeblich beeinträchtigen.

Nicht immer hat man die Möglichkeit die Problemgröße zu verändern. Möchte man Kundennummern sortieren, so kann man sicherlich 5000 Kunden schneller als 500000 Kunden sortieren (Anzahl der Objekte). Allerdings wächst das Problem nicht nur durch die Kundenanzahl, denn 500000 Kunden erfordern mindestens sechsstellige Kundennummern, während 5000 Kunden auch mit vierstelligen Kundennummern betreut werden könnten.

Sortierverfahren vergleichen

Wie gut ein Sortierverfahren im Vergleich zu einem anderen ist, hängt von mehreren Faktoren ab. Zunächst seien die Faktoren genannt, die für jeden Algorithmus[14] gelten müssen:

- **Korrektheit:** Das Sortierverfahren muss jede unsortierte Menge, die in der vorgegebenen Datenstruktur vorliegt, sortieren können.
- **Endlichkeit**: Das Verfahren muss mit einer endlichen Anzahl von Anweisungen beschrieben werden können.
- **Determiniertheit**: Der Sortieralgorithmus muss bei gleicher unsortierten Menge zur gleichen sortierten Menge führen.
- **Eindeutigkeit**: Während des Durchlaufs muss zu jeden Zeitpunkt eindeutig sein, welche Anweisung als nächstes zu befolgen ist.

[14] Algorithmen sind Verfahren, mit denen man Probleme (effizient) lösen kann. Algorithmen sind nicht auf die Informatik oder Mathematik beschränkt, sondern können auch Bezug auf Alltägliches nehmen.

- **Terminiertheit**: Der Sortieralgorithmus muss nach einer endlichen Anzahl von Schritten zu einer sortierten Menge führen.

Um die Sortierverfahren nun miteinander zu vergleichen, gibt es zwei wesentliche Qualitätseigenschaften von Algorithmen:

- **Effizienz**: Nach möglichst kurzer Zeit kommt das Verfahren ohne Ressourcenverschwendung zur richtigen Lösung.

- **Verständlichkeit**: Die Darstellung ist verständlich und nachvollziehbar, d. h. der Quellcode ist kommentiert.

Die Verständlichkeit hängt vorrangig vom Programmierer ab, so dass wir hier auf die Effizienz eingehen wollen. Wie viel Zeit benötigt ein Algorithmus? Es gibt zwei Möglichkeiten die Laufzeit zu überprüfen.

1. **Benchmark-Tests**
   Unter gleichen computertechnischen Bedingungen werden die unterschiedlichen Algorithmen mit den gleichen Testdaten ausgeführt. Der benötigte Zeitaufwand bis zur Lösung des Problems wird gestoppt und dann miteinander verglichen.

2. **Theoretische Laufzeitbetrachtung**
   Bei der theoretischen Laufzeitbetrachtung muss man sich zunächst überlegen, welche Schritte des Algorithmus zeitkritisch sind, d. h. Zeit kosten. Der Zugriff auf den Speicher ist beispielsweise zeitintensiv. Diese Operationen werden anschließend gezählt.
   Welche Operationen wichtig für die Laufzeitberechnung sind, hängt auch vom jeweiligen Anwendungsfall ab und kann nicht pauschal festgelegt werden. Für Sortierverfahren sind die Speichervorgänge (lesen und schreiben im Speicher), sowie der Tausch von zwei Objekten zeitkritisch. Der Tausch von zwei Objekten kann dabei in drei Verschiebeoperationen unterteilt werden:
   - Verschiebe Objekt A vom Platz n in den Zwischenspeicher
   - Verschiebe Objekt B vom Platz m auf den Platz n
   - Verschiebe Objekt A aus dem Zwischenspeicher auf den Platz m

Das Testverfahren ist in beiden Fällen nur gut, wenn die Testdaten entsprechend gut ausgewählt wurden und jedes Verfahren mit der gleichen unsortierten Menge gestartet wird. Grundsätzlich lohnt es sich auch, mal zu gucken, wie viele Schritte nötig sind, wenn eine sortierte Menge als Ausgangsmenge verwendet wird.

Bei einer echt unsortierten Menge ergaben sich folgende Ergebnisse bei einer Testreihe (Benchmark-Test): (Gallenbacher, 2008)

Karten (= Problemgröße)	SelectionSort	Bubblesort	SelectionSort/Karte	Bubblesort/Karte
2	4	4	2,0	2,0
3	9	12	3,0	4,0
4	15	6	3,8	1,5
5	22	22	4,4	4,4
10	72	96	7,2	9,6
15	147	273	9,8	18,2
20	247	535	12,4	26,8
30	522	1074	17,4	35,8

Man kann nicht nur die steigende Anzahl von Schritten bei jedem Sortierverfahren erkennen, sondern auch, dass der Aufwand pro Karte bei steigender Problemgröße zunimmt. Man hätte vermuten können, dass der Aufwand pro Karte linear zunimmt, dem ist aber nicht so. Weiterhin kann man erkennen, dass der Aufwand beim Bubblesort ungefähr doppelt so hoch ist wie beim SelectionSort. Im Bereich der theoretischen Informatik beschäftigt man sich genauer mit der Aufwandsabschätzung.

Grundsätzlich kann man die Sortierverfahren nur vergleichen, wenn man von der gleichen Voraussetzung bezüglich der Rechnerleistung ausgeht.

Aufgaben:

1. Sortiere jeweils 5 bzw. 10 Länderkarten. Damit du dein Ergebnis vergleichen kannst, musst du dir die unsortierte Menge notieren. Starte jeden Versuchsdurchlauf mit derselben unsortierten Menge. Notiere, wie viele Schritte nötig sind, bis die Länderkarten alle in der richtigen Reihenfolge liegen. Wähle selbst das Sortierkriterium aus.
    a. Verwende SelectionSort.
    b. Verwende InsertionSort.
    c. Verwende BubbleSort.
    d. Verwende Quicksort.
2. Erstelle ein Java-Programm, welches 10 Zahlen sortieren kann.
    a. Zunächst musst du die Datenstruktur anlegen, in der deine unsortierte Menge gespeichert werden kann.

    b. Erstelle eine Benutzeroberfläche. Dort soll das Feld mit den 10 Zahlen eingegeben werden können. Zusätzlich soll man per Button später das gewünschte Sortierverfahren auswählen können.

c. Beim Sortieren müssen ständig zwei Objekte, in diesem Fall Zahlen, getauscht werden. Schreibe daher eine Methode: tauschen().

d. Wähle zunächst ein Sortierverfahren aus und implementiere es.

e. Prüfe, ob dein Programm folgende Zahlenreihen richtig sortieren kann:

   i. 3, 5, 1, 9, 2, 8, 4, 11,34, 0

   ii. 9, 12, 34, 2, 1, 3, 2, 9, 13, 6

   iii. 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

   iv. 1, 3, 5, 7, 8, 9, 11, 13, 15, 17

f. Um ein Sortierverfahren richtig testen zu können, müsste man größere Felder sortieren können. Erstelle daher ein Textfeld, indem man die Anzahl der Elemente eingeben kann. Die Zahlen sollen nun per Zufall erzeugt werden. Hierfür benötigt man eine Methode void feldErzeugen(). Die erzeugten Zahlen müssen zum einen in einem Array abgespeichert werden und zum andern auf der Benutzeroberfläche in einer TextArea angezeigt werden. Dies geschieht über die zu schreibende Methode feldAusgeben(). Teste nun dein Sortierverfahren. Sind noch Änderungen notwendig? Das Ergebnis soll in der TextArea angezeigt werden.

g. Programmiere weitere Sortierverfahren.

h. Zur Laufzeitmessung kannst du die Systemzeit deines Rechners nutzen. Lass das Programm direkt vor dem Methodenaufruf des Sortierverfahrens und direkt nach dem Sortieren die Zeit auslesen. Wenn du die Differenz bildest, erhältst du die Laufzeit.

```
long zeit;
…
//Systemzeit vor dem Sortieren
zeit = System.currentTimeMillis();
…
//Ausführen des Sortieralgorithmus
…
//Differenzeit in ms
Zeit = System.currentTimeMillis()-zeit;
```

i. Bestimme die Laufzeit für verschiedene Feldgrößen und Sortierverfahren. Visualisiere deine Ergebnisse in einem Diagramm. (per Hand oder Excel)

Glossar[15]

Abhängigkeit (engl.: dependency) ist eine Beziehung zwischen Modellelementen derart, dass eine Modifikation des einen Elements eine solche des abhängigen Elements nach sich zieht. Abhängigkeiten werden verwendet, wenn ein Zusammenhang vorliegt, der nicht durch eine Assoziation ausgedrückt werden soll bzw. kann. Beispielsweise sind die Anwendungsfälle Bootverleihen und Kundendatenändern vom Anwendungsfall Kundeidentifizieren abhängig. In UML wird eine Abhängigkeitsbeziehung durch einen gestrichelten Pfeil mit offener Spitze dargestellt, die auf das Element weist, von dem das andere abhängig ist.

Abstraktion (von lat.: abstrahere = abziehen, absehen von etwas) ist das logische Verfahren der Begriffsbildung. Dabei wird von unwesentlichen Einzelheiten abgesehen und das für den Begriff Wesentliche herausgehoben. Die Möglichkeit abstrakter Begriffe beruht darauf, dass wir über wohlunterschiedene Gegenstände in Absehung ihrer jeweiligen Unterschiede unter einem bestimmten Aspekt reden können.
Beispiel: Unter Absehung ihrer Verschiedenheit kann man Hund, Katze und Pferd als Säugetiere (oder auch als Haustiere) bezeichnen. Die abstrakten Begriffe Säugetier oder Haustier bezeichnen somit jeweils eine Klasse von Tieren. Ein Pferd ist der (konkrete) Fall eines Haustiers; Roland ist der konkrete Fall (Exemplar) eines Pferdes.
Abstrakte Klasse ist eine Klasse, zu der keine Exemplare (Objekte) gebildet werden können bzw. dürfen. Sie ist absichtlich unvollständig definiert und bildet damit eine Basis für Unterklassen, zu denen es Exemplare geben kann. Diese sind verpflichtet, die abstrakten Operationen zu implementieren. Beispiel: Begriff Vertrag. Es gibt keinen Vertrag an sich, wohl aber Kaufverträge, Werkverträge, Versicherungsverträge usw. Man kann die abstrakte Klasse Vertrag bilden, von der die Klassen Kaufvertrag usw. abgeleitet werden. In UML wird eine abstrakte Klasse wie eine normale Klasse dargestellt, der Klassenname ist jedoch kursiv gesetzt.

Aggregation (von lat.: aggregare = sich zugesellen, hinzunehmen, zusammenhäufen) ist ein Sonderfall der Assoziation; sie drückt eine Teile-Ganzes-Beziehung aus im Sinne von: Das Ganze besteht aus Teilen. Die Existenz der Komponenten ist von der des Aggregats nicht abhängig (im Gegensatz zur Komposition). Jede Klasse kann als Aggregat ihrer Attribute angesehen werden. In UML wird die Aggregation durch eine gerade Linie mit einer leeren Raute auf der Seite des Ganzen (des Aggregats) dargestellt.

Anwendungsfall (engl.: use-case) ist eine Gesamtheit von Aktionen, die in einer bestimmten Reihenfolge ablaufen und ein bestimmtes Ergebnis zum Ziel haben. Er modelliert einen Arbeitsablauf aus der Sicht seiner Akteure. Ein Anwendungsfall wird stets durch einen Akteur initiiert und führt zu einem für alle Akteure wahrnehmen Ergebnis. Beispiele: (1) Reise buchen, (2) Datei speichern, (3) Geld abheben 3, vorige Seite). Mithilfe (der Beschreibung) von Anwendungsfällen können die funktionalen Anforderungen an ein Softwaresystem formuliert werden. Die Exemplare eines Anwendungsfalls heißen Szenarien. In UML wird ein Anwendungsfall durch eine Ellipse oder ein Rechteck mit Ellipse grafisch dargestellt.

Assoziation (von lat.: associare = beigesellen; vereinigen, verbinden) ist eine Beziehung zwischen Objekten bzw. Klassen. Eine Assoziation zwischen Klassen beschreibt die Beziehung abstrakt; zwischen den Objekten besteht die Verknüpfung (engl.: link) konkret. In UML wird eine Assoziation durch eine gerade Linie zwischen den Klassensymbolen dargestellt; der Name wird über der Linie notiert. Mehrstellige

[15] Vgl. (LOG IN, Nr. 128/129)

Assoziationen werden durch eine Raute dargestellt, von der aus je eine Linie zu den miteinander verbundenen Klassen gezogen wird
Beispiel: Kunde entleiht Buch beschreibt eine Assoziation zwischen den Klassen Kunde und Buch. Die Aussage Heinz Hinterpförtner entleiht Joseph der Ernährer beschreibt ein konkretes Exemplar dieser Assoziation.
Assoziative Klasse ist eine Klasse, deren Exemplare die Assoziation realisieren. Beispiel: Kunde bestellt Ware. Jede einzelne Bestellung kann ihrerseits als Objekt aufgefasst und mit Attributen (z.B. Datum) und Methoden versehen werden. Die Gesamtheit dieser assoziativen Objekte bildet die assoziative Klasse Bestellung.

Attribut (von lat.: attributum = das Zugeteilte) ist ein Merkmal, Kennzeichen oder eine wesentliche Eigenschaft. Attributwert ist ein Element aus dem Wertebereich eines Attributs. Beispiel: Das Attribut Ampelfarbe hat die Werte {rot, grün, gelb}; das Attribut Alter (eines Menschen) hat die Attributwerte {0, 1, 2, …, 120}.

Basisklasse: Oberklasse

Destruktor (von lat.: destructio = das Niederreißen) ist eine Methode zum Entfernen eines Objekts aus dem Arbeitsspeicher.

Dynamische Bindung (engl.: late binding) ist die Bindung an (unterschiedliche) Exemplare innerhalb einer Klassenhierarchie zur Programmlaufzeit. Beispiel: Objekt schüler der Klasse Schüler wird deklariert; später (zur Programmlaufzeit) wird jedoch in Abhängigkeit von der Programmsituation entschieden, ob Exemplare der Klasse Schüler oder der Unterklasse Oberstufenschüler erzeugt werden.

Eigenschaft (engl.: property): Attribut

Exemplar (engl.: instance) einer Klasse ist ein Objekt dieser Klasse (entsprechend dem Element einer Menge). Jedes Exemplar besitzt einen individuellen Satz von Attributwerten. Allgemeiner spricht man von Exemplaren (oder Ausprägungen) eines Klassifizierers. Die Menge aller Exemplare einer Klasse bildet die Extension dieser Klasse.

Generalisierung (von lat.: generalis = zur Gattung gehörend, allgemein) ist Verallgemeinerung, d.h. ein Verfahren, um aus einer Aussage eine allgemeinere Aussage zu gewinnen, also eine solche, die auf eine umfangreichere Menge von Gegenständen zutrifft. Beispiel: Aus „Alle Menschen sind sterblich“ wird durch Generalisierung: „Alle Lebewesen sind sterblich“. Konvers dazu: Spezialisierung.

Geschäftsprozess (engl.: business use-case, workflow) besteht aus mehreren zusammenhängenden Tätigkeiten, die ausgeführt werden, um ein geschäftliches Ziel zu erreichen oder ein gewünschtes Betriebsergebnis zu erzielen.

Instanz (engl.: instance = Beispiel, Einzelfall): Exemplar

Interaktionsdiagramm (engl.: interaction diagram) stellt dar, wie die Komponenten eines Systems miteinander interagieren. Oberbegriff zu: Sequenzdiagramm, Kollaborationsdiagramm.

Kapselung (engl.: encapsulation) bezeichnet das Verfahren, Information zu verstecken (engl.: information hiding). Daten und Operationen sind in Objekten eingeschlossen; auf sie kann nur mittels spezieller Methoden zugegriffen werden (Schnittstelle). Der Benutzer dieser Methoden braucht über die Art und

Weise, wie die Interna implementiert sind, nichts zu wissen. Der Programmierer dagegen kann das Innere der Objekte ändern, ohne Nebenwirkungen fürchten zu müssen.

Klasse (engl.: class, von lat.: classis = versammelte Menge, Abteilung, Flotte) bezeichnet, meist gleichbedeutend mit Menge, eine Zusammenfassung mehrerer Gegenstände zu einem Ganzen. Die Gegenstände heißen dann Exemplare der Klasse (bzw. Elemente der Menge). In der OOM ist eine Klasse die Gesamtheit der Objekte mit gleichen Attributen und Methoden. Eine Klassendefinition ist eine Art Schablone (Bauplan, Vorlage) für die Exemplare dieser Klasse. Da durch Aufruf eines Konstruktors der Klasse ein Objekt erzeugt werden kann, werden Klassen zuweilen auch als Objektfabriken bezeichnet. In UML wird eine Klasse durch ein dreigeteiltes Rechteck dargestellt; oben steht der Name der Klasse, dann werden (jeweils durch einen waagerechten Strich getrennt) die Attribute und anschließend die (Signaturen der) Methoden aufgeführt. Das Klassenattribut ist ein Attribut, das für alle Objekte der Klasse den gleichen Wert hat. Es drückt keine Eigenschaft eines Objekts, sondern eine der Klasse aus (z.B. Anzahl der Exemplare der Klasse).

Klassendiagramm (engl.: class diagram) beschreibt die statische Struktur eines Systems, d.h. aus welchen Klassen es besteht und in welcher Beziehung diese zueinander stehen.

Komponente (von lat.: componere = zusammenfügen, verfassen) ist allgemein irgendein (Bestand-) Teil eines Ganzen. In UML ist eine Komponente ein Systemteil, der seinen Inhalt kapselt, eine definierte Funktionalität anbietet und über Schnittstellen mit anderen Komponenten kommunizieren kann. Komponenten werden durch ein mit dem Stereotyp «Komponente» versehenes Klassensymbol dargestellt.

Komposition (von lat.: compositio = Zusammenstellung, Zusammensetzung) ist ein Sonderfall der Aggregation; sie modelliert eine physische Teile-Ganzes-Beziehung. Die Komponenten sind vom Kompositum existenzabhängig, d. h. die Erzeugung (bzw. Löschung) des Kompositums erzeugt (bzw. löscht) auch die Komponenten. In UML wird die Komposition durch eine gerade Linie mit einer gefüllten Raute auf der Seite des Ganzen (des Kompositums) dargestellt.

Konstruktor ist eine spezielle Methode zur Erzeugung und Initialisierung von Objekten. Eine Klasse kann mehrere Konstruktoren besitzen, die sich in der Signatur unterscheiden. Konvers dazu: Destruktor.

Lebenslinie (engl.: life line) ist eine der senkrechten Geraden im Sequenzdiagramm; sie symbolisiert die Zeitachse.

Mehrfachvererbung ist eine Vererbungsbeziehung derart, dass die abgeleitete Klasse mehrere Oberklassen besitzt. Beispiel: Die Klasse Ratenkaufvertrag kann von Kaufvertrag und Kreditvertrag abgeleitet werden. Einige OO-Sprachen (C++, EIFFEL) erlauben Mehrfachvererbung, JAVA nur indirekt mittels Schnittstellen, SMALLTALK jedoch nicht.

Methode (engl.: method, von griech.: methodós = Weg, etwas zu erreichen) ist ein mehr oder weniger planmäßiges Verfahren zum Erreichen eines Ziels, das nach einiger Zeit beherrscht werden kann. In der OOM ist Methode die Handlungsvorschrift zur Erbringung eines Dienstes, speziell eine objektlokale Prozedur oder Funktion.

Nachricht (oder: Botschaft, engl.: message) ermöglicht den Informationsaustausch zwischen Teilnehmern an einer Interaktion. Nachrichten werden in UML als Pfeile zwischen den Lebenslinien der

Interaktionsteilnehmer (vom Sender zum Empfänger) notiert; sie können entweder im Aufruf einer Operation oder in der Übermittlung eines Signals bestehen. Man kann den Sender als Kunden (engl.: client), den Empfänger als Anbieter eines Dienstes (engl.: server) interpretieren: Der Anbieter empfängt die Nachricht und erbringt einen Dienst, indem er die entsprechende Operation ausführt.

Oberklasse (engl.: superclass) ist eine zweistellige Relation zwischen Klassen. Es gilt: A ist Oberklasse von B, wenn jedes Exemplar von B auch Exemplar von A ist. A ist Ergebnis einer Generalisierung. Konvers dazu: Unterklasse.

Objekt (von lat.: obiectum = das Entgegengeworfene) ist seit dem achtzehnten Jahrhundert im deutschen philosophischen Sprachgebrauch durch Gegenstand ersetzt. In der OOM repräsentiert ein Objekt einen beliebigen Gegenstand (Person, Ding, Thema, Sachverhalt, Menge anderer Gegenstände) und besitzt messbare, durch Werte erfassbare Eigenschaften. Ferner kann ein Objekt Tätigkeiten, d. h. Methoden ausführen und damit Ereignisse auslösen und Nachrichten übermitteln. Objekte treten als Exemplare einer Klasse auf.

Polymorphie (wörtlich: Vielgestaltigkeit, von griech.: polys = viel und morphé = Gestalt) bedeutet die Möglichkeit, dass Objekte einer Klassenhierarchie Operationen gleichen Namens unterschiedlich implementieren. Eine Operation ist polymorph, wenn sie abhängig vom Typ ihrer Argumente unterschiedliches Verhalten zeigt. Erst zur Laufzeit wird aufgrund der Klassenzugehörigkeit des Arguments entschieden, welche Implementation zur Ausführung gelangt (dynamische Bindung).

Relation (engl.: relationship, von lat.: relatio = Erwiderung, Vortrag, Beziehung, Verhältnis) ist eine Beziehung zwischen Modellelementen. Eine zweistellige Relation ist (extensional) eine Paarmenge. Beispielsweise besteht die Beziehung Kunde entleiht Buch aus alle Paaren (K, B), wo K für einen Kundennamen bzw. eine Kundennummer und B für den Bezeichner eines Buchs steht.

Realisierung ist eine Beziehung zwischen einem Modellelement, das eine Anforderung beschreibt, und einem, das diese Anforderung erfüllt. Sie wird in UML durch einen gestrichelten Pfeil mit nicht ausgefüllter Spitze dargestellt (Schnittstelle). Eine Realisierungsbeziehung ist statt einer Spezialisierung angezeigt, wenn sich die beteiligten Elemente in verschiedenen Modellen befinden.

Restriktion: Einschränkung

Schnittstelle (engl.: interface) beschreibt das extern wahrnehmbare Verhalten von Modellelementen durch Angabe von Methodensignaturen. Man sagt, dass die Klasse A die Schnittstelle S implementiert, wenn sie die Methoden, deren Signatur in S gegeben ist, durch Prozeduren realisiert. In UML werden Schnittstellen wie Klassen dargestellt; sie tragen jedoch das Stereotyp «interface». Der Pfeil einer Klasse zur Schnittstelle sieht aus wie der einer Vererbungsbeziehung, der Pfeilschaft ist jedoch gestrichelt (Realisierung). Mithilfe von Schnittstellen können verschiedene Entwickler Verträge schließen (Entwurf per Vertrag, engl.: design by contract): Der eine hat die Schnittstelle in seiner Klasse zu implementieren, der andere nutzt die Methoden, die in der Schnittstelle vereinbart sind. Eine Klasse kann mehrere Schnittstellen implementieren (Mehrfachvererbung).

Sequenzdiagramm (engl.: sequence diagram) zeigt, in welcher Reihenfolge Objekte miteinander kommunizieren, um eine bestimmte Aufgabe zu lösen. Die Objekte werden auf der Horizontalen angetragen, die Vertikale (Lebenslinie) bestimmt die zeitliche Reihenfolge, in der die Teilaufgaben gelöst

werden. Eine Nachricht wird als Pfeil dargestellt, der von der Lebenslinie des Senders zu der des Empfängers verläuft.

Spezialisierung (von lat.: specialis = besonders) ist ein Verfahren, um aus einer Aussage eine speziellere Aussage zu gewinnen, d.h. eine solche, die auf eine kleinere Menge von Gegenständen zutrifft. Beispiel: Aus „Alle Schüler bekommen ein Zeugnis“ wird „Alle Oberstufenschüler bekommen ein Zeugnis“. Konvers dazu: Generalisierung. In der OOM versteht man unter Spezialisierung die Beziehung zwischen einem allgemeinen und einem speziellen Modell element; sie wird in UML durch einen Pfeil mit nicht ausgefüllter Spitze bezeichnet, die zum allgemeinen Element zeigt (Vererbung).

Spezifikation (von lat.: specificus = eigentümlich) ist die präzise Beschreibung eines Sachverhalts (z.B. die Beziehung von Objekten zueinander oder Aufbau und Wirkungsweise eines geplanten Systems).

Subklasse (engl.: subclass): Unterklasse

Szenarium (auch: Szenario; Plural: Szenarien; wörtlich: Szenenfolge) ist eine zeitlich und örtlich spezifizierte Folge von Aktionen bzw. Verarbeitungsschritten. In der Praxis werden Szenarien durchgespielt, um Erkenntnisse über kausale Abhängigkeiten zu gewinnen. Die abstrakte Form eines Szenariums heißt Anwendungsfall, d.h. jedes Szenarium ist Exemplar eines Anwendungsfalls.

Transaktion ist ein Vorgang, der entweder vollständig oder gar nicht vollzogen werden kann bzw. darf. Das heißt: Wenn der Vorgang irgendwo mittendrin abgebrochen wird, hat dies die Wirkung, als hätte er nie stattgefunden.

Typ beschreibt Merkmale und Verhalten von Konstanten, Variablen und Objekten. Die Wörter Klasse und Typ (oder: Objekttyp) betonen unterschiedliche Aspekte des gleichen Begriffs.

Überladen (engl.: overloading) macht es möglich, dass Operationen mit gleichen Bezeichnern, aber unterschiedlichen Parameterlisten ohne Konflikte nebeneinander existieren können. Eine bereits definierte Operation kann in einer abgeleiteten Klasse unter Verwendung des gleichen Namens neu definiert werden. Zur Laufzeit kann der Compiler anhand des Objekts und der beim Aufruf der Operation übergebenen Parameterwerte erkennen, welche Version ausgeführt werden muss.

Überschreiben (engl.: overriding) macht es möglich, dass Operationen mit gleichen Bezeichnern und gleichen Parameterlisten (also gleicher Signatur) ohne Konflikte nebeneinander existieren können. Eine bereits definierte Operation kann in einer abgeleiteten Klasse unter Verwendung der gleichen Signatur neu definiert werden. Beim Aufruf erkennt das Laufzeitsystem anhand der Unterklasse, zu der das Objekt gehört, welche Version auszuführen ist. Zwei syntaktisch völlig übereinstimmende Aufrufe können sich auf zwei unterschiedliche Methoden beziehen je nachdem, auf welche Art Objekt die Objektvariable verweist: Der Methodenname wird erst zur Laufzeit an die entsprechende Methode gebunden (dynamische Bindung, Polymorphie).

Unterklasse (engl.: subclass) ist eine zweistellige Relation zwischen Klassen. Es gilt: A ist Unterklasse von B, wenn jedes Exemplar von A auch Exemplar von B ist. B ist eine Oberklasse von A.

Use-Case: Anwendungsfall

Vererbung (engl.: inheritance) beschreibt die Beziehung zwischen zwei Klassen, von denen die eine Oberklasse, die andere Unterklasse (oder: abgeleitete Klasse) ist. Die abgeleitete Klasse besitzt die gleichen

Attribute und Methoden wie die Oberklasse (sie erbt, d.h. übernimmt sie), besitzt aber eventuell noch weitere (Spezialisierung).

Wiederverwendung (engl.: reuse bzw. code reuse) ist die Möglichkeit oder das Bestreben, bereits entwickelte Systemkomponenten bzw. Programmtext möglichst oft und in möglichst vielen unterschiedlichen Zusammenhängen einzusetzen. Anwendungsbereiche sind Funktions- und Klassenbibliotheken, abgeleitete Klassen bzw. Vererbung, komponentenbasierte Programmierung, Entwurfsmuster.

Datenbanken

Das konzeptuelle Modell

Warum beschäftigen wir uns im Informatikunterricht mit Datenbanken?

Datenbanken sind aus der modernen Welt nicht mehr wegzudenken. Zahlreiche Informationssysteme arbeiten auf der Grundlage der Datenbankidee. Um große Datenmengen sinnvoll und effektiv zu verwalten, benötigt man gut organisierte Datenbanken.

In der Schule werden die Schülerdaten aller hessischen Schulen mithilfe LUSD (Lehrer- und Schülerdatenbank) festgehalten und verwaltet. Eine große Bücherei führt die Bücherbestände, die Kundendatei etc. mithilfe eines Datenbanksystems.

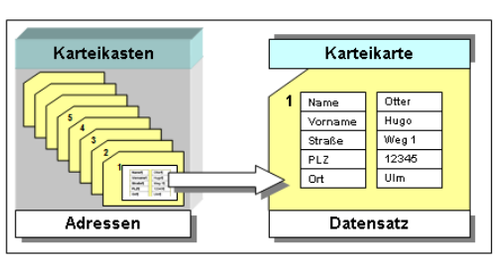
Um eine gute Datenbank zu erhalten, müssen bestimmte Informationen richtig aus der Realität in das zu stellende Programm übertragen werden. Diesen Vorgang nennt man Modellbildung. Er ist ein wesentlicher Aspekt des Informatikunterrichts. Anhand dieses Vorgangs lernt man viel über Datenstrukturen, Wertebereiche, Datenkapselung, Mensch-Maschine-Kommunikation und Datenorganisation.

Ziel ist es, dass ihr neben der Fähigkeit konkret mit einer Datenbank zu arbeiten auch die Analyse, die Beschreibung und schließlich die Modellierung realer Problemstellungen in komplexe Systeme erledigen könnt.

Vom Karteikasten zur Datenbank

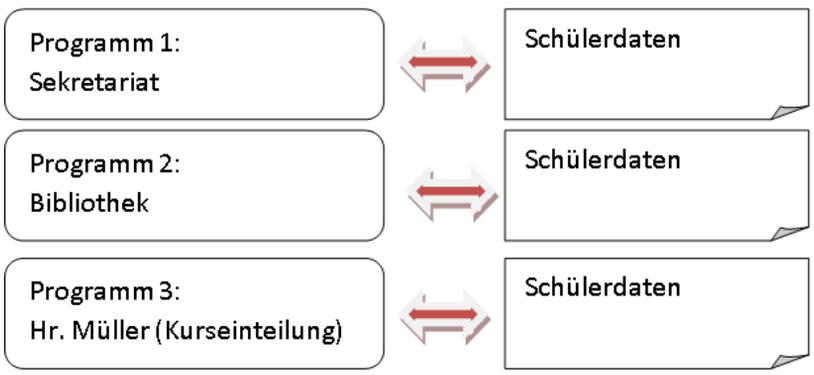
Bevor es Computer im heutigen Ausmaß gab, wurden die Kundendaten eines Unternehmens zum Beispiel in einem Karteikasten gesammelt. Dort konnte man dann bei Bedarf nachlesen, wie der Kunde mit der Kundennummer 234 heißt, wo er wohnt und welche Telefonnummer er hat. Um auf diese Daten zugreifen zu können, mussten die Karteikarten erstellt werden. Von Kunden, die seit Jahren nicht mehr vorbeigekommen waren, mussten die Karteikarten aus dem System entfernt werden. Die Karteikarten mussten in den richtigen Karteikasten, die entsprechend beschriftet werden mussten, einsortiert werden. Wollte man mit bestimmten Kunden Kontakt aufnehmen, musste per Hand eine Liste geschrieben werden.

Eine Datenbank ist ein modernes System mehrere Karteikästen, wobei die Karteikästen bei Bedarf voneinander wissen und notwenige Daten miteinander verknüpfen können. Ein Karteikasten entspricht dabei einer Datenbanktabelle. Eine einzelne Zeile in solch einer Datenbanktabelle bezeichnet man als Datensatz. Er ist mit einer einzelnen Karteikarte vergleichbar. In der Zeile stehen die Informationen über einen Kunden. Um Kunden hinzuzufügen, muss man also in der Tabelle eine Zeile mit den neuen Kundendaten hinzufügen. Nicht mehr benötigte Datensätze können gelöscht werden. Das Sortieren der Kundendaten und das Zusammenstellen bestimmter Daten übernimmt das Programm.



Vorteile der Datenbank gegenüber des Karteikastens

An einer Schule werden an unterschiedlichen Stellen Stammdaten der Schüler wie Vorname, Zuname, Geburtsdatum, Klassenzugehörigkeit usw. benötigt. Jeder führt seinen eigenen Karteikasten mit den Schülerdaten. Der erste Schritt zu Modernisierung wäre, dass jede Stelle (Sekretariat, Oberstufenleiter, Büchereileiter) die Daten mithilfe eines eigenen Programms verwalten.



Bei dieser Vorgehensweise kann es vorkommen, dass im Sekretariat ganz andere Informationen über einen Schüler vorhanden sind als in der Bibliothek oder beim Oberstufenleiter. Man stelle sich vor, ein Schüler verlässt die Schule und meldet sich im Sekretariat ab. Dort wird der Schüler aus dem Karteikasten oder dem Programm entnommen. Er bleibt aber, wenn keine Information an die beiden anderen Nutzer geht, in den beiden anderen Systemen als Schüler der Schule vorhanden.

Mögliche Probleme:

- **Datenredundanz**: Gleiche Informationen werden mehrmals gespeichert.
  „Da die Daten jeweils speziell für bestimmte Anwendungen entworfen werden, werden dieselben Daten in verschiedenen Dateien wieder auftauchen(..) . Redundanz führt zu Speicherverschwendung und zu erhöhten Verarbeitungskosten, vor allem bei Änderungen. Schlimmer jedoch ist es, dass diese Redundanz in der Regel nicht zentral kontrolliert wird, so dass Konsistenzprobleme auftreten.“ (Matzke, 2000)
- **Dateninkonsistenz**: eigentlich gleiche Datensätze sind unterschiedlich gespeichert
  „Die Konsistenz der Daten (d. h. die logische Übereinstimmung der Datei-Inhalte) kann nur schwer gewährleistet werden. Bei der Änderung einer Größe müssten alle Dateien geändert werden, die diese Größe beinhalten, und diese Änderungen müssten so miteinander abgestimmt geschehen, dass nicht verschiedene Programme zum selben Zeitpunkt unterschiedliche Werte derselben Größe sehen können.“ (Matzke, 2000)
- **Inflexibilität**:
  „Da die Daten nicht in ihrer Gesamtheit sondern nur anwendungsbezogen gesehen werden, ist es in vielen Fällen sehr kompliziert, neue Anwendungen oder Auswertungen vorhandener Daten zu realisieren. Dies gilt insbesondere für Auswertungen, die Daten aus verschiedenen Dateien benötigen. Die Organisation nach diesem konventionellen Vorgehen ist sehr wenig anpassungsfähig an die sich verändernden Anforderungen in einem Unternehmen bzw. in einer Schule.“ (Matzke, 2000)
- Fehlende **Standardisierung**: Daten werden programmabhängig gespeichert

„Ändert sich der Aufbau einer Datei oder ihrer Organisationsform, so müssen darauf basierende Programme geändert werden. Wird beispielsweise für eine Anwendung ein weiteres Datenelement in einem Satztyp benötigt (z. B. zweite Telefonnr. eines Schülers), so müssen infolge der notwendigen Neudefinitionen der Datei alle Programme geändert werden, ob sie dieses neue Datenelement sehen wollen oder nicht.“ (Matzke, 2000)

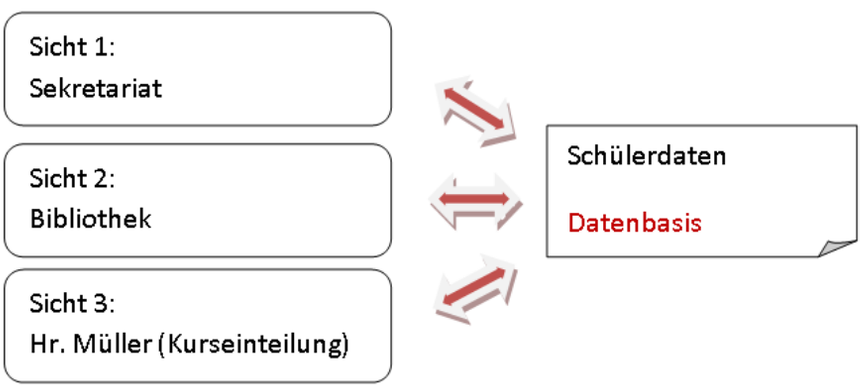
- Unflexibel gegenüber Änderungen

- Keine Datenintegrität: Datensätze können gelöscht werden, obwohl sie an anderen Stellen noch benötigt werden.

Aufgaben:

1. Nenne zu den oben genannte Problemen je ein Beispiel bzgl. der Schülerdaten.

Konzept des Datenbanksystems



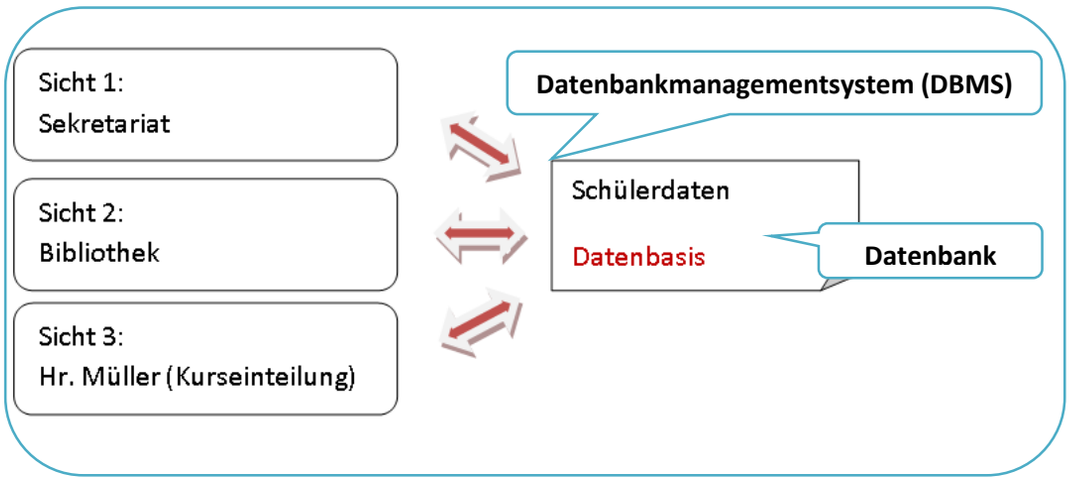
Abhilfe bei den oben aufgetretenen Problemen kann ein gut programmiertes Datenbankmanagement - System schaffen. Wichtigster Aspekt hierbei ist, dass die Daten zentral verwaltet werden. Alle Daten sind gemäß ihren logischen Zusammenhängen organisiert und nicht entsprechend den Anforderungen gewisser Bereiche.

Das Datenbankmanagement-System stellt dann die Daten den verschiedenen Nutzern in der für sie brauchbaren Übersicht dar.

Aufgaben:

2. Sind die oben aufgeführten Probleme nun gelöst?
3. Wie sieht es mit dem Datenschutz aus?

Begriffe



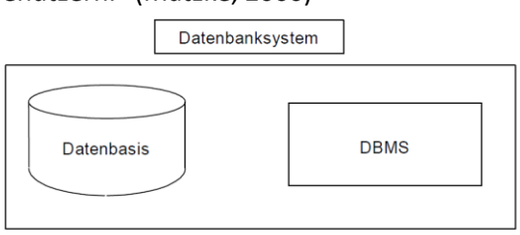
Datenbanksystem (DBS)

„Sehr allgemein gesprochen stellt man sich unter einem **Datenbanksystem** ein System vor, das es erlaubt, große Datenmengen abzuspeichern, nach beliebigen Kriterien Daten wiederzufinden und Daten zu verändern. Beispielsweise möchte man, wenn man die Daten über alle Schüler einer Schule abgespeichert hat, die Anfrage stellen können:

> Liste alle Schüler auf, die nicht in Augsburg wohnen und 20 Jahre alt sind.

Datenbanksysteme werden im Folgenden als ein Organisationsmittel betrachtet, das folgende Aufgaben in einer Organisation abdeckt:

- Die Daten der Organisation sind für alle Benutzer in einer gemeinsamen Datenbasis (die Datenbank) abgespeichert.
- Viele Benutzer mit unterschiedlichen Anforderungen arbeiten mit diesen Daten. Das Datenbanksystem übernimmt den Zugriff und die Darstellung der gewünschten Daten.
- Das Datenbanksystem kontrolliert den Zugang zu den Daten, es zeigt Daten nur berechtigten Benutzern.“ (Matzke, 2000)



Genaugenommen bezeichnet also der Begriff **Datenbank** nur die zentrale Datenbasis eines Datenbanksystems. Sowohl in der Literatur als auch im allgemeinen Sprachgebrauch verwendet man den Begriff Datenbanken jedoch auch für das gesamte Datenbanksystem.

Das **Datenbankmanagementsystem** (DBMS) sorgt für die Verknüpfung der verschiedenen Benutzer (das können Menschen oder Programme sein) mit der Datenbasis. Mit Hilfe des DBMS kann das Datenbanksystem verwaltet werden.

„Ein Datenbanksystem ermöglicht es dem Benutzer, über ein Datenbankmanagementsystem (…)

- die Struktur einer Datenbasis aufzubauen (Datendefinition),
- Daten zu pflegen: Datensätze eingeben, ändern und löschen (Datenmanipulation),
- Informationen aus der Datenbasis zu gewinnen (Datenabfrage),
- Zugangs- und Zugriffsrechte zu verwalten (Datenkontrolle),
- Daten zu sichern, zu exportieren und zu importieren (Datenübertragung).“ (Matzke, 2000)

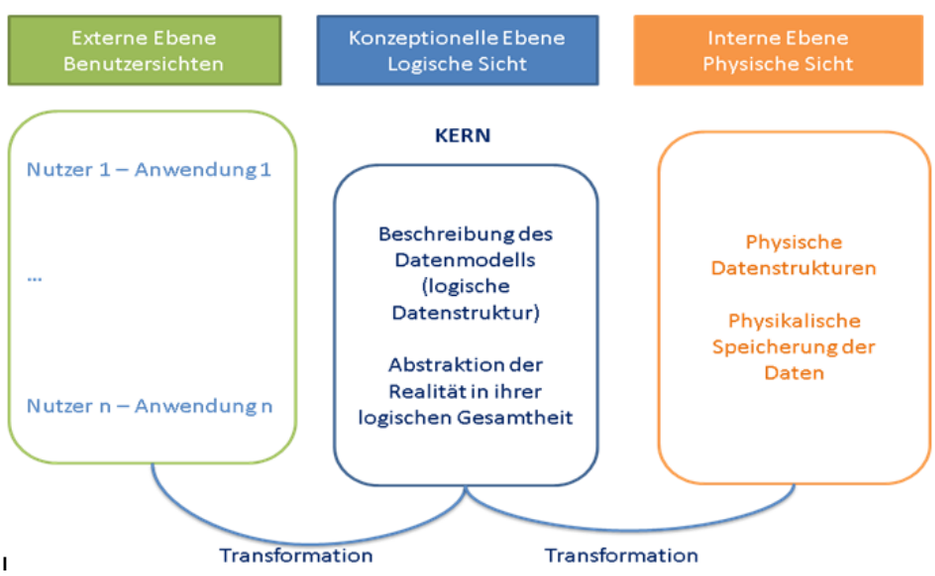
Durch das Konzept des Datenbanksystems werden zusätzlich die Datensicherheit und der Datenschutz gewährleistet. Bestimmte Benutzer können nur für sie relevante Daten sehen, während andere Benutzer weitreichendere Rechte haben. Zudem wird kontrolliert, ob bestimmte Änderungen oder Löschungen überhaupt erlaubt sind.

Aufgaben:

4. Welche Daten sollte das Sekretariat sehen können, die Büchereiverwaltung jedoch nicht?
5. Unter welchen Voraussetzungen sollte das Sekretariat einen Schüler nicht aus der Datenbasis löschen können?
6. Wann kann Hr. Müller einem Schüler keinen Kurs zuweisen?

Die Ebenen des Datenbankmanagement-Systems

Urheber der 3-Schichten-Architektur ist das American National Standards Institute, d. h. der nationale Normenausschuss der USA (vergleichbar mit dem DIN in Deutschland). 1975 unterbreitete das ANSI den Vorschlag für die prinzipielle Architektur von Datenbanksystemen. Die wichtigste Ebene ist dabei die konzeptionelle Ebene.



Externe Ebene: Jeder sieht nur, was er für seine Aufgaben benötigt! Jeder kann nur die Daten verändern, die er die Berechtigung hat. Für den Benutzer ist die Datenbank ein Informationsspeicher und -lieferant!

Konzeptionelle Ebene: Sie bildet den Kern der Architektur. In ihr wird die Gesamtheit der logischen Zusammenhänge und Abhängigkeiten der in der Realität gegebenen Daten nachgebildet. Die Modellieren der gegebenen Daten bezüglich ihrer Eigenschaften und Beziehungen ist sowohl unabhängig von der

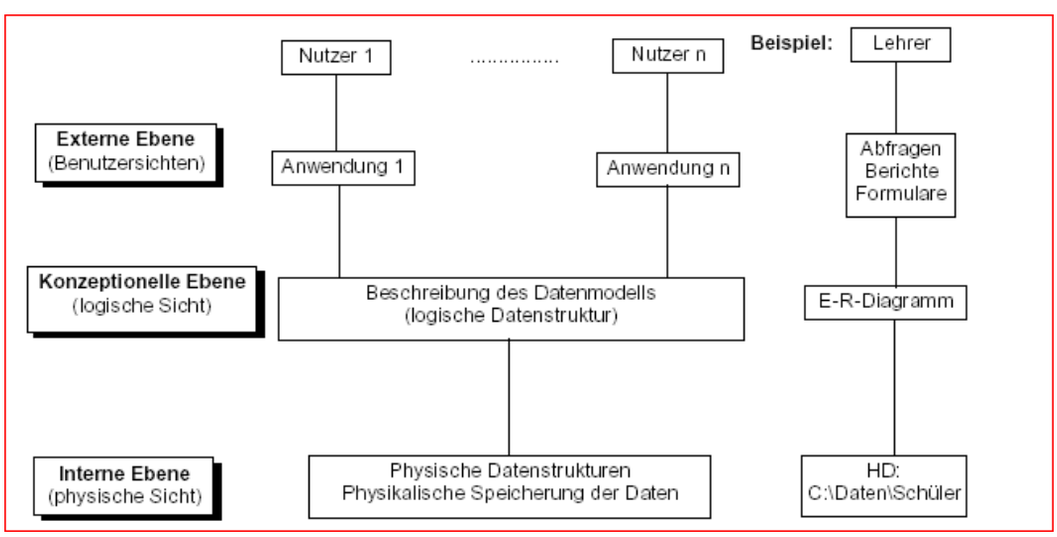
später verwendeten Hard- und Software, als auch von den jeweiligen Anforderungen der Benutzer. Diese Abstraktion von der realen Welt zu der logischen Gesamtsicht bezeichnet man als konzeptionelles Modell.

Mehrere Möglichkeiten des Datenmodells:

- relationales: Das relationale Datenmodell ist momentan am weitesten verbreitet. Es basiert rein auf Tabellen (Relationen) und deren Verknüpfungen.
- hierarchisch: Beziehungen werden durch eine Baumstruktur dargestellt, dabei hat jedes Datenobjekt genau einen Vorgänger.
- netzwerkartig: vergleichbar mit dem hierarchischen Modell

Spezielle Sicherheitsmaßnahmen bei gleichzeitigem Zugriff von mehreren Nutzern nötig -> Transaktionsmanager

Interne Ebene: Hier geht es um Realisierung der Daten auf der Computeranlage
Wo werden die Daten gespeichert? -> Speichermedien
Wie wird auf die Daten zugegriffen? ->Zugriffspfade



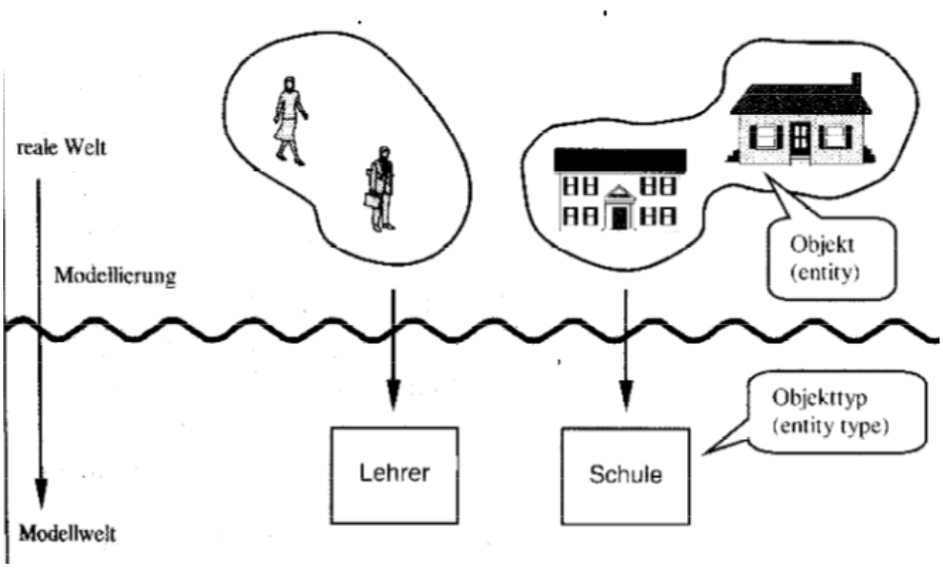
Im Idealfall sind die drei Ebenen völlig unabhängig voneinander, d. h. Änderungen in einer Ebene sollten nicht zwangsläufig Änderungen in einer anderen Ebene nach sich ziehen. Nur die Transformationen müssen den Änderungen angepasst werden.

Aufgaben:

7. Es soll eine Datenbank für eine Bücherei eingerichtet werden. Ein einzelnes Buch sei beschrieben durch die folgenden Merkmale: Inventarnummer des Buches, ISBN-Nummer, Autor, Titel, Fachgebiet, Verlag, Erscheinungsort und –jahr, Auflage, Preis. Zu welcher Ebene gehört diese Beschreibung?
8. Analysiere bei jeder der folgenden Umstellungen, auf welcher der Ebenen einer bestehenden Datenbank und an welchen Transformationen jeweils etwas geändert werden muss (im Idealfall).
   a) Neuer Rechner, aufwärts kompatibel, gleiches Betriebssystem, gleiches DBS
   b) Ein neues Programm nutzt bestehende Daten
   c) Ein neues Programm X benutzt bestehende Daten und zusätzlich neue Datenstrukturen
   d) Der Bestand einer Datenbank wird geteilt.

9. Welche Anforderungen werden also an eine Datenbank und ihr Managementsystem gestellt? Was passiert bei mehreren Benutzern, wie werden die Daten gesichert usw.

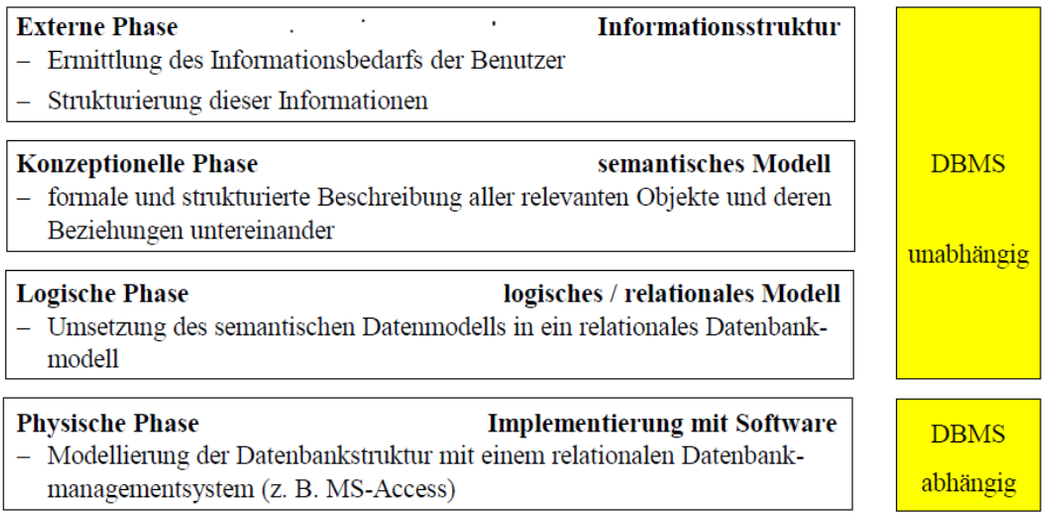
Von der Realität zum Modell



(Röhner, 2003)

Bei der Datenmodellierung geht es darum, die für die Anwendung wichtigen Faktoren von den unwichtigen zu filtern und dann in ihrem logischen Zusammenhang richtig zu erfassen. Zunächst muss also ermittelt werden, welche Informationen für den späteren Benutzer relevant sind und wie diese Informationen strukturiert sind. Anschließend muss sich der Erzeuger des Datenbanksystems mit der Frage beschäftigen, wie diese Miniwelt konzeptionell dargestellt werden kann, d. h. wie lassen sich die Objekte der Realität als Objekttyp beschreiben, welche Eigenschaften müssen über die Objekttypen gespeichert werden. Die Umsetzung erfolgt, indem das Datenmodell nun in ein relationales Datenbankmodell erzeugt wird, d. h. die notwendigen Informationen werden in Tabellenform gesammelt. Der bisherige Weg verläuft ohne Rücksichtnahme auf das später verwendete Datenbankmanagementsystem. Erst nach diesem Ablauf wird das ganze am Computer mit einer entsprechenden Software verwirklicht.

Arbeitsphasen bei der Erstellung eines Datenbanksystems



(Matzke, 2000)

Die externe Phase

Durch Gespräche mit dem Auftraggeber muss geklärt werden, welche Informationen in der Datenbank gespeichert werden sollen und welche Ausgaben später von den unterschiedlichen Benutzern erwartet werden. Aufgrund dieser Informationen muss der Datenbanksystemersteller dann die Informationsstruktur des Modells planen. Hierbei gibt es zwei grundsätzliche Ansätze:

„Top-down-Ansatz (globales Datenmodell)

Die Informationsanforderungen aller späteren Datenbanknutzer (nicht die einzelne Anwendung) bestimmt die Informationsstruktur. Beispiel: Alle für die Schule relevanten Objekte (Schüler, Lehrer, Klassen, Fächer, Eltern, Räume, Ausstattung usw.) werden erfasst. Es wird zunächst eine grobes Datenmodell entworfen und dann schrittweise verfeinert, so dass einzelne Anwendungen (Applikationen) entstehen.

Bottom-up-Ansatz (anwendungsorientiertes Datenmodell)

Ein spezielles Problem ist Ausgangspunkt für die Datenbankentwicklung. Für die Lösung des Problems wird eine Anwendung entwickelt. Beispiel: Um Zeit zu sparen, sollen die Zeugnisse und Schulbesuchsbescheinigungen über eine EDV-Anlage ausgefertigt werden. Die Integration der einzelnen Anwendungen kann zu einem globalen Datenmodell führen. „ (Matzke, 2000)

Die Ermittlung der Informationen kann dabei vorrangig aufgrund von Realitätsbeobachtungen, aufgrund von Benutzersichtanalysen oder aufgrund von Datenbestandsanalysen durchgeführt werden. Im Bereich der Benutzersichtanalysen können vorhandene Dokumente, z. B. Zeugnisse oder Klassenlisten Aufschluss über die Anforderungen an das DBS bringen. Der reduzierte Ausschnitt der Wirklichkeit wird Miniwelt genannt. Beobachtet man die Realität, so kann man bestimmte Objekte erkennen, z. B. das Fach Französisch oder den Schüler Tom Maier. Für jedes erkannte Objekt muss nun eingeschätzt werden, ob es sich um ein für die Anwendung relevantes Objekt handelt. Der Hausmeister Jan Gutauge ist sicherlich wichtig für die Schule, spielt aber beim Zeugnisdruck keine Rolle. Auch Beziehungen zwischen Objekten können beobachtet werden. Das Verfahren der Ermittlung über die Analyse bereits bestehender Datenbankbestände muss verwendet werden, wenn bereits existierende Datensätze in das neue DBS integriert werden sollen.

Die konzeptionelle Phase

Die ermittelte Miniwelt muss nun ihrem gesamten logischen Zusammenhang in einem Datenmodell abgebildet werden. Dazu müssen alle relevanten Objekttypen und ihre Beziehungen in einem semantischen Datenmodell erfasst werden. Auch hier kann sowohl mit der Top-Down-Methode (vom Groben zum Detail) oder der Bottom-up-Methode (vom Detail zum Ganzen) vorgehen. Allerdings wird vorrangig die Top-Down-Methode verwendet, da sie nicht so zeitaufwendig und kompliziert ist. Wie man in dieser Phase konkret vorgeht, folgt im Abschnitt Entity-Relationship-Modell.

Die logische Phase

Das erstellte Entity-Relationship-Modell wird nun mithilfe von Ableitungsregeln in Tabellen umgesetzt.

Die physische Phase

Die Realisierung mit Access, XAMPP oder einem anderen System werden wir näher im Teil 3 betrachten.

Beispiel: Verwaltung von Schülerdaten einer Schule

Die Schule stellt einen Teilbereich der realen Welt dar. Unter der Miniwelt versteht man nun die Beschreibung der Schule, wobei man sich nur auf die relevanten Objekte der Schule und ihre Beziehungen untereinander beschränkt. Für die Verwaltung von Schülerdaten bedeutet dies, dass unsere Miniwelt aus den Objekten Schüler, Lehrer, Bücher, Kurse, Noten und deren Beziehungen zueinander bestehen würde. Reinigungskosten, der Hausmeister oder Räume spielen bei der Verwaltung von Schülerdaten keine Rolle und werden in der Miniwelt nicht berücksichtigt. Ziel ist es, ein Modell dieser Miniwelt zu erzeugen, so dass wir Fragestellungen mithilfe dieses Modells beantworten können. Durch die Einschränkungen, die wir bei der Auswahl der relevanten Informationen vorgenommen haben, können auch nur Fragen beantwortet werden, die sich auf diese Miniwelt beschränken. Beispielsweise kann die Frage, warum ein Schüler im Abitur gescheitert ist, nicht vom Modell beantwortet werden. Hierzu fehlen die nötigen Informationen und Methoden. Deshalb darf man die externe Phase nicht vernachlässigen. Wer hier nur schlampig ermittelt, in welchem Bereich Antworten später erwartet werden, wird keinen zufriedenen Auftraggeber zurücklassen.

Mithilfe eines Datenbanksystems können die Informationen dann nicht nur gesammelt, gespeichert und gelöscht werden, sondern man kann auch neue Informationen durch die Verknüpfung von bekannten Informationen erstellen, z. B. die Berechnung des Abiturdurchschnitts aus den Einzelnoten.

Aufgaben:

1. Betrachte das Zeugnisformular[16] des Internatgymnasiums Schloss Neuschwanstein.
   a. Ergänze in der Tabelle, welche Eigenschaften die Objekte[17]: besitzen.

   b. Die Information über die erreichte Punktzahl passt nicht so recht zum Objekt Kurs und nicht so recht zum Objekt Schüler. Die Information Punktzahl ist vielmehr abhängig von beiden Objekten, also von der Beziehung Schüler ↔ Kurs, d. h. diese Beziehung hat die Eigenschaft Punktzahl. Gibt es noch andere Informationen, welche statt einem Objekt eher einer Beziehung zugeordnet werden könnten?

[16] Extra-Arbeitsblatt!!!

[17] **Achtung**: In der OOP sind Objekte immer konkrete Elemente einer Klasse. In der Datenbankmodellierung sind Objekte zunächst übergeordnete Begriffe, die eher mit Klassen vergleichbar sind.

c. Erweitere die Tabelle.

d. Für die Speicherung der Daten können sogenannte Geschäftsregeln aufgestellt werden. Eine könnte z. B. lauten: „Ein Zeugnis geht genau an eine(n) Schüler(in) und existiert ohne diesen nicht.“ Stelle weitere Geschäftsregeln auf, welche deutlich machen, in welchem Zusammenhang sich die Objekte befinden.

Entität (Objekt)	**Beziehung**	**Eigenschaft**	**Beispiel**
Schule		Schulname	Schloss Neuschwanstein
		Schulort	Wiesbaden
Kurs			
	Schüler(in) *besucht* Kurs	Punktzahl	06
Schüler(in)			
	Schüler(in) erhält Zeugnis		
Zeugnis			

„In der Informatik spricht man im Zusammenhang mit Datenbanksystemen meist nicht von Objekten, sondern benutzt den Begriff Entity oder auch Entität. Entities können demnach Personen sein, reale Objekte wie Zeugnisse oder Räume, aber auch abstrakte Objekte, wie z.B. Kurse, die nur gedanklich als ein unterscheidbares und identifizierbares Objekt existieren. Eine Entity wird im Wesentlichen durch seine Eigenschaften beschrieben, im konkreten Fall besitzt jede Eigenschaft einen Wert, z.B. die Eigenschaft „Kursbezeichnung“ der Entity „Kurs“ den Wert 3M11. Dies entspricht dem Klassen-Objekt Prinzip in der objektorientierten Programmierung. Die Attribute (Spalten) einer Tabelle, ihre Eigenschaften, bilden die

Klasse, während die konkreten Daten das Objekt bilden.“ (Burkert, Lächa, & Meyer, Version 1.000001, S. 11 (Kap. 2))

Die Tabelle oben zeigt alle wesentlichen Merkmale unserer Miniwelt, wie sie sich durch die Zeugnisformulare dokumentiert. Dies allein reicht jedoch noch nicht aus, um Abiturzeugnisse zu drucken. Dazu müssten sämtliche Kursbelegungen während der Oberstufe mit Noten und Fehlstunden verwaltet werden können.

Die Qualität des Modells unserer Miniwelt ist um so besser, je genauer die Geschäftregeln festgelegt sind, da mithilfe der Geschäftsregeln u.a. die Datenintegrität sicherstellt wird. So ist es z. B. von Bedeutung, ob ein Objekt nur in Abhängigkeit von einem anderen Objekt existieren kann. Ein Zeugnis ist immer einem bestimmten Schüler bzw. einer bestimmten Schülerin zugeordnet. Ohne Schüler auch kein Zeugnis. Die Geschäftsregeln sind ein Teil des Pflichtenhefts im Softwareengineering, indem genau festgehalten werden soll, was das Softwareprodukt leisten soll.

Aufgaben:

2. Erläutere die Begriffe Objekt, Entität (Entity), Eigenschaften, Attribute und Beziehungen.

3. Beschreibe die Vorgehensweise in der externen Phase der Entwicklung eines Datenbanksystems.

4. Die folgenden Musterrechnungen dokumentieren eine Miniwelt “Rechnungschreiben“ in einer Firma.

Herrn
Horst Staniczek
Birnbaum 3
65510 Hünstetten

Rechnungsnummer:	R123	Rechnungsdatum:	11.06.1995
Kundennummer:	K002	Rechnungsbetrag:	1397,00

Position	Artikelnummer	Bezeichnung	Anzahl	Einzelpreis	Gesamtpreis
1	A3257	Minitor 17“	2	499,00	998,00
2	A4210	Nadeldrucker	1	399,00	399,00

Firma
Irma Computer GmbH&Co KG
Am Festungswall 45
65189 Wiesbaden

Rechnungsnummer:	R457	Rechnungsdatum:	04.03.1996
Kundennummer:	K195	Rechnungsbetrag:	359,00

Position	Artikelnummer	Bezeichnung	Anzahl	Einzelpreis	Gesamtpreis
1	A3117	CD-Rom	1	359,00	359,00

a. Stelle fest, welche Objekte und Beziehungen sich daraus ableiten lassen. Stelle die Ergebnisse in einer Tabelle dar.

Entität	**Beziehung**	**Eigenschaften**
…		

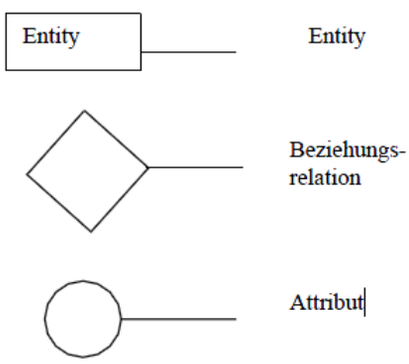
b. Formuliere die Geschäftsideen für die Miniwelt.

5. Überlege dir eine Miniwelt deiner Wahl und erstelle eine Tabelle (vgl. Nr. 4 a)) passend zu deiner Miniwelt. (Dir fällt nichts ein? Mini-Welt-Zoo, Mini-Welt-Tante-Emma-Laden,…)

Das Entity-Relationship-Modell

In der konzeptionellen Phase geht es nun darum, die gefundene Informationsstruktur in ein semantisches Modell umzuwandeln, so dass man eine formale und strukturierte Beschreibung aller Elemente der Miniwelt erhält.

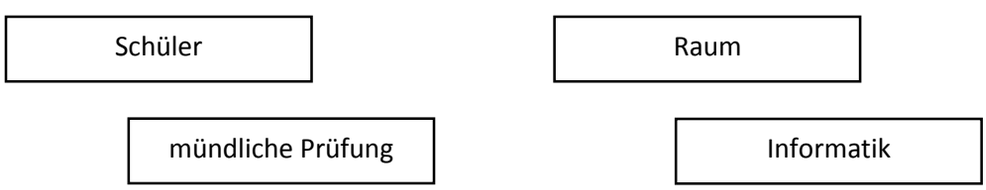
Weit verbreitet ist die Verwendung des Entity-Relationship-Modells (E-R-Modell), da es sich in der Entwicklung von relationalen Datenbanksystemen bewährt hat. Zur Modellierung werden im E-R-Modell drei verschiedene Elemente verwendet:



Entity

Eine Entität (Entity) kann eine Person, ein reales Objekt, ein abstraktes Konzept oder ein Ereignis sein. Eine Entität ist eine eindeutig identifizierbare Einheit. Eine Entitätsmenge fasst alle Entitäten zusammen, die durch gleiche Merkmale, nicht notwendigerweise aber gleiche Merkmalsausprägungen, charakterisiert sind. D. h. die Entitäten besitzen gemeinsame Eigenschaften, (Merkmale, Attribute).

Zur grafischen Darstellung verwendet man Rechtecke.



Beziehungen

Zwischen zwei Entitäten können Beziehungen (Relationships) bestehen. „Eine Beziehung assoziiert wechselseitig zwei (oder mehrere) Entitäten. Assoziation bedeutet, dass eine Entität eine andere Entität kennt und mit ihr in Wechselwirkung steht. Die Kardinalität einer Assoziation a(E1, E2) gibt an, wie viel Entitäten der Entitätsmenge E2 einer beliebigen Entität der Entitätsmenge E1 zugeordnet sein können. Die Kardinalität spezifiziert also die Anzahl der an der Assoziation möglicherweise beteiligten Entitäten (Objekte) zu jedem beliebigen Zeitpunkt.“ (Matzke, 2000, S. 25)

Beispiel: Eine Klasse kann von mehreren Schülern besucht werden: besucht(Klasse, Schüler). Ein Schüler besucht genau eine Klasse: besucht(Schüler, Klasse).

Kardinalitätstypen von Assoziationen:

Anzahl der Entitäten *E2*, die einer Entität *E1* zugeordnet werden können (a(E1, E2))	Symbol	Bezeichnung des Kardinalitäts-typs der Assoziation
genau eine	1	*einfach*
keine oder eine	c oder 0, 1	*einfach-bedingt (konditionell)*
mindestens eine oder mehrere (m >= 1)	m oder 1, m	*mehrfach (viele, komplex)*
keine, eine oder mehrere (m >= 0)	mc oder 0, m	*mehrfach-bedingt*

(Matzke, 2000, S. 25)

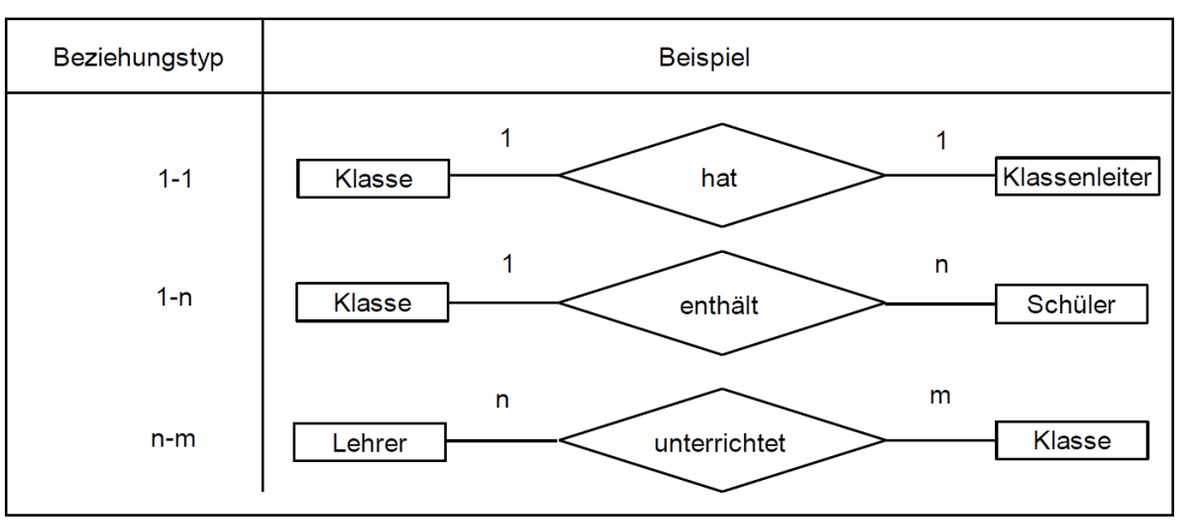
Daher unterscheidet man meist folgende drei Beziehungstypen:

1. **1:1-Beziehung**: Jedes Objekt vom Typ E1 steht höchstens mit einem Objekt vom Typ E2 in Beziehung und umgekehrt (also mit einem oder keinem).
   Beispiel: Jeder Schüler erhält höchstens ein Abiturzeugnis, und jedes Abiturzeugnis gehört eindeutig einem Schüler.

2. **1:n-Beziehung**: Jedes Objekt vom Typ E2 steht höchstens mit einem Objekt vom Typ E1 in Beziehung, es können aber mehrere (oder keiner) aus E2 zum selben Objekt von E1 eine Beziehung haben. (analog: n:1-Beziehungen)
   Beispiel: Jeder Schüler hat einen Tutor, aber der Lehrer, der Tutor ist, hat mehrere Schüler in seiner Tutorengruppe.

3. **n:m-Beziehung:** Jedes Objekt vom Typ E1 kann zu mehreren Objekten vom Typ E2 eine Beziehung haben und umgekehrt.
   Beispiel: Jeder Schüler besucht mehrere Kurse, und jeder Kurs wird von mehreren Schülern besucht.

Grafisch werden die Beziehungstypen in einer Raute dargestellt. Die Kardinalitäten werden am gegenüberliegenden Entitätentyp notiert.



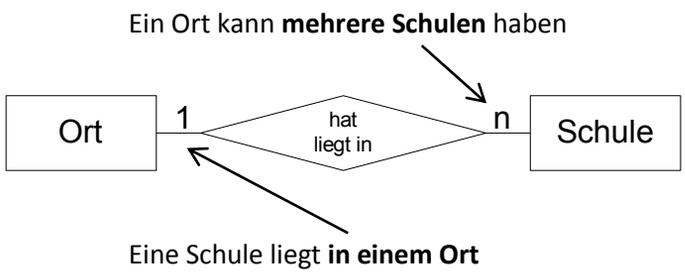
(Matzke, 2000, S. 26)

Tipps:

1. Wie finde ich die Kardinalitäten heraus? Stelle die Frage: **Kann ein Objekt** des Typs A **mit mehreren** Objekten des Typs B in Beziehung stehen?

   – Ja → Kardinalität ist n
   – Nein → Kardinalität ist 1

   Beispiele:
   Kann ein Mann mit mehreren Frauen verheiratet sein? Nein, Kardinalität ist 1.
   Kann ein Ort mehrere Schulen haben? Ja, Kardinalität ist n.
2. An welche Seite kommt die Kardinalität?



   Die Kardinalität wird an das Ende der Beziehung geschrieben.

Attribut

Ein Attribut beschreibt eine bestimmte Eigenschaft, die alle Entitäten einer Entitätenmenge oder sämtliche Einzelbeziehungen einer Beziehung ausweisen. Vergleichbar ist dies mit den Eigenschaften einer Klasse in der OOP. Attribute können dabei nur einen einzigen Wert umfassen (d. h. sie sind atomar) aus mehreren Attributen bestehen. Es kann vorkommen, dass ein spezielles Objekt nicht nur eine Attributbelegung sondern mehrere besitzt. Solche Attribute nennt man Mehrfachattribute.

Beispiele:

- einfaches Attribut mit einfacher Belegung: Geschlecht
- Attribut mit mehreren Attributteilen: Adresse – Straße, Hausnummer, PLZ, Ort
- Mehrfachattribut: zuvor besuchte Schule

Grafisch werden Attribute durch Kreise oder Ovale veranschaulicht. Mehrfachattribute erhalten einen Doppelkreis.[18]

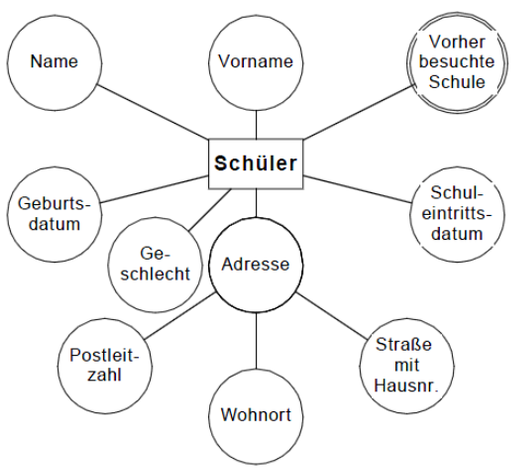
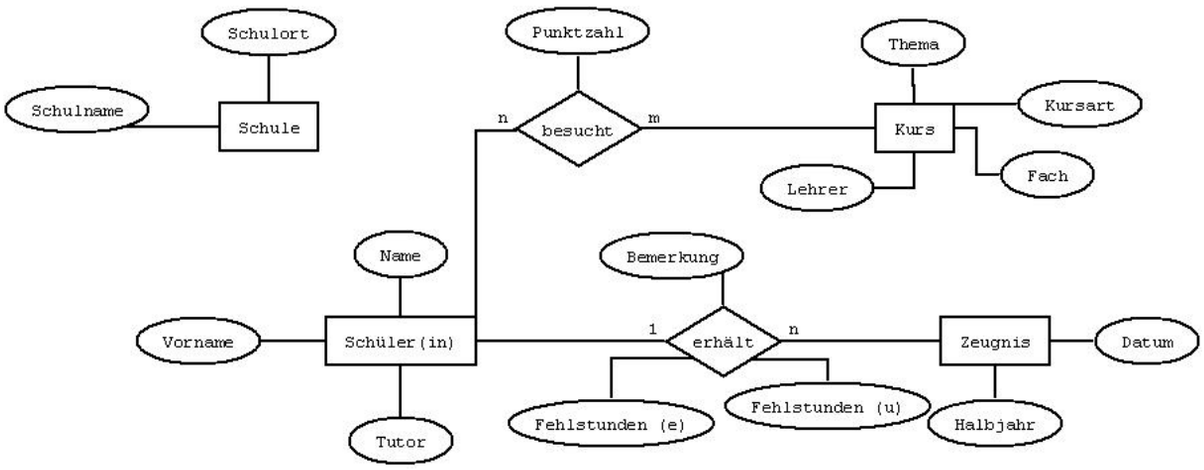


Abbildung: ER-Modell Schüler

[18] Abbildung: (Burkert, Lächa, & Meyer, Version 1.000001, S. 19, (Kap 2))

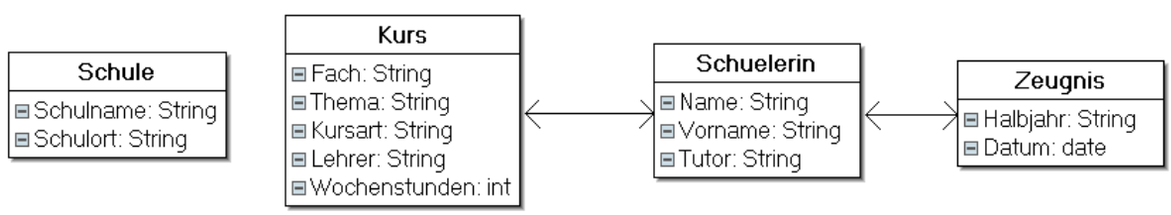
Übertragung der Informationsstruktur in ein E-R-Modell

Unter Berücksichtigung der Geschäftsregeln kann man dann die gesammelten Informationen in einem ER-Diagramm darstellen. Für das Beispiel des Zeugnisdrucks ergibt sich folgendes ER-Diagramm:



Wie man sieht, besteht aktuell noch keine Beziehung zwischen der Schule und den anderen Objekttypen.

Ein ER-Diagramm unterscheidet sich also nicht in seinem Informationsgehalt von einem Klassendiagramm (OOP), sondern in der äußeren Darstellung. Jede Entity entspricht einer Klasse und jede Relationship einer Beziehung zwischen diesen. Das Klassendiagramm zu unserem Zeugnisdruck-Problem würde wie folgt aussehen:



Primärschlüssel – Schlüssel- Sekundärschlüssel

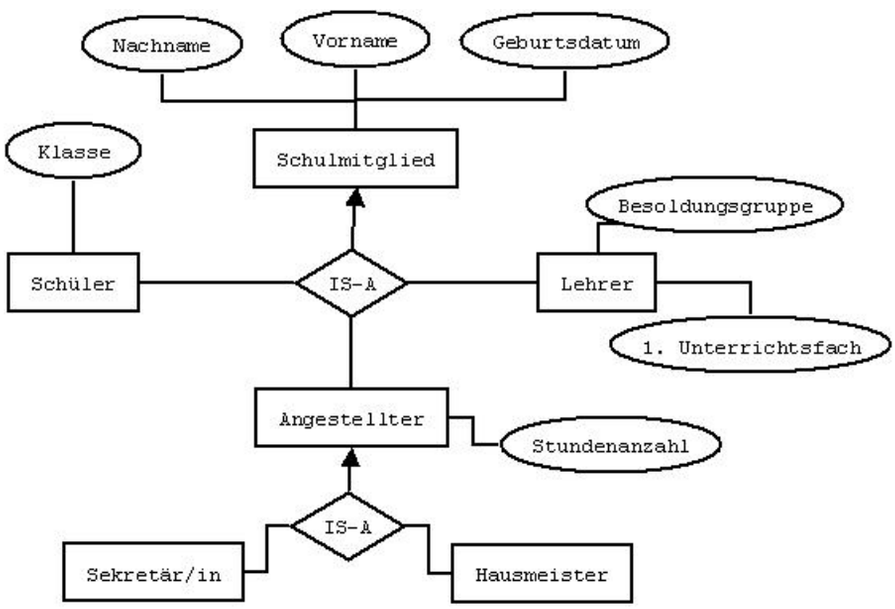
Bevor wir die konzeptionelle Phase verlassen können, müssen die Attribute noch bezüglich eines eindeutigen Erkennungszeichens untersucht werden. Ein oder mehrere Attribute, die eine Entität eindeutig charakterisieren, so dass sie mit keiner anderen Entität desselben Typs verwechselt werden kann, nennt man Schlüssel. Die Wahl eines Schlüssels ist häufig nicht eindeutig. Wünschenswert ist ein Identifikationsschlüssel der aus möglichst wenigen Attributkombinationen besteht. Um kenntlich zu machen, welche Attribute als Primärschlüssel, also als eindeutiges Identifikationsmerkmal, verwendet werden sollen, unterstreicht man diese Attribute im ER-Diagramm. In der Praxis verwendet man häufig automatisch vergebene Primärschlüssel, die die Daten durchnummerieren. Für unser obiges Beispiel wäre z. B. für den Schüler eine Schülernummer als Primärschlüssel denkbar. Besteht der Primärschlüssel aus mehreren Attributen, so spricht man vom zusammengesetzten Primärschlüssel.

Ein Sekundärschlüssel kann für mehrere Entitäten eines Typs den gleichen Wert annehmen und dient so zur Zusammenfassung von Entitäten mit gleichen Eigenschaften, z. B. könnte das Attribut Tutor der Entity Schüler als Sekundärschlüssel verwendet werden.

IS-A-Beziehung

Vergleichbar mit der Vererbungstheorie beim objektorientierten Modellieren gibt es auch Beziehungen zwischen Entitäten, die verdeutlichen sollen, dass es sich um eine Spezialisierung einer Klasse bzw. von der anderen Seite aus betrachtet um eine Generalisierung einer Klasse handelt. Diese Beziehung wird als IS-A-Beziehung bezeichnet.

Im ER-Diagramm wird sie durch einen Pfeil gekennzeichnet. In manchen Darstellungen fehlt der Pfeil auch, aber die Beziehung ist mit „IS-A“ benannt.



Alle Schulmitglieder besitzen gemeinsame Eigenschaften, z. B. hat jeder einen Namen und jeder ein Geburtsdatum. Alle gemeinsamen Eigenschaften werden an die Entität „Schulmitglied“ angehängt. Alle Spezialisierungen von Schulmitglied erben diese Eigenschaften. Sie müssen daher nicht mehr extra aufgeführt werden. Jede Spezialisierung kann zusätzlich noch Attribute besitzen, die nur sie hat. So haben Lehrer eine Besoldungsklasse und Schüler eine Klassenzugehörigkeit.

Aufgaben:

1. Es soll eine Datenbank erstellt werden, die alle Flüge rund um die Welt enthält, so dass sich registrierte Kunden im Buchungssystem über aktuelle Flüge informieren können und Flüge buchen können. Jeder Buchung wird dabei eine Buchungsnummer zugewiesen. Um den Kunden die Suche zu erleichtern, werden nicht nur die Flugnummern sondern auch die Fluglinie, der Abflugs- und Ankunftsorte, sowie die Abflugs- und Ankunftszeiten gespeichert. Die Anzahl der maximalen Flugpassagiere wird ebenfalls angegeben. Für die Registrierung müssen die Kunden neben ihrem Vor- und Nachnamen auch ihr Alter angeben. Sie erhalten dann eine Kundennummer vom System zugewiesen.
    a. Erstelle ein ER-Diagramm für die oben gemachten Angaben.
    b. Kennzeichne jeweils den Primärschlüssel.
    c. Ein Primärschlüssel kann durch ein künstliches Merkmal oder durch ein natürliches Merkmal bzw. eine Kombination von natürlichen Merkmalen gebildet werden. Um welche Art von Schlüssel handelt es sich jeweils?

2. Überlege, welche Komplexität die IS-A-Beziehung besitzt.

3. Gegeben sind jeweils zwei Entitytypen und ein Beziehungstyp. Gib die jeweilige Komplexität an.

	E-Typ 1	E-Typ 2	Beziehungs-Typ
a.	Vater	Tochter	hat
b.	Fuß	Zehe	gehört zu
c.	Onkel	Neffe	hat
d.	Schüler	Lehrer	hat Unterricht
e.	Person	Personalausweis	besitzt
f.	PROGRAM	Methode	benötigt
g.	Bruder	Schwester	hat
h.	Ort	Ort	kürzeste Entfernung

4. Der folgende Ausschnitt aus einem Kursverzeichnis dient zur Kurswahl der Schülerinnen und Schüler zur Q-Phase. Ergänze mit den darin befindlichen Kursdaten das Modell auf Seite 60!

Leiste G6: Mi 9./10. Std. und Fr 5. Std

5 D 13 Kreß	165	Literatur und Gesellschaft
5 D 16 Biedermann	164	Literatur und Gesellschaft
5 E 12 Meyer K.	127	The Individual and Society
5 K 10 Blum	7	Kunst und Gesellschaft

Leiste G8: Do 1.2. Std. und FR 6. Std

5 D 10 Biedermann	170	Literatur und Gesellschaft
5 D 17 Beste	164	Angst
5 E 11 Meyenburg	125	The Individual and Society
5 K 11 Repert	12	Kunst und Gesellschaft
5 Mu 10 Lamprecht	140	Musik und Gesellschaft

5. Im Folgenden soll eine Datenbank für eine Buchausleihe modelliert werden. Der Informatiklehrer macht folgenden Vorgaben:

   - Es sollen alle Informatikbücher der Modellschule Obersberg in einer Datenbank erfasst werden.
   - Informatikbücher gibt es im Moment in der Mediathek zur Individualausleihe und in der Lernmittelbücherei. Weiterhin gibt es Informatikbücher in den Informatikräumen der gymnasialen Abteilung und zwar als Einzelexemplare und in jeweils größerer Stückzahl. Auch die kaufmännische Abteilung verwaltet in Eigenregie Bücher aus diesem Themengebiet. Weitere mögliche Standorte sollen vorgesehen werden.
   - Es sollen über ein Buch gespeichert werden: Autor, Titel, Themengebiet, Ort der Aufbewahrung, Ausleihstatus (Präsenz, übers Wochenende, vier Wochen, halbes Jahr), Erscheinungsjahr, Verlag, Stichworte (bis zu 20 pro Buch), Kurzbeschreibung.
   - Eine Suche soll nach Themengebiet, Autor, Titel und Stichwort möglich sein.

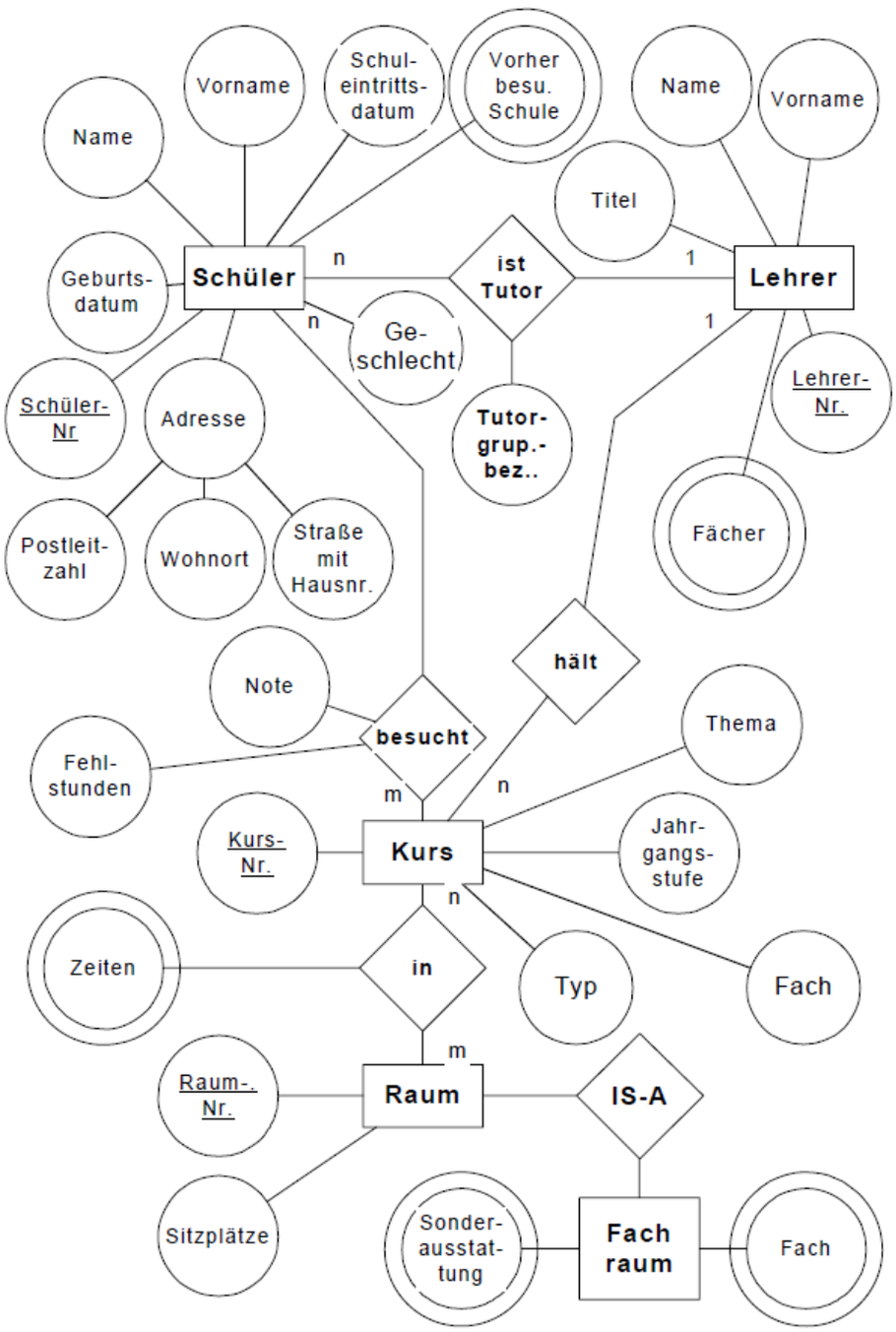
- Weiterhin soll mit dieser Datenbank die Ausleihe in den Informatikräumen der gymnasialen Abteilung organisiert werden. Dazu sollen die Ausleiher erfasst werden (Schüler und Lehrer) mit Namen und Adresse, bei den Schülern außerdem Klasse und Tutor. Auf Lesekarten und Ausweise kann verzichtet werden. Die Ausleihe soll von den Lehrern durchgeführt werden.
- Lehrer und Schüler können mehrere Bücher ausleihen. Die meisten Bücher sind schon mit einer Inventarnummer versehen (10-stellige Zahl).
- Es muss möglich sein, neue Entleiher zu erfassen, bzw. alte zu löschen.
- Entleih- und Rückgabevorgänge müssen durchgeführt werden können.
- Das Programm soll ausgeben können: Entleiher eines bestimmten Buchtyps, alle entliehenen Bücher eines bestimmten Entleihers, Mahnungen an alle Entleiher, die ihre Frist überschritten haben.

Das so zusammengestellte Anforderungsprofil könnte nach gründlicher Durchsicht z.B. noch folgende Änderungen annehmen:

- Auf die Speicherung der Adresse kann verzichtet werden, da die Schüler leichter über ihren Tutor zu erreichen sind (zudem keine Postgebühren) und die Lehrer Fächer im Lehrerzimmer besitzen. In Ausnahmefällen kann auf die Adresse im Sekretariat zurückgegriffen werden.
- Beim Entleihen sollte ein Zettel für den Entleiher gedruckt werden, der neben dem Titel des Buches auch ein Rückgabedatum enthält.
- Die Datenbank sollte auf dem Server des Netzwerkes geführt werden und von jedem Rechner mit jeder Zugangsberechtigung eine Suche nach Buchtiteln möglich sein.
- Die Ausleihe kann nur mit einem Lehrerpasswort durchgeführt werden. Die Frage, ob das auf jedem Rechner möglich sein soll oder nur auf einem speziellen dafür vorgesehenen, wird zunächst offengelassen.
- Für die Bücher an „fremden“ Standorten ist es nicht notwendig, Signaturen zu speichern, da ohnehin keine Ausleihe von der zu entwickelnden Software durchgeführt werden soll. Es würde bei diesen Büchern ausreichen, neben den allgemeinen Angaben und dem Standort die Anzahl der maximal vorhandenen einzugeben, damit der Sucher eine Vorstellung hat, wo und wie oft das Buch vorhanden ist.

Mit diesen Festlegungen können die Entities und die Beziehungen zwischen ihnen bereits angegeben werden. Überlege dir zunächst, welche Entities du benötigst und welche Beziehungen bestehen. Dies kannst du in einer Tabelle festhalten. Erstelle anschließend ein ER-Diagramm.

6. Beschreibe das ER-Modell zur Schulverwaltung verbal. Dazu gehört auch, dass jede Beziehung verbal beschrieben wird, z. B. „Jeder Lehrer unterrichtet eine Klasse. Jede Klasse wird von einem Lehrer unterrichtet.“ Verwende in deiner Beschreibung die Begriffe Generalisierung bzw. Spezialisierung, zusammengesetztes Attribut, Mehrfachattribut, Beziehungen der Komplexität 1:1 / 1:n / n:m und Primärschlüssel.

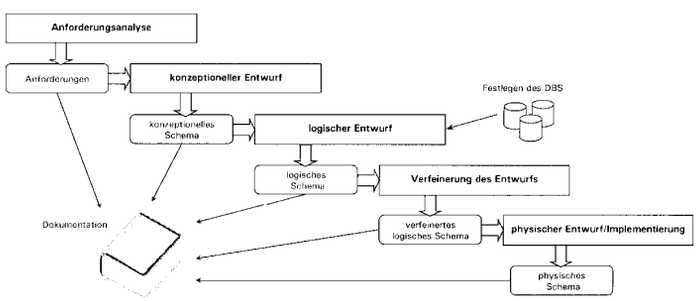


7. Die Computerzubehörfirma Microtec GmbH möchte ihre Verwaltung auf EDV umstellen. Sie verkauft ein Sortiment von Artikeln, die sie von verschiedenen Herstellern bezieht. Außerdem hat sie einen bestimmten Kundenkreis, der bei ihr Bestellungen aufgibt. Eine Bestellung kann natürlich mehrere Artikel umfassen. Derselbe Artikel kann oft von mehreren Herstellern bezogen werden, und ein Hersteller liefert natürlich meist mehr als einen Artikel.

Erstelle im Entity-Relationship-Modell ein sinnvolles Datenmodell für die Firma, das Datenredundanz vermeidet. Wähle geeignete Entities mit notwendigen Attributen und gib die zwischen den Entities bestehenden Beziehungen mit ihrem Komplexitätsgrad an.

8. Zugrunde gelegt werde das ER-Modell der Schulverwaltung aus Aufgabe 5. Gib bei den nachfolgenden Änderungen an, welche Ebene des DBMS von der Änderung betroffen ist.
    a. Der Lehrer Franz Schlauspruch kommt neu an die Schule.
    b. Die Lehrer sollen bei der Noteneingabe nicht mehr die Noten des Schülers bei anderen Lehrern abfragen können.
    c. Bei den Schülerdaten soll durch eine zusätzliche Indexdatei, in der die Schüler nach Wohnort sortiert sind, ein schnellere Suche nach Schülern eines Ortes ermöglicht werden, um die Busverbindungen besser koordinieren zu können.
    d. Zur Erstellung von Altersstatistiken soll auch bei Lehrern das Geburtsdatum gespeichert werden. Der Raum 556 soll als neuer Fachraum für Mathematik verwendet werden.
    e. Auf Wunsch des Hausmeisters wird zusätzlich erfasst, welcher Lehrer einen Schlüssel für welchen Fachraum hat.
    f. Neben dem/der Oberstufenleiter/in sollen auch die Tutoren die komplette Belegung ihrer Schüler einschließlich der bisher vergebenen Noten am Computer einsehen können.
    g. Vom Sekretariat sollen die Schüler, die mehr als drei Grundkurse unter 5 Punkten einbringen müssen, per Serienbrief auf die Gefahr der Nichtzulassung zum Abitur hingewiesen werden.
    h. Auf Antrag der SV dürfen Lehrer, die nicht Tutor des entsprechenden Schülers sind, die Fehlstundenzahl nicht mehr einsehen.
    i. Daraufhin beschließt die Schulkonferenz, die Fehlstundenspeicherung ganz abzuschaffen.

9. Erläutere das dargestellte Wasserfallmodell:



Datenbank-Entwurfsphasen (Wasserfallmodell)

10. Hugo Unbedarft besitzt eine große Spedition. Er will seine Auftragsverwaltung auf EDV umstellen und macht sich dazu einen genauen Plan. Seine Aufträge sind immer so, dass sie nur zu einem Ziel führen, es kann allerdings möglich sein, dass mehrere LKWs für einen Auftrag nötig sind. Nicht jeder LKW-Typ ist dazu geeignet, alle Ziele zu erreichen (z.B. zu niedrige Brücken), und nicht jeder Fahrer kann jeden LKW-Typ fahren. Hugo will folgende Daten speichern: AuftragsNr. und LKW-Nr., Ziel, Zielentfernung, Auftragsdatum, LKWTyp, max. Zuladung eines LKW-Typs, TÜV-Datum, Fahrer-Nr, Fahrer-Name. Erstelle ein ER-Modell für die Spedition.

Das relationale Datenbankmodell

Als nächster Schritt folgt die Umsetzung des semantischen Modells, also des ER-Diagramms, in ein logisches Modell. Wir beschränken uns, wie bereits vorne erwähnt, auf das relationale Modell. Das relationale Datenbankmodell wurde bereits 1970 von dem Briten Codd entwickelt. Der Begriff Relation kommt von der Tabellenstruktur. Relationen lassen sich gut als Tabellen darstellen.

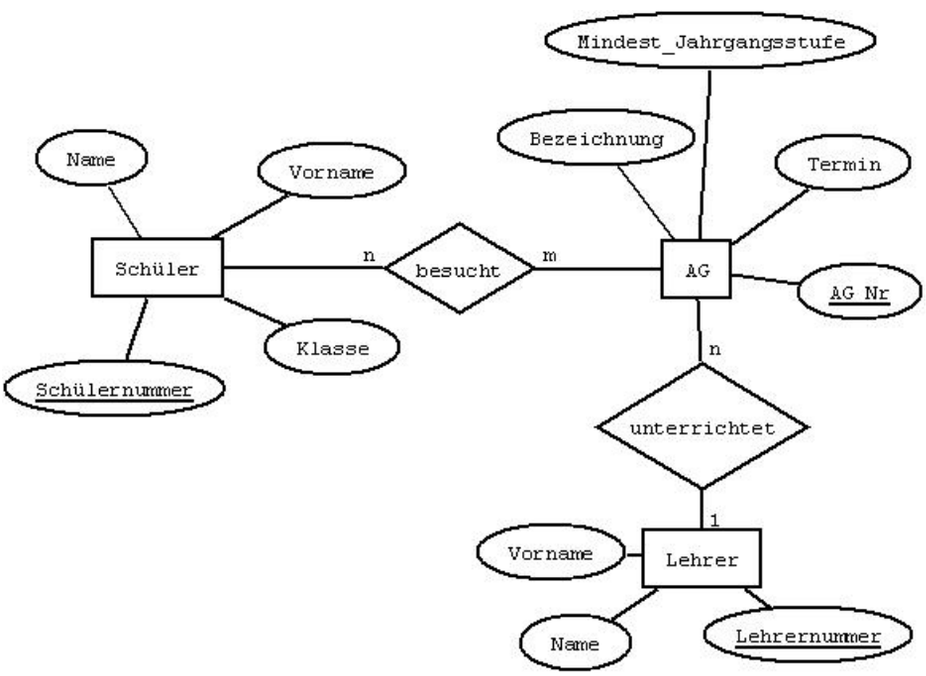
Abbildung des ER-Modells auf ein relationales Datenbankmodell

Wie kann ein ER-Modell nun in ein relationales Datenbankmodell überführt werden?

Die Entitäten:

Für die Umsetzung wird zunächst jede Entität als Tabelle abgespeichert, wobei die Attribute der Entität das Schema der Tabelle und die Attributwerte einzelner Objekte dieser Entität einen Datensatz bilden.

Beispiel:



(Burkert, Lächa, & Meyer, Version 1.000001, S. 23 (Kap. 2))

In diesem ER-Modell steckt noch eine kleine Nachlässigkeit. Die Attribute Name und Vorname tauchen sowohl beim Schüler als auch beim Lehrer auf. Dies führt später zu Schwierigkeiten, daher ergänzen wir den Schülernamen und den Schülervornamen in der Tabelle jeweils um ein vorgestelltes großgeschriebenes S, entsprechend verfahren wir beim Erstellen der Lehrertabelle.

SCHÜLER				Überschrift der Tabelle, Relationenname
Schülernummer	SName	SVorname	Klasse	Schema der Tabelle, Relationenschema
004	Frei	Paula	9a	Datensatz
005	Frei	Markus	9c	Datensatz
099	Miller	Johanna	5c	Datensatz
549	Zungel	Jakob	7b	Datensatz

LEHRER		
Lehrernummer	LName	LVorname
L342	Linde	Erika
L921	Müller	Markus
L012	Prinz	Johann
L452	Amberger	Anna

AG			
AG_Nr	Bezeichnung	Termin	Mindest_Jahrgangsstufe
AG_01	Foto-AG	Mo. 7./8.	8
AG_02	Badminton-AG	Do. 9./10.	5
AG_03	DELF-AG	Di. 7./8.	9
AG_04	Orchester	Fr. 7./8.	8

Die Beziehungen:

Beziehungen werden als eigene Beziehungstabelle umgesetzt. Dazu werden die Primärschlüssel der beiden Entitäten, die miteinander in Beziehung stehen, als ein Datensatz in der Tabelle gespeichert. Wenn Paula an der Foto-AG teilnimmt, wird dies durch den Datensatz (004, AG_01) ausgedrückt.

Die Tabellen sehen dann wie folgt aus:

BESUCHT	
Schülernummer	AG_Nr
004	AG_01
004	AG_03
005	AG_01
005	AG_04
099	AG_02
549	AG_02

Paula ist nicht nur in der Foto-AG sondern auch in der DELF-AG.

Markus ist ebenfalls in der Foto-AG. Zusätzlich ist Markus im Orchester.

Jakob und Johanna sind in der Badminton-AG.

Die Attribute Schülernummer und AG_Nr verweisen auf die Primärschlüsselattribute der beiden (fremden) Tabellen von den Entitäten Schüler und AG. Sie werden als **Fremdschlüssel** bezeichnet. In einer n:m-Beziehung kann jeder Fremdschlüssel mehrmals vorkommen, jede Kombination aus Fremdschlüssel kommt jedoch nur einmal vor.

UNTERRICHTET	
Lehrernummer	AG_Nr
L342	AG_01
L921	AG_03
L452	AG_04
L342	AG_02

Fr. Linde unterrichtet die Foto-AG und die Badminton-AG. Hr. Müller unterrichtet die DELF-AG und Fr. Amberger leitet das Orchester.

Da jede AG nur von einem Lehrer geleitet wird, kann man die Anzahl der Tabellen reduzieren, indem man die Leitung der jeweiligen Kurse an als weitere Spalte an die Tabelle AG anhängt, d. h. man benötigt keine eigene Beziehungstabelle, wenn die Beziehung eine Kardinalität von n:1 oder 1:1 aufweist. Die erweiterte Tabelle AG_MIT_LEITUNG sieht dann folgendermaßen aus:

AG				
AG_Nr	Bezeichnung	Termin	Mindest_Jahrgangsstufe	Lehrernummer
AG_01	Foto-AG	Mo. 7./8.	8	L342
AG_02	Badminton-AG	Do. 9./10.	5	L342
AG_03	DELF-AG	Di. 7./8.	9	L921
AG_04	Orchester	Fr. 7./8.	8	L452

Auf die Tabelle UNTERRICHTET kann verzichtet werden, ohne dass die Informationen verloren gehen.

Das relationale Datenbankmodell

Alle gewonnenen Tabellen bilden das relationale Datenbankmodell. Ergänzt man zusätzlich die Datentypen, so erhält man folgendes Schema für unser Beispiel:

Entitäten:

SCHÜLER[Schülernummer: Zahl; SName: Text; SVorname: Text, Klasse: Text]
LEHRER[Lehrernummer: Text, LName: Text; LVorname: Text]
AG[AG_Nr: Text; Bezeichnung: Text; Termin: Text; Mindest_Jahrgangsstufe: Zahl;
Lehrernummer: Text]

Beziehungen:

BESUCHT[Schülernummer: Zahl; AG_Nr: Text]

Zusammengefasst können wir folgende **Abbildungsregeln** festhalten:

1. Jede Entitätsmenge muss als eigenständige Relation definiert werden.
2. Entitäten, die über eine 1:1-Beziehung verknüpft sind, können durch einer gemeinsamen Relation definiert werden. Aus Datenschutzgründen oder für selten benötigte Informationen kann jedoch für jede Entität eine eigene Relation erstellt werden.
3. Eine n:1-Beziehung wird realisiert, indem der 1-Relation der Fremdschlüssel der n-Relation hinzugefügt wird. Eine eigene Relation ist nicht notwendig.
4. Jede n:m-Beziehung muss als eigenständige Relation definiert werden. Die Primärschlüssel der beteiligten Entitäten treten als Fremdschlüssel in der Beziehungsrelation auf.

Konzepte des relationalen Datenbankmodells

Das relationale Datenbankmodell basiert auf dem mathematischen Begriff der Relation.
Unter einer mathematischen Relation versteht man eine Teilmenge R des kartesischen Produkts von Wertebereichen $W_1 \times W_2 \times W_n$. Die Element einer Relation sind n-Tupel $(v_1, v_2, ..., v_n) \in R$ mit $v_i \in W_i$

Beispiel:
AG[AG_Nr; Bezeichnung; Termin; Mindest_Jahrgangsstufe; Lehrernummer]
mit den folgenden Wertebereichen: AG_Nr : W_1 = Text
Bezeichnung: W_2 = Text
Termin: W_3 = Text
Mindest_Jahrgangsstufe: W_4 = Zahl
Lehrernummer: W_5 = Text
mögliches Element: AG_1 = (AG_01, Foto-AG, Mo. 7./8., 8, L342)
Es handelt sich um eine 4-elementige Relation AG = {AG_1, AG_2, AG_3, AG_4}
Diese Darstellung ist sehr unübersichtlich, daher verwendet man die Tabellenschreibweise (siehe oben). Jede Relation hat einen Relationennamen, hier AG. Jeder Datensatz ist ein Tupel der Relation, d. h. ein Element der Relation.

Zusammenhang zwischen relationalem Modell und dem ER-Modell

Relationales Modell	Beschreibung	ER-Modell	Darstellung
Relationenname	Name der Tabelle	Name des Entitytyps	Name im Rechteck
Attribut	Spalte einer Tabelle	Attribut	Oval
Relationenschema	Menge von Attributen	Entitytyp	Rechteck samt Ovalen
Tupel	Zeile einer Tabelle	Entity	-
Relation	Menge aller Zeilen (ohne Relationenschema)	Entitymenge	-

Wichtig: Es dürfen nie zwei gleiche Zeilen vorkommen. Die Reihenfolge der Zeilen darf keine Rolle spielen.

Aufgaben:

1. Bilde das ER-Modell von Aufgabe 6, Seite 64 auf das relationale Modell ab. Gib hierzu alle benötigten Relationen in der Form Relation[Relationenschema] an.
2. Bilde das ER-Modell von Aufgabe 5, Seite 63 auf das relationale Modell ab. Gib hierzu alle benötigten Relationen in der Form Relation[Relationenschema] an.

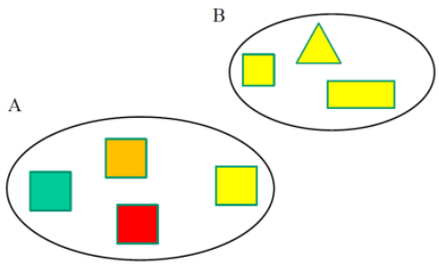
Operatoren des Relationenmodells

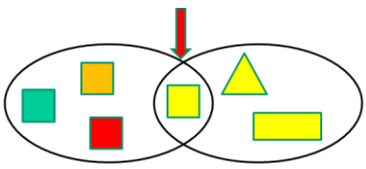
Bisher haben wir die rein statischen Eigenschaften des ER-Modells in das Relationenmodell übertragen. Eine Datenbank kann jedoch nur dann effektiv genutzt werden, wenn sie auch dynamische Eigenschaften modellieren kann. Durch die Anwendung der Operatoren auf die Relationen können z. B. neue Relationen erzeugt werden.

Die Relationenalgebra

Für die Verarbeitung der in den Relationen gesammelten Daten benötigt man Kenntnisse aus der Relationenalgebra. Die Relationenalgebra leistet die Verknüpfung unterschiedlicher Tabellen und somit fügt sie das wieder logisch zusammen, was aus Modellierungsgründen getrennt wurde.

Früher wurde die Relationenalgebra unter dem Begriff Mengenlehre in der Grundschule vermittelt, da dies heute nicht mehr so ist, müssen wir hier zunächst einige Grundlagen klären.
Betrachten wir zunächst die beiden Ausgangsmengen A und B:



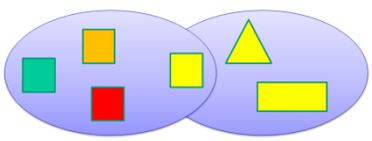


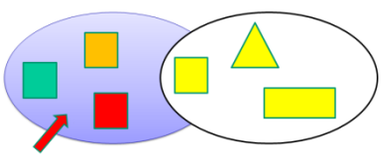
Der Durchschnitt A ∩ B

Der Durchschnitt ist die Menge aller Elemente, die sowohl in A als auch in B ist.

Die Vereinigung A ∪ B

Die Vereinigung ist die Menge aller Elemente, die in A oder in B oder in beiden Mengen vorkommt.





Die Differenz A\ B

Die Differenz (A ohne B) ist die Menge aller Elemente, die in A aber nicht gleichzeitig in B enthalten sind.

Operatoren des Relationenmodells

Operator	Schreibweise	Bedeutung
Durchschnitt	$R \cap S$	Schnittmenge
Vereinigung	$R \cup S$	Vereinigungsmenge
Differenz	$R \setminus S$	Differenz von Mengen
Produkt	$R \times S$	kartesisches Produkt zweier Mengen
Selektion	$\sigma_{\text{Formel}}(R)$	Auswahl von Tupel gemäß Formel, Streichung von Zeilen
Projektion	$\pi_{\text{Attribute}}(R)$	Auswahl von Attributen, Streichung von Spalten
Join (natural)	$R \rhd\lhd S$	Verknüpfung zweier Relationen zu einer neuen mit den Attributen beider Tabellen über gemeinsames Attribut
Join (Theta)	$R \overset{\rhd\lhd}{_C} S$	Verknüpfung zweier Relationen zu einer neuen mit formulierter Bedingung C

Dabei wird zwischen **Sets** und **Bags** unterschieden. (Burkert, Lächa, & Meyer, Version 1.000001, S. 4, (Kap 3))

„Sind Mehrfachvorkommen eines Tupels (Zeile) erlaubt, so spricht man von Bags (Eselsbrücke: Die Ergebnistupel einfach ohne Nachbearbeitung in die ,,Tasche (bag)“ werfen). Werden Duplikate entfernt handelt es sich um Sets. Demnach werden temporär (zeitweilig) erst Bags gebildet, aus denen dann mehrfach vorkommende Elemente entfernt werden, um so als finales Ergebnis ein Set zu erhalten. Wir gehen im Folgenden von Sets aus.“[19]

Das kartesisches Produkt A × B

Hierbei handelt es sich um eine künstliche Operation (keine Verbindung zur Mengenalgebra). Sie ist in Verbindung mit der Operation Join, die wir kennen lernen werden, wichtig. Mithilfe des kartesischen Produktes erhält man alle möglichen Kombinationen aus den beiden Relationen A und B.

Die Selektion σ_{Formel}(Relation)

Die Selektion σ (Sigma) wählt die Zeilen einer Relation aus, die eine bestimmte Formel erfüllen, d. h. es werden Datensätze aus einer Relation gefiltert, die bestimmte Voraussetzungen erfüllen. Diese Voraussetzungen werden in einer Formel als Index an das σ angegeben. Die Formel kann Konstanten Attribute sowie Vergleichsoperatoren (<, ≤, =, ≠, ≥, >) und logische Operatoren (and, or , not) enthalten. Die Relation, aus der die Datensätze gefiltert werden wird in Klammern hinter σ angegeben.

[19] (Burkert, Lächa, & Meyer, Version 1.000001, S. 4 (Kap. 3)) Die Tabelle stammt ebenfalls aus dieser Quelle.

Beispiel: AG_ab_Jahrgang8 = $\sigma_{Mindest\_Jahrgangsstufe \geq 8}$ (AG) führt zu folgender Tabelle:

AG_ab_Jahrgang8				
AG_Nr	Bezeichnung	Termin	Mindest_Jahrgangsstufe	Lehrernummer
AG_01	Foto-AG	Mo. 7./8.	8	L342
AG_03	DELF-AG	Di. 7./8.	9	L921
AG_04	Orchester	Fr. 7./8.	8	L452

Die Projektion $\pi_{Attribute}$(Relation)

Die Projektion π wählt die Spalten einer Relation aus, die durch die Attribute angeben werden, d. h. es werden nur bestimmte Spalten einer Relation ausgegeben.

Beispiel: AG_Bez_Lehrer = $\pi_{AG\_Nr,\ Bezeichnung,Lehrernummer}$(AG)

AG_Bez_Lehrer		
AG_Nr	Bezeichnung	Lehrernummer
AG_01	Foto-AG	L342
AG_02	Badminton-AG	L342
AG_03	DELF-AG	L921
AG_04	Orchester	L452

Sowohl bei der Projektion als auch bei der Selektion kann eine Relation zuvor dadurch erzeugt werden, dass die Schnittmenge, der Durchschnitt oder die Differenz von zwei Relationen gebildet wird.

Join (natürlicher Verbund) – Tabellen miteinander verbinden

Der Join verbindet zwei Tabellen über gleichnamige Spalten bei gleichen Attributwerten miteinander. „Der natürliche Verbund ist äußerst wichtig, um Relationen, die aus entwurfstheoretischen Gründen zerlegt wurden, während der Abfrage wieder zu kombinieren. Die Zerlegung findet in der Regel über Schlüsselattribute statt. Dementsprechend findet der Join in aller Regel über ein gemeinsames Schlüsselattribut in den beiden verknüpften Tabellen statt.“[20]

A >< B: Zunächst wird das kartesische Produkt A × B der beiden beteiligten Tabellen A und B gebildet. Für jedes Attribut, das sowohl in A als auch in B vorkommt, selektiert man die Tupel, für die die Werte der gleichnamigen Attribute übereinstimmen. Eine der gleichen Spalten wird weg projeziert. (Ich sehe ganz genau die vielen Fragezeichen in euren Gesichtern!)

Ein Beispiel: Wir verwenden die beiden folgenden Tabellen:

AG				
AG_Nr	Bezeichnung	Termin	Mindest_Jahrgangs-stufe	Lehrer-nummer
AG_01	Foto-AG	Mo. 7./8.	8	L342
AG_02	Badminton-AG	Do. 9./10.	5	L342
AG_03	DELF-AG	Di. 7./8.	9	L921
AG_04	Orchester	Fr. 7./8.	8	L452

[20] (Burkert, Lächa, & Meyer, Version 1.000001, S. 8, (Kap. 3))

LEHRER		
Lehrernummer	LName	LVorname
L342	Linde	Erika
L921	Müller	Markus
L012	Prinz	Johann
L452	Amberger	Anna

Das kartesische Produkt von AG × LEHRER ergibt:

AG × LEHRER							
AG_Nr	Bezeichnung	Termin	Mindest_Jahr-gangsstufe	Lehrer-nummer	Lehrer-nummer	LName	LVor-name
AG_01	Foto-AG	Mo. 7./8.	8	L342	L342	Linde	Erika
AG_02	Badminton-AG	Do. 9./10.	5	L342	L342	Linde	Erika
AG_03	DELF-AG	Di. 7./8.	9	L921	L342	Linde	Erika
AG_04	Orchester	Fr. 7./8.	8	L452	L342	Linde	Erika
AG_01	Foto-AG	Mo. 7./8.	8	L342	L921	Müller	Markus
AG_02	Badminton-AG	Do. 9./10.	5	L342	L921	Müller	Markus
AG_03	DELF-AG	Di. 7./8.	9	L921	L921	Müller	Markus
AG_04	Orchester	Fr. 7./8.	8	L452	L921	Müller	Markus
AG_01	Foto-AG	Mo. 7./8.	8	L342	L012	Prinz	Johann
AG_02	Badminton-AG	Do. 9./10.	5	L342	L012	Prinz	Johann
AG_03	DELF-AG	Di. 7./8.	9	L921	L012	Prinz	Johann
AG_04	Orchester	Fr. 7./8.	8	L452	L012	Prinz	Johann
AG_01	Foto-AG	Mo. 7./8.	8	L342	L452	Amberger	Anna
AG_02	Badminton-AG	Do. 9./10.	5	L342	L452	Amberger	Anna
AG_03	DELF-AG	Di. 7./8.	9	L921	L452	Amberger	Anna
AG_04	Orchester	Fr. 7./8.	8	L452	L452	Amberger	Anna

Die Tabelle AG × LEHRER enthält nun sämtliche Kombinationsmöglichkeiten zwischen den Datensätze der Relation AG und den Datensätzen der Relation Lehrer, im Beispiel also $4 \cdot 4 = 16$ Datensätze. Das bedeutet nicht, dass jede Kombination auch in der Realität eine sinnvolle Verknüpfung darstellt.

Der Join im natürlichen Verbund wählt nun die Datensätze aus, bei denen die Lehrernummer identisch ist. Zusätzlich wird eine Spalte Lehrernummer gelöscht.

AG >< LEHRER						
AG_Nr	Bezeichnung	Termin	Mindest_Jahr-gangsstufe	Lehrer-nummer	LName	LVorname
AG_01	Foto-AG	Mo. 7./8.	8	L342	Linde	Erika
AG_02	Badminton-AG	Do. 9./10.	5	L342	Linde	Erika
AG_03	DELF-AG	Di. 7./8.	9	L921	Müller	Markus
AG_04	Orchester	Fr. 7./8.	8	L452	Amberger	Anna

Man erhält also eine Tabelle, der man auf den ersten Blick die AG und den zugeordneten Namen des Lehrers entnehmen kann.

Der natürliche Verbund vereint Tabellen über gemeinsame Attribute, hier die gemeinsame Lehrernummer. Manchmal möchte man aber gerne vorgeben, über welchen Voraussetzungen der Verbund vorgenommen werden soll.

Der Join (Theta-Verbund) macht dies möglich. Hier kann man eine Bedingung C (condition) angeben, die den Stellenwert der gemeinsamen Attribute beim natürlichen Verbund einnimmt.

Anwendung relationaler Operatoren[21]

Zur Darstellung eines komplexeren Beispiels zur Relationenalgebra gehen wir von

Schüler

Schüler-Nr	Name	Vorname	Tutor	Geschlecht
123	Alberti	Hans	Müller	m
034	Glücklich	Gesine	Abel	w
321	Müser	Angelika	Abel	w
111	Weber	Wolfgang	Zange	m

Kurs

Kurs-Nr	Typ	Fach	Thema	Jahrgangsstufe
13	GK	Mathematik	Analysis 2	12/I
11	GK	Physik	Mechanik 1	11/I
03	GK	Informatik	Datenbanken	12/II
25	LK	Englisch	Short Stories	12/I
89	GK	Informatik	Compilerbau	13/II

Besucht

Schüler-Nr	Kurs-Nr	Fehlstunden	Punkte
123	03	00	12
123	25	03	07
321	89	00	14
111	03	21	03

[21] aus: (Burkert, Lächa, & Meyer, Version 1.000001, S. 10ff (Kap. 3))

Es soll die Frage beantwortet werden, welche Mädchen Informatikkurse besuchen und welche Punktzahlen sie dabei erreicht haben.

1. Bestimmung der Informatikkurse :

Informatikkurse = $\pi_{\text{Kurs-Nr}}(\sigma_{\text{Fach = Informatik}}(\text{Kurs}))$

Kurs-Nr
03
89

2. Bestimmung der Informatikschüler (alle Schülernummern der Informatikschüler):

Join mit der Besucht-Tabelle über das gemeinsame Schlüsselattribut Kurs-Nr liefert die Informatikschüler

Informatikschüler = Informatikkurse >< Besucht

Kurs-Nr	Fach	Schüler-Nr	Fehlstunden	Punkte
03	Informatik	123	00	12
03	Informatik	111	21	03
89	Informatik	321	00	14

3. Filtern der notwendigen Informationen:

Projektion auf die benötigten Attribute Schüler-Nr und Punkte:

InformatikschülerPunkte = $\pi_{\text{Schüler-Nr, Punkte}}$ (Informatikschüler)

Schüler-Nr	Punkte
123	12
111	03
321	14

4. Zuordnung der Schülernummern zu den Daten der Schüler:

Join mit Schüler-Tabelle über das gemeinsame Schlüsselattribut Schüler-Nr.

InformatikschülerPunkteName = InformatikschülerPunkt >< Schüler

Schüler-Nr	Punkte	Name	Vorname	Tutor	Geschlecht
123	12	Alberti	Hans	Müller	m
111	03	Weber	Wolfgang	Zange	m
321	14	Müser	Angelika	Abel	w

5. Selektion der Mädchen und Projektion auf die geforderten Angaben.

Ergebnis = $\pi_{\text{Punkte,Name,Vorname}}(\sigma_{\text{Geschlecht=w}}(\text{InformatikschülerPunkteName}))$

Punkte	Name	Vorname
14	Müser	Angelika

Aufgaben:

1. Gegeben seien drei Relationen mit den folgenden Tupeln:

Besucht

Gast	Bistro
Hans	Uno
Ede	Uno
Ede	Dos
Ede	Chico
Karl	Dos
Karl	Chico
Heini	Uno

Serviert

Bistro	Getränk
Uno	Wasser
Uno	Kaffee
Dos	Kaffee

Mag

Gast	Getränk
Hans	Wasser
Ede	Wasser
Ede	Kaffee
Karl	Kaffee

   a. Bilde Serviert x Mag.
   b. Bilde Serviert >< Mag. Welche Informationen beinhaltet diese Relation?
   c. Gib alle Bistros aus, die Getränke servieren, die Karl mag. Überprüfe deine Operationen in der Relationenalgebra anhand des Beispiels.
   d. Gib alle Gäste aus, die mindestens ein Bistro besuchen, welches auch das Getränk serviert, das sie mögen. Formuliere die Anfrage mit Operatoren der Relationenalgebra.

2. Gegeben sind folgende Relationen (# ist das Zeichen für Nummer):

   Lieferanten (L#, LName, Status, Stadt)
   Teile (T#, TName, Farbe, Gewicht, Stadt)
   Projekte (P#, PName, Stadt)
   Lieferungen (L#, T#, P#, Anzahl)

   Hierbei bedeutet Stadt einmal die Stadt, in der ein Lieferant sitzt, die Stadt, in der das entsprechende Teil hergestellt wird, bzw. die Stadt, in der ein Projekt stattfindet.

   Löse die folgenden Aufgaben durch Operationen aus der Relationenalgebra:
   a. Finde alle Lieferungen mit Anzahlen zwischen 300 und 750 und gib alle dazu in der Relation Lieferungen verzeichneten Informationen aus.
   b. Gib alle Städte aus, in denen Lieferanten sitzen.
   c. Gib alle vorkommenden Paarungen TName, Stadt aus.
   d. Finde alle schwarzen Teile. Gib ihre Nummer und ihren Namen aus.
   e. Finde alle Lieferanten, die in einer Einzellieferung mehr als 150 Teile geliefert haben. Gib ihren Namen aus.
   f. Finde alle Teile, die von Lieferanten in London geliefert wurden. Gib davon die Teilenummer (Teilenamen) aus.
   g. Finde alle Orte, in denen sowohl Projekte als auch Lieferanten beheimatet sind.
   h. Finde alle Projekte, die mindestens einen Lieferanten für das Projekt im gleichen Ort haben. Gib die Projektnummer aus.

i. Finde alle Teile, die der Lieferant Lux geliefert hat. Gib alle Teilinformationen von diesen Teilen aus.

3. Gegeben sind die folgenden Tabellen:

Tabelle1

A	B	C
4	2	8
2	2	1
6	7	3

Tabelle2

C	D
8	2
3	6

Tabelle3

B	E
5	3
4	4
5	4

Tabelle4

B	C	D
1	2	1
4	2	1
1	2	9

Führe folgende relationalen Operationen durch und stelle die Ergebnistabelle auf! Beschreibe die Aufgabenstellung mittels der behandelten Symbolik!

a. Selektion von Tabelle1 mit der Bedingung B=2
b. Projektion von Tabelle3 auf E
c. Join Tabelle1 und Tabelle2 nach dem gemeinsamen Attribut C
d. (Selektion von Tabelle 1 mit B>C) vereinigt mit (Selektion von Tabelle1 mit A<5)

4. Gegeben seien folgende Tabellen:

GK-Fach 1

Raum	Fach	Lehrer
137	Mathematik	Müller
221	Deutsch	Schmidt
104	Englisch	Lehmann

GK-Fach 2

Raum	Fach	Lehrer
127	Informatik	Müller
104	Englisch	Lehmann
123	Physik	Paulsen
018	Musik	Schmidt

Themen

Nr.	Thema	Klasse
001	Analysis	12/I
002	Klassik	13/1

a. GK-Fach 1 ∩ GK-Fach 2
b. GK-Fach 1 ∪ GK-Fach 2
c. GK-Fach 1 \ GK-Fach 2
d. Themen x GK-Fach 2

5. Ein Drogeriemarkt speichert die Informationen über seine angebotenen Artikel in der Tabelle SORTIMENT. Dabei steht „EP“ für Einkaufspreis und „VP“ für Verkaufspreis.

SORTIMENT						
ArtNr	Artikel	Kategorie	Hersteller	EP	VP	Bestand
1001	Seife	Hygiene	Wash Me	0,44 €	0,99 €	300
1002	Duschgel	Hygiene	Wash Me	0,79 €	1,29 €	250
1003	Shampoo	Hygiene	Wash Me	0,83 €	1,49 €	250
1120	Deo 4you	Parfum	KC Two	2,23 €	4,88 €	100
1121	Deo Only	Parfum	KC Two	2,23 €	4,88 €	150
2441	Toilettenpapier	Hygiene	DrEye	1,11 €	1,99 €	800
2447	Taschentücher	Hygiene	DrEye	1,23 €	2,19 €	1100
3894	Spülmittel	Haushalt	Scheuerfix	2,04 €	3,99 €	750
3991	Spülmittel	Haushalt	Dr. Sauber	2,45 €	4,49 €	300
3992	Reiniger	Haushalt	Dr. Sauber	2,71 €	4,69 €	300

a. Formuliere folgende Abfragen umgangssprachlich und gib die Ergebnistabelle an.

(1) $\pi_{Hersteller}$(Sortiment)
(2) $\pi_{Artikel,\ Hersteller,EP}$(Sortiment)
(3) $\sigma_{Bestand>500}$ (Sortiment)
(4) $\sigma_{Kategorie='Hygiene' \cap Bestand <= 250}$ (Sortiment)
(5) $\pi_{Artikel}(\sigma_{VP\text{-}EP >1{,}50\ € ' \cap Kategorie <> ,Parfum'}$ (Sortiment)
(6) $\pi_{Artikel,\ EP,\ Bestand}(\sigma_{Bestand*EP > 1000€}$ (Sortiment)

b. Welche Operationen sind nötig, um folgende Informationen zu erhalten? Gib jeweils auch die Ergebnistabelle an.
   (1) Welche Artikel verkauft der Drogeriemarkt?
   (2) Gesucht sind sämtliche Informationen über die Artikel des Herstellers „Wash Me“.
   (3) Von welchen Artikeln (Artikelnummer, Artikel und Bestand) sind weniger als 300 Stück auf Lager?
   (4) Gesucht sind die Nummern derjenigen Artikel, die im Laden mehr als vier Euro kosten.
   (5) Welche Hygieneartikel verkauft der Drogeriemarkt?
   (6) Bei welchen Artikeln (Artikelnummer, Artikel und Verkaufspreis) verdient der Drogeriemarkt pro verkauften Artikel mehr als 1,50 €?

Normalisierung

Bisher ist es möglich, dass die von uns erstellten Relationen in der späteren Datenbank dennoch zu ungewünschten Datenanomalien führen. Daher sollte man das aufgestellte Relationenmodell hinsichtlich der Normalisierungsregeln überprüfen und ggf. überarbeiten. Mit der Normalisierung soll vermieden werden, dass unerwünschte Anomalien beim Einfügen, Löschen oder Verändern in der Datenbank auftreten, die dann zur Inkonsistenz der Daten führen könnten. Ebenso soll eine Datenredundanz vermieden werden. Zusätzlich erreicht man durch die Normalisierung, dass man einen systematischen Entwurf der Datenbank erhält, der für Benutzer und Programmierer übersichtlicher ist.

Die Redundanz eines Merkmals

Ein Merkmal einer Tabelle ist redundant, wenn einzelne Werte des Merkmals innerhalb der Tabelle ohne Informationsverlust weggelassen werden können. Betrachten wir ein Beispiel:

Tabelle: Lehrerdaten

L-Nr	*L-Nachname*	*L-Vorname*	*Amtsbez*	*Besoldungsgruppe*
1	Bavaria	Eusebia	StRin	A13
2	Börse	Hanna	StRin	A13
3	Kalkulus	Carl-Friedrich	StD	A15
4	Spieking	Josef	OStR	A14

Tabelle 3-14: Redundante und anomalienträchtige Tabelle [22]

Das Merkmal „Besoldungsgruppe“ ist redundant, da mehrmals ein und dieselbe Besoldungsgruppe in der Tabelle vorkommt. Da die Besoldungsgruppe von der Amtsbezeichnung abhängig ist, würde es genügen,

[22] (Matzke, 2000, S. 44)(auch die folgenden Beispieltabellen sind hieraus entnommen)

wenn die Amtsbezeichnung mit der zugehörigen Besoldungsgruppe in einer eigenen Tabelle gespeichert würde. Wenn man diesen Sachverhalt für jeden Mitarbeiter abspeichert, sind die Daten redundant und es kann zu Problemen kommen. Wenn man z. B. eine neue Besoldungsgruppe, z. B. A12 einfügen möchte, so kann man dieses Besoldungsgruppe nicht ohne weiteres ergänzen, da man keine neue Tabellenzeile ergänzen kann, wenn man keine eindeutige Lehrernummer dieser Tabellenzeile zuordnen kann. Dies nennt man **Einfügeanomalie**. Zu einer **Löschanomalie** kann die obige Tabelle führen, wenn es nur noch einen OStR gibt und dieser nun die Schule verlässt. Löscht man Josef Spieking aus der Tabelle, gehen leider auch die Informationen über die Besoldungsgruppe des OStR verloren. Von einer **Änderungsanomalie** spricht man, wenn z. B. durch eine Reform die Besoldungsgruppen umstrukturiert würden und nun für alle StRin die Besoldungsgruppe auf A13.2 umgestellt werden müsste. Diese Änderung müsste bei jedem Kollegen einzeln durchgeführt werden, dabei könnte es passieren, dass manche Daten geändert werden und andere nicht.

Die drei Normalisierungsregeln

Bereits 1972 hat Codd drei Normalisierungsregeln aufgestellt, die von der 1. Normalform über die 2. Normalform zur 3. Normalform einer Relation führen. Verschiedene andere Informatiker haben später weitere Regeln und Normalformen aufgestellt, die aber in der Praxis keine große Rolle spielen und daher hier unbeachtet bleiben. Die drei Normalisierungsregeln bauen aufeinander auf, zunächst muss Regeln 1 erfüllt sein, da Regel 2 nur erfüllt sein kann, wenn Regel 1 bereits gilt.

Gegeben ist die Tabelle:

Lehrer (unnormalisiert)

L Nr	*L-Name*	*L-Vorname*	*Amtsbez*	*Klassen*
1	Bavaria	Eusebia	StRin	11WA, 11WB, 12TD, 12SC
3	Kalkulus	Carl-Friedrich	StD	11TC, 11TD, 12GB

Nun zu den Regeln:

1. **Eine Tabelle liegt in der ersten Normalform (1 NF) vor, wenn jeder Attributwert eine atomare, nicht weiter zerlegbare Dateneinheit ist.**

Eine Tabelle ist nicht in 1 NF, wenn Attribute mehrfach oder komplex in einer Spalte auftreten; d. h. 1 NF ist eine Strukturierungsvorschrift. Um eine Relation in die 1. NF zu bringen, muss man die nicht atomaren Attribute in verschiedene Zeilen oder mehrere Spalten oder einer eigene Tabelle auslagern.

Am Beispiel: In der Tabelle Lehrer ist das Merkmal „Klassen“ ein Mehrfachattribut. Dies ist nicht zulässig, so dass man in diesem Fall aus einer Zeile mehrere Tabellenzeilen erzeugen muss, so dass die Klasse nur noch einfach gespeichert wird. Man erhält folgende Tabelle:

Lehrer 1NF (in erster Normalform)

L Nr	*L-Name*	*L-Vorname*	*Amtsbez*	*Klasse*
1	Bavaria	Eusebia	StRin	11WA
1	Bavaria	Eusebia	StRin	11WB
1	Bavaria	Eusebia	StRin	12TD
1	Bavaria	Eusebia	StRin	12SC
3	Kalkulus	Carl-Friedrich	StD	11TC
3	Kalkulus	Carl-Friedrich	StD	11TD
3	Kalkulus	Carl-Friedrich	StD	12GB

Die Anwendung der ersten Normalisierungsregel führt in diesem Beispiel jedoch dazu, dass der bisherige Primärschlüssel LNr nicht mehr ausreicht, um ein Tupel (eine Zeile der Tabelle) eindeutig identifizieren zu können. Zudem kommt es paradoxerweise durch die Anwendung der Regel zu Redundanzen. Sowohl der Name als auch der Vorname und die Amtsbezeichnung werden mehrfach wiederholt. Doch die Anwendung der zweiten Normalisierungsregel wird die Probleme beseitigen, so dass wir hierauf zunächst keine Rücksicht nehmen müssen. Entscheidend ist, dass keine mehrfachbelegten Attribute mehr vorhanden sind.

Häufig taucht auch das Mehrfachattribut „Adresse“ auf, indem dann neben der Straße und der Hausnummer auch die PLZ und der Ort gespeichert werden. In diesem Fall müssen keine Zeilen sondern Spalten ergänzt werden.

2. **Eine Tabelle liegt in der zweiten Normalform (2 NF) vor, wenn sie in der ersten Normalform ist und jedes Nichtschlüsselattribut voll, funktional abhängig vom Primärschlüssel ist.**

Was bedeutet „voll funktional abhängig“?

In einer Relation ist das Merkmal bzw. die Merkmalskombination A vom Merkmal bzw. der Merkmalskombination B funktional abhängig, wenn zu jedem Wert von A genau ein Wert von B gehört.

Voll funktional abhängig erweitert diese Aussage insofern, dass A funktional abhängig ist von B, aber nicht bereits von einem Teil von B.

D. h. die zweite Normalisierungsregel fordert, dass zu jedem Wert eines Nichtschlüsselattributes A genau ein Wert des Schlüsselattributes gehört. Zusätzlich wird durch die volle funktionale Abhängigkeit gefordert, dass jedes Nichtschlüsselattribut nur durch die Gesamtheit des zusammengesetzten Schlüssels eindeutig bestimmt werden kann und nicht bereits ein Teil des Schlüssels zur Identifizierung ausreicht. Eine Tabelle ist also nicht in 2 NF, wenn Attribute von einem Teil des Schlüssels eindeutig identifiziert werden können.

Um eine 1NF-Tabelle in eine 2NF –Tabelle umzuformen müssen ggf. Teilschlüssel ausgelagert werden und zugehörige Informationen in eigenen Tabellen nach Sachgebieten gespeichert werden bzw. separate Entitätstypen mit eigenem Schlüssel gefunden werden. Beim Auslagern durch entsprechende Beziehungen muss darauf geachtet werden, dass keine Informationen verloren gehen.

Betrachten wir die Tabelle Lehrer 1NF. Der zusammengesetzte Schlüssel (L-Nr, Klasse) muss auf seine volle funktionale Abhängigkeit überprüft werden. Zunächst ist klar, dass L-Nr und Klasse gemeinsam eindeutig eine Zuordnung des Lehrervornamens, Lehrernachnamens und der Amtsbezeichnung gestatten. Es handelt sich also um einen möglichen Schlüssel. Allerdings sind der Lehrervorname, der Lehrernachname und die Amtsbezeichnung auch eindeutig identifiziert werden können, wenn man keine Klassenbezeichnung hat, daher sind diese drei Merkmale funktional abhängig von einem Teil des Schlüssels. Dies ist ein Widerspruch zur Definition der zweiten Normalform. Die gegebene Tabelle ist somit noch keine 2NF-Tabelle und muss entsprechend bearbeitet werden.

Die Tabelle muss zerlegt werden, indem man alle Merkmale, die von einem Teilschlüssel abhängig sind mit diesem Teilschlüssel in einer eigenständigen Tabelle abspeichert. Die restlichen Attribute und die weiteren Schlüsselattribute werden in einer zweiten Tabelle abgespeichert. Somit ergibt sich:

Lehrer (2NF)

L-Nr	*L-Name*	*L-Vorname*	*Amtsbez*
1	Bavaria	Eusebia	StRin
3	Kalkulus	Carl-Friedrich	StD

Unterricht (2NF)

L-Nr	***Klasse***
1	11WA
1	11WB
1	12TD
1	12SC
3	11TC
3	11TD
3	12GB

Die Tabelle Lehrer (2NF) enthält nun als Schlüsselattribut nur noch die L-Nr. Von dieser Lehrernummer sind alle Attribute voll funktional abhängig. Die Tabelle befindet sich auch in der 1NF, so dass nun beide Tabellen in der zweiten Normalform vorliegen.

3. Eine Tabelle liegt in der dritten Normalform (3 NF) vor, wenn sie sich in der 2 NF befindet und jedes Nichtschlüsselattribut nicht transitiv abhängig vom Primärschlüssel ist.

Was bedeutet „transitiv abhängig“?

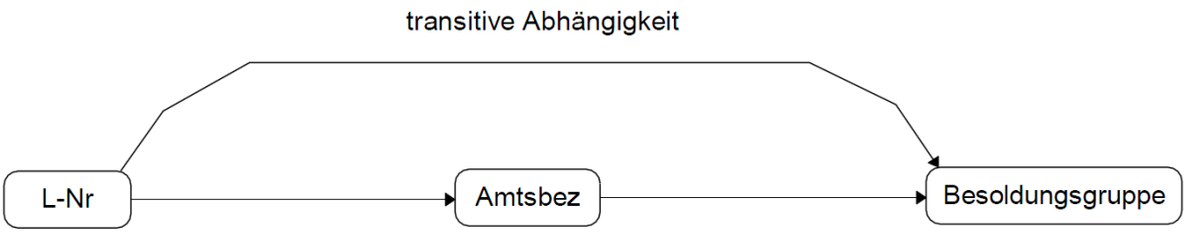
Seien A, B und C Attribute bzw. Attributkombinationen einer Tabelle, dann heißt C transitiv abhängig von A, wenn gilt: B ist funktional abhängig von A und C ist funktional abhängig von B. C ist also mittelbar über B von A abhängig.

Betrachtet man noch einmal die Tabelle „Lehrerdaten“, so kann man leicht nachweisen, dass diese Tabelle in der 2 NF gegeben ist, da der Primärschlüssel

Lehrerdaten (2NF)

L-Nr	*L-Nachname*	*L-Vorname*	*Amtsbez*	*Besoldungsgruppe*
1	Bavaria	Eusebia	StRin	A13
2	Börse	Hanna	StRin	A13
3	Kalkulus	Carl-Friedrich	StD	A15
4	Spieking	Josef	OStR	A14

Das Attribut Besoldungsgruppe ist von dem Attribut Amtsbezeichnung funktional abhängig. Das Attribut Amtsbezeichnung ist von der Lehrernummer funktional abhängig, somit ist die Besoldungsgruppe transitiv abhängig von der Lehrernummer.



Man kann also sagen: Eine Tabelle ist nicht in der 3 NF, wenn Attribute von anderen Nicht-Schlüsselattributen identifiziert werden.

Um eine 2NF-Tabelle in eine 3NF-Tabelle umzuformen, muss man die „transitiv abhängigen" Attribute in eigene Tabellen auslagern, d. h. Nichtschlüsselattribute, die von anderen Nichtschlüsselattributen abhängig sind, werden mit diesen als Schlüsselattribut in eine eigenständige Tabelle ausgelagert.

Lehrerdaten (3NF)

L-Nr	*L-Nachname*	*L-Vorname*	*Amtsbez*
1	Bavaria	Eusebia	StRin
2	Börse	Hanna	StRin
3	Kalkulus	Carl-Friedrich	StD
4	Spieking	Josef	OStR

Amtlich (3NF)

Amtsbez k	*Amtsbez L*	*Bes-gruppe*
StRin	Studienrätin	A13
StR	Studienrat	A13
OStRin	...	A14
OStR	...	A14
...	...	...

Aufgaben

1. Der Zoo
   a. Überführe das ER-Diagramm (nächste Seite) in das zugehörige relationale Model. Unterstreiche dabei die Primärschlüssel und kennzeichne auch sämtliche Fremdschlüssel.

   b. Gib für die Tabellen Tier und Mitarbeiter je drei passende Datensätze an.

   c. Überführe die Tabellen mithilfe der Normalisierungsregeln in die 3NF.

   d. Welche Operationen sind nötig, um folgende Informationen zu erhalten (Relationenalgebra)?

      (1) Welche Tiere kann man in dem Zoo betrachten?

      (2) Für welche Tiere ist der Tierpfleger mit der Mitarbeiternummer 24 zuständig?

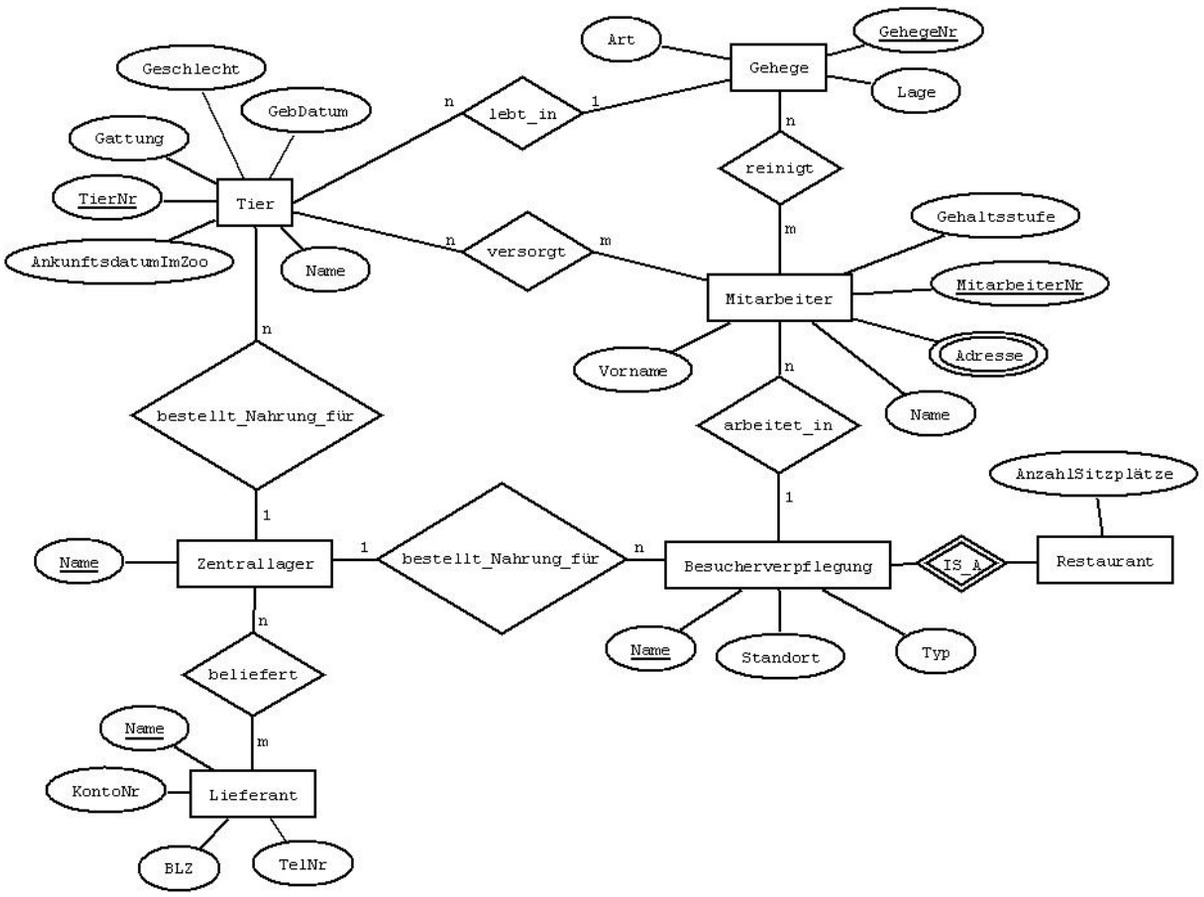
      (3) Welche Mitarbeiter arbeiten im Restaurant „Zum Goldfisch"?

      (4) Welche Tiere (Name und Gattung) leben in den Gehegen Nr. 4 bis 10?

      (5) Welche Lieferanten (Name und Telefonnummer) beliefern das Zentrallager „Am Eingang"?

(6) Es wird eine Adressliste Vorname, Nachname, Adresse) aller Mitarbeiter benötigt.

(7) Welches Restaurant (Name, Standort) haben mehr als 40 Sitzplätze?



2. „Ein Hochschulberater berät Studenten, die alle im Wohnheim der Universität leben und alle das gleiche Hauptfach studieren. Aus Besprechungen mit Studenten will der Berater eine kleine Datenbank zur Unterstützung der Beratung entwickeln. Er legt folgende Attribute und Regeln fest:

SNr	Studentennummer, ganze Zahl , eindeutiger Schlüssel für Studenten
SName	Name des Studenten, nicht eindeutig
ZNr	Zimmernummer ganzzahlig; jeder Student hat ein Zimmer, das aber von mehreren Studenten bewohnt werden kann
TNr	Telefonnummer des Studenten; sie ist für alle Zimmerbewohner gleich
Kurs	Identifikationsnummer der Kurse, die ein Student belegt oder bereits abgeschlossen hat
Semest	Semester, in dem ein Kurs abgeschlossen wurde; ein Student darf den gleichen Kurs in einem späteren Semester wiederholen
Note	Note eines abgeschlossenen Kurses

Der Berater möchte folgende Daten in der Datenbank speichern (Auszug):

SNr	SName	ZNr	TNr	Kurs	Semest	Note
3215	Jonas Mike	120DH	2136	MAT122	W88	1,4
				PHY120	S88	2,5
				WIW330	W89	3,1
				MAT122	S89	1,2
3456	Schmidt Klaus	237VH	3127	MAT122	W87	3,2
				MAT130	S87	2,9
4576	Neider Paul	120DH	2136	PHY230	W88	2,8
				MAT120	S88	2,1

„

(Burkert, Lächa, & Meyer, Version 1.000001, S. 18, (Kap. 3))

a. Erstelle ein Diagramm zum ER-Modell für diese Daten.
b. Mache aus der obigen Tabelle eine gültige Relation in der 1. Normalform. Welche Anomalien können dabei auftreten?
c. Erstelle anhand des ER-Modells und der Umsetzungsregeln das relationale Modell. Begründe, inwieweit die Tabellen die 3. Normalform erfüllen.
d. Gib die relationalen Operationen an, um Kurslisten auszugeben.

3. Du importierst Daten aus einer Excel-Tabelle und erhältst die folgende Struktur. Normalisiere sie bis zur 3. Normalform. Dokumentiere dabei ausführlich deine Schritte mit der entsprechenden Begründung (Definition der Normalformen und ihre Folgerungen).

PersNr	Nachname	Vorname	Abteilung	Projekt	Stunden
1	Lorenz	Christian	Einkauf	Verkaufsanalyse	198, 201
2	Baumann	Peter	Verkauf	Optimierung	120, 189, 43
3	Petersen	Anna	Personal	Weiterbildung	120

4. Ein Betrieb mit Außendienstmitarbeitern erfasst ihre Reisekosten in einer Tabelle, die wie folgt aufgebaut ist.

Reisekosten									
Rechnungs-nummer	Datum	Name	Vorname	Straße	PLZ	Ort	Kostenart	Anzahl	Einzelvergütung

Normalisiere sie und dokumentiere ausführlich deine jeweiligen Schritte mit Verweis auf die Fachbegriffe. Überlege dir am Ende ferner, welche Verbindungen zwischen welchen Tabellen mit entsprechender Kardinalität auftreten könnten.

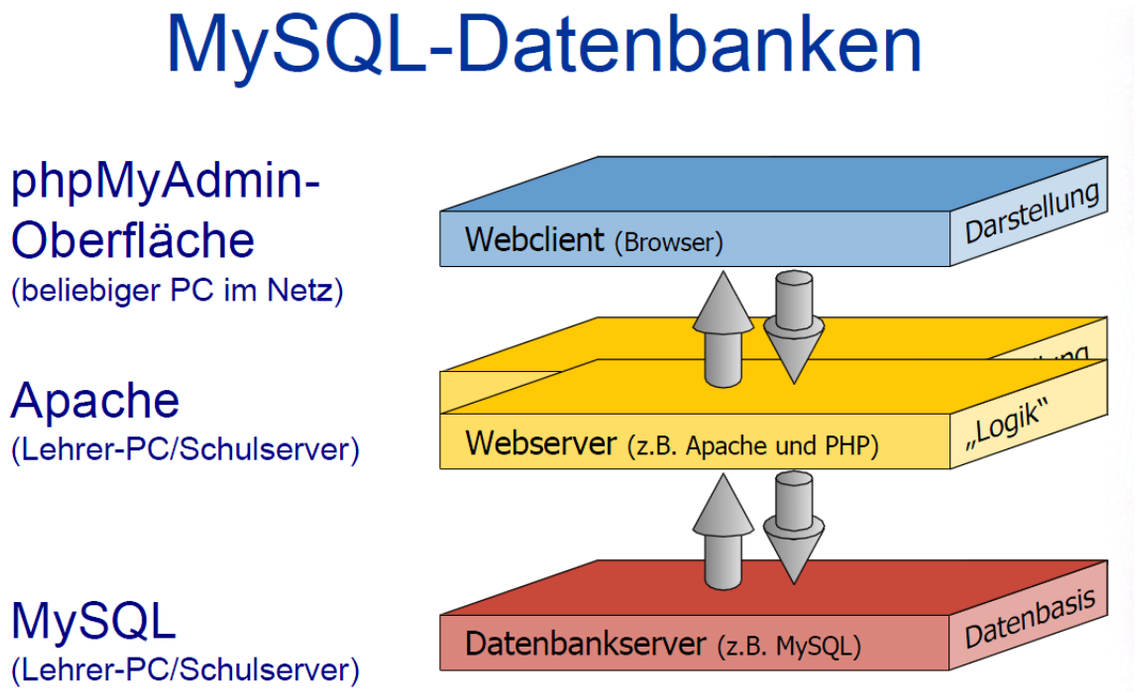
Die Umsetzung des Modells mit XAMPP

XAMPP ist ein Programmpaket, welches u. a. Apache, MySQL und PHP beinhaltet. Mit phpMyAdmin steht zudem ein DBMS zur Verfügung, mit welchem sehr komfortabel MySQL-Datenbanken bearbeitet werden können. Ein großer Vorteil liegt in der einfachen Installation und Verwaltung dieses Programms im Unterricht und zu Hause.

- X: Das Programmpaket ist mit verschiedene Betriebssysteme verwendbar.
- A: Apache Webserver
- M: MySQL-Datenbank
- P: Perl Skriptsprache
- P: PHP Skriptsprache

Wie man schnell erkennen kann, bietet das Programmpaket eine Vielzahl an Verwendungsmöglichkeiten, die wir gar nicht alle nutzen wollen. Neben XAMPP gibt es auch ein Programmpaket XAMPP Lite, welches schon alle von uns benötigten Programmpakete enthält.

Allgemeiner Aufbau einer MySQL-Datenbank in XAMPP:



(Heusel)

XAMPP ist kostenlos und kann von mehreren Benutzern zeitgleich verwendet werden. Der Installationsaufwand ist sehr gering, da das Programm nur entpackt und gestartet werden muss. Deinstalliert wird das Ganze, indem der entsprechende Ordner einfach gelöscht wird. Es handelt sich um ein modernes, praxisnahes Konzept.

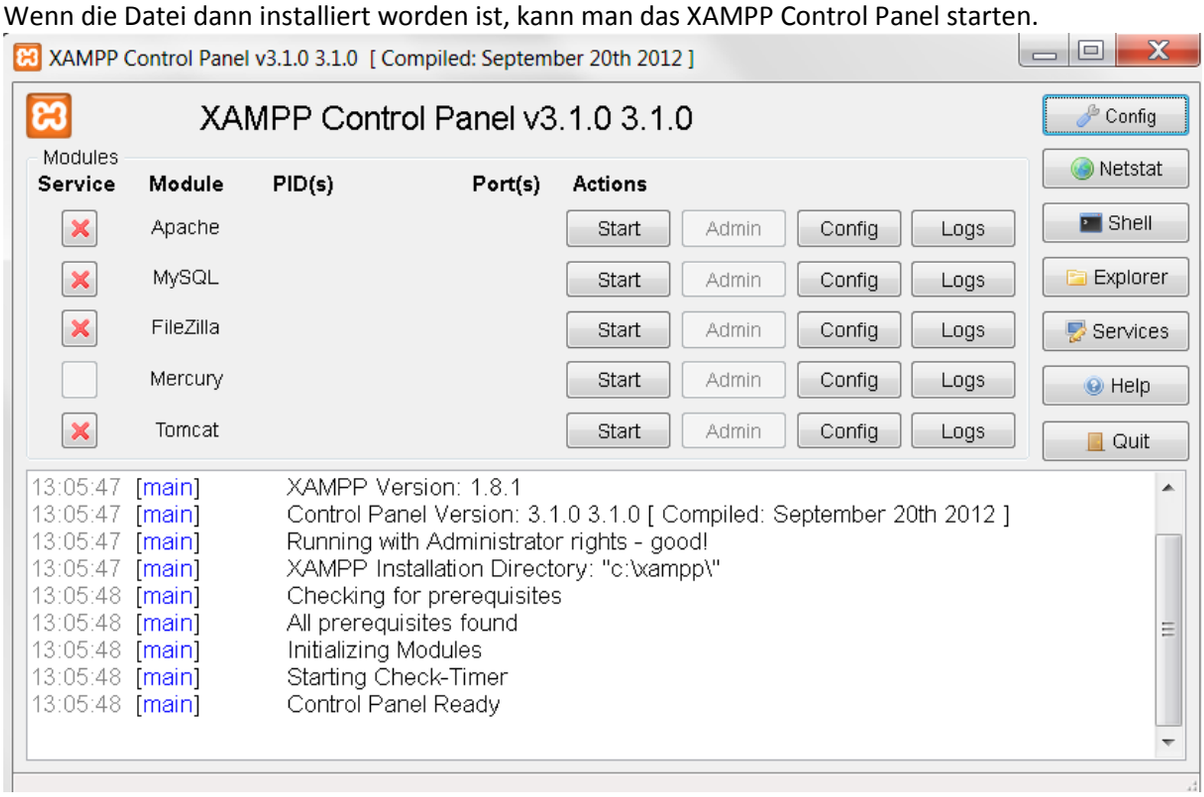
„Installation“ von XAMPP

Auf dieser Webseite http://www.apachefriends.org/de/xampp-windows.html#631 findet man (Stand: 02.05.2013) neben der normalen XAMPP-Version auch XAMPP Portable Lite, welches für unsere Zwecke absolut ausreichend ist und ggf. auch von einem Stick zu starten ist.

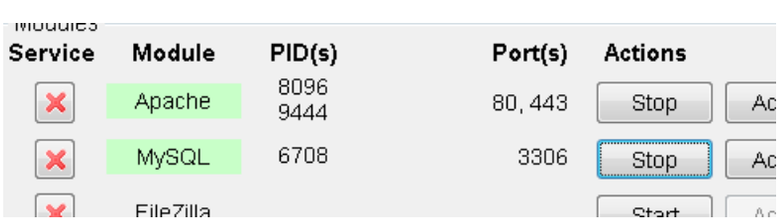
Hier kann nun jeder die entsprechende kostenlose Datei herunterladen. Weitere Hilfen sind (falls, die jemand benötigt) auf der gleichen Seite angegeben. Mit dem Entpacken dieser Datei wird in einem individuell festgelegten Installationslaufwerk ein Ordner "xampp" angelegt, in den alle notwendigen Daten kopiert werden. Das Installationsverzeichnis sollte sich auf der Root-Ebene eines beliebigen Laufwerks (evtl. Stick) befinden, wobei dies auch ein Netzlaufwerk, beispielsweise H:\XAMPP sein kann. Damit ist ein kompletter WAMP-Server mit Apache-Webserver und php-Unterstützung, MySQL-Datenbank und phpMyAdmin als webbasiertes Frontend zur Datenbankadministration installiert. "phpMyAdmin" ist ein Managementsystem für MySQL-Datenbanken. Es hilft, typische Aufgaben wie das Anlegen von Datenbanken, das Bearbeiten von Strukturen, die Benutzerverwaltung etc. zu vereinfachen.

Eine Datenbank mit phpMyAdmin erstellen

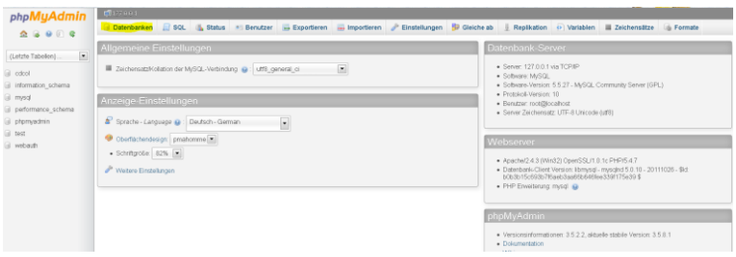
Wenn die Datei dann installiert worden ist, kann man das XAMPP Control Panel starten.



Nun muss Apache und anschließend MySQL gestartet werden. Nach getaner Arbeit, müssen die beiden Funktionen wieder in umgekehrter Reihenfolge gestoppt werden.

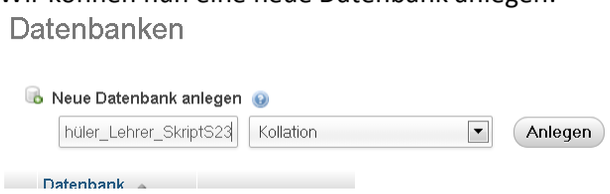


Nun kann es losgehen. Entweder du klickst auf den Button „Admin“ in der Zeile Apache oder du gibst in deinem Browser localhost im Adressfeld ein. Dann öffnet sich folgendes Fenster:



Klicke hier nun auf phpMyAdmin (siehe Pfeil) und anschließend auf „Datenbanken“, um eine Datenbank erstellen zu können.

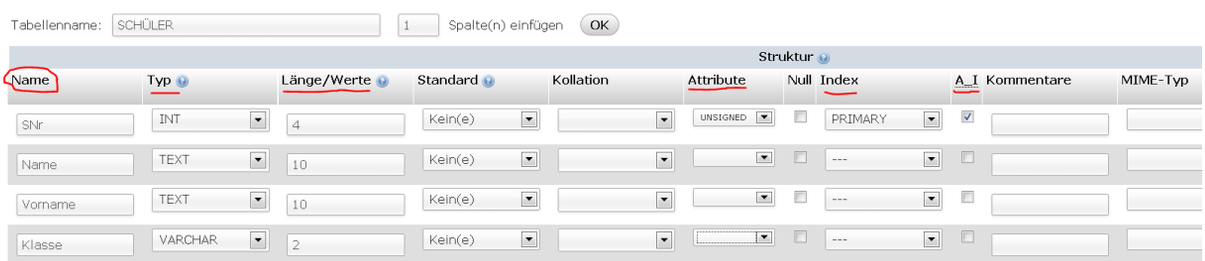
Wir können nun eine neue Datenbank anlegen:



Als Beispiel verwenden wir das ER-Modell von S. 23, welches wir dort bereits in ein Relationenmodell abgeleitet haben. Um neue Tabellen erzeugen zu können, musst du zunächst in der linken Spalte auf die entsprechende Datenbank klicken. Nun kannst du eine Tabelle anlegen.

Tabellen anlegen

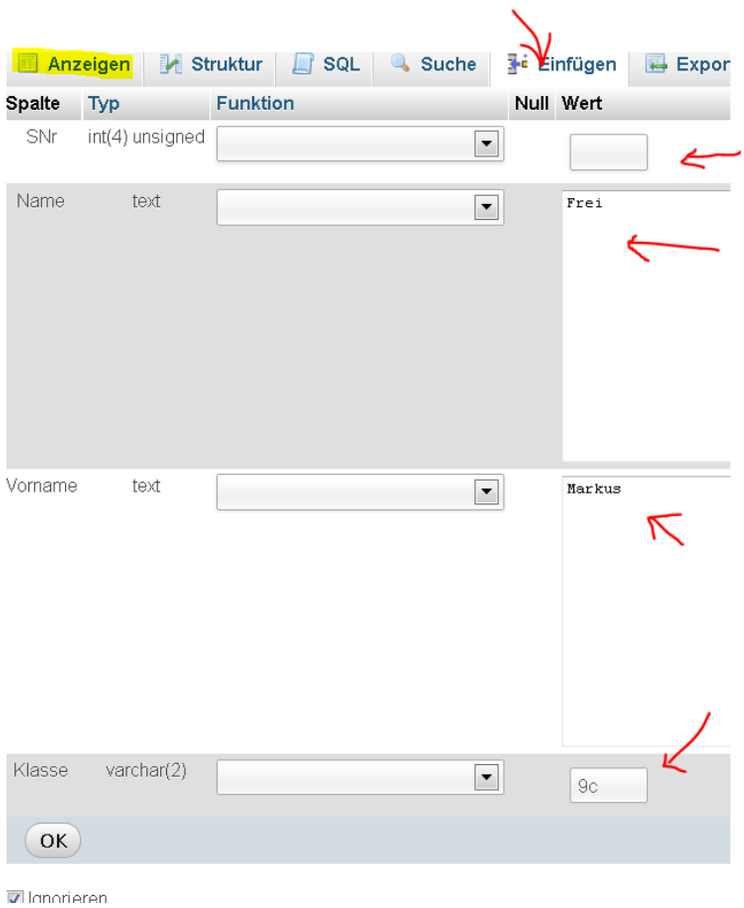
→ Datenbank auswählen
→ neue Tabelle: Name der Tabelle (hier: SCHÜLER) und Spaltenzahl (hier: 4) eingeben
→ Datenfelder mit Name(Schema der Tabelle) und Typ angeben. Jedem Spaltennamen muss ein Datentyp zugeordnet werden. Grundsätzlich kann zwischen numerischen Typen (INT, FLOAT, usw.) und Zeichenketten (CHAR, TEXT, usw.) und dem Datum (DATE, TIME, YEAR, usw.) unterschieden werden. Die Länge darf nicht zu kurz definiert werden.
→ Wenn nur positive Werte zulässig sind, kann man das Attribut ‘unsigned’ hinzufügen.
→ Die Primärschlüssel warden mit primary gekennzeichnet.
→ Wird unter A_I ein Häkchen in dem Feld gesetzt, so werden die Felder automatisch gefüllt, in dem von Null beginnend hochgezählt wird. A_I steht für auto-increment und darf nur maximal einmal je Tabelle für eine INT-Spalte verwendet werden.
→ (fast) alles kann nachträglich verändert werden

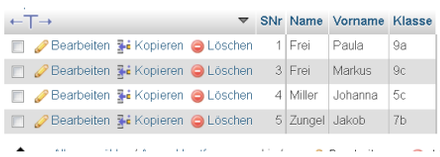


Datensätze anlegen und anzeigen

Ist die Tabelle soweit vorbereitet, können einzelne Datensätze eingefügt werden. Hierzu klickt man zunächst in der linken Spalte des Bildschirms auf die entsprechende Datei und dann in der Aktionsleiste auf ‚Einfügen'.

Da wir angegeben haben, dass die Schülernummer automatisch gefüllt werden soll, lassen wir diese frei und ergänzen nur die Namen, Vornamen und die Klasse der entsprechenden Schüler. Durch die Automatische Nummerierung entsprechen die Schülernummern jetzt nur nicht der Vorlage auf Seite 23. Die Werte werden also mit den gewünschten Daten gefüttert. Wenn man alle Schüler eingegeben hat, kann man die eingetragenen Datensätze sich auflisten lassen, indem man in der Aktionsleiste auf ‚Anzeigen' klickt.

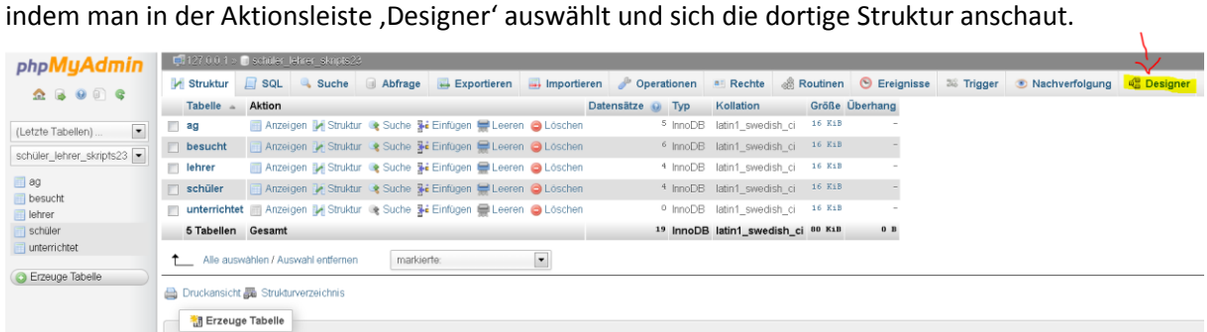




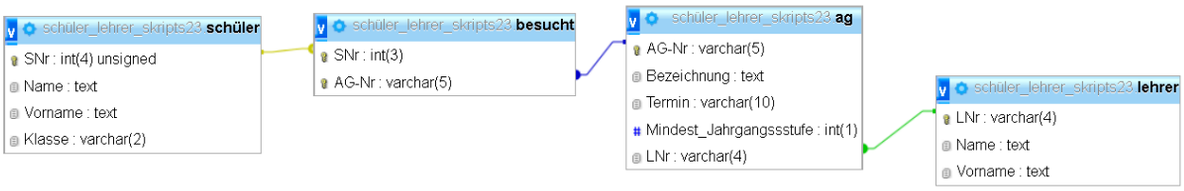
Da ich bei der Eingabe zunächst nach Paula Frei einen falschen Schüler eingegeben hatte, ist die SNr 2 gelöscht worden. Diese Zahlen werden nachträglich nicht aufgefüllt.

Designer – ER-Diagramm

Wenn man die Tabellen erstellt hat, kann man sich die erstellten Strukturen noch einmal anzeigen lassen, indem man in der Aktionsleiste ‚Designer' auswählt und sich die dortige Struktur anschaut.

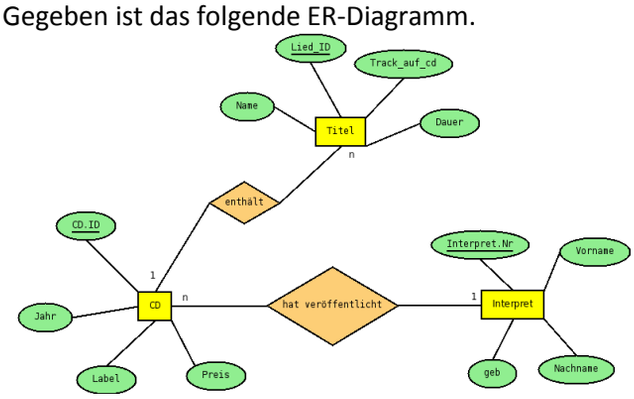


Das Ergebnis erinnert an ein Klassendiagramm, welches nun gut mit dem ER-Diagramm verglichen werden kann.



Aufgaben:

1. Gegeben ist das folgende ER-Diagramm.



   a. Entwickle ein relationales Modell.
   b. Realisiere die CD-Datenbank in XAMPP. Ergänze pro Tabelle drei bis fünf Datensätze.

2. Auf Seite 86 findest du die Aufgabe „Der Zoo“. Erstelle diese Datenbank in XAMPP. Es genügt, wenn du pro Tabelle drei Datensätze erfindest.

Die Datenabfragesprache SQL (Structured Query Language)

Was ist SQL?

Die Sprache SQL (Structured Query Language) macht es für den Benutzer möglich, relationale (tabellenorientierte) Datenbanken zu erstellen und die Inhalte dieser Datenbanken zu verwalten, bzw. zu verändern. Sie wurde von IBM entwickelt. SQL unterscheidet normalerweise nicht zwischen Groß- und Kleinschreibung, zur besseren Übersicht werden wir jedoch festgelegte Befehle in Großbuchstaben schreiben.

Die Sprache SQL wird verwendet für:

- um Daten zu definieren: Data Definition Language (DDL)
- um Daten abzufragen: Data Query Language (DQL)
- um Daten zu manipulieren: Data Manipulation Language (DML)
- und um Daten zu kontrollieren Data Control Language (DCL)

Mit Hilfe der nächsten Tabelle wirst du intuitiv an die Datenabfragesprache SQL herangeführt. In der linken Tabellenspalte wird jeweils eine SQL-Abfrage angegeben. Deine Aufgabe ist es, das Ergebnis der Abfrage in der rechten Tabellenspalte zunächst anzugeben. Überlege auch, in welchem Zusammenhang der SQL-Befehl mit der Relationenalgebra steht. Zugrunde gelegt wird die folgende Schülerdatenbank.

Tabelle: Schüler

SNR	Name	Vorname	GebDat	Geschlec	Klasse	Religion
1	Maier	Paul	14.01.1995	m	8a	rk
2	Müller	Meike	24.02.1996	w	8b	
3	Paschulke	Hermann	16.06.1994	m	9b	rk
4	Lustig	Petra	17.06.1993	w	10a	ev
5	Adams	Michael	18.06.1992	m	10d	rk
6	Baumgärtner	Sabine	19.06.1997	w	7c	
7	Kettwig	Maria	20.06.1997	w	7c	
8	Müller	Kevin	21.06.1996	m	8b	rk
9	Hartmann	Marco	22.06.1998	m	5c	ev
10	Wagner	Stefanie	20.06.1997	w	6b	ev
11	Jennicke	Torben	28.02.1998	m	5a	ev
12	Markwart	Sabine	04.06.1997	w	5b	ev

Tabelle: Ausleihe

AuslNr	Buch	Schüler	AuslDatum	RückDatum
1	9	3	13.11.10	13.12.10
2	5	4	21.11.10	21.12.10
3	2	5	23.11.10	23.12.10
4	3	11	1.12.10	1.1.11
5	11	11	14.12.10	14.1.11
6	5	7	18.12.10	18.1.11
7	15	3	10.1.11	10.2.11
8	5	10	10.1.11	10.2.11
9	2	5	21.1.11	21.2.11
10	12	5	17.2.11	18.2.11

Tabelle: Bücher

InvNr	Titel	Autor	Verlag	Jahr	ISBN	Sprache
1	Taschenbuch der Algorithmen	Vöcking, Alt, Dietzfelbinger	Springer	2008	3540763937	deutsch
2	Duden Basiswissen Schule Politik	Rytlewski, Wuttke	Bibliographisches Institut	2002	3411045906	deutsch
3	Duden Basiswissen Schule Geschichte	keine Angabe	Bibliographisches Institut	2004	3411715812	deutsch
4	Warm Up Computer	Thomas Alker	Herdt-Verlag	1998	3938178000	deutsch
5	Kompendium der Informationstechnik	Sascha Kersken	Galileo Press	2001	3898423557	deutsch
6	Warm Up Internet	Susanne Weber	Herdt-Verlag	2001	3938178019	englisch
7	Mond über Marrakesch	Waldtraut Lewin	Ravensburger	1985	3473352446	deutsch
8	Die Hexe von Aggunda	Olov Svedelid	DTV	1993	3423708948	deutsch
9	Die Outsider	Susan E. Hinton	DTV	2000	3423781696	deutsch
10	Krabat	Otfried Preußler	Thienemann	1990	3522144104	deutsch
11	The Hobbit or There and Back Again	John Roland Reuel Tolkien	DTV	1980	3423071516	englisch
12	Rechnen mit Brüchen	Anja Wolf	Ravensburger	1978	3473413666	deutsch
13	Straßenkinder	Reiner Engelmann	Elefanten Press	1998	3570146251	deutsch
14	Lesebuch zur Weltgeschichte	Manfred Mai	Hanser	2008	3446204474	deutsch
15	Sternkinder	Clara Asscher-Pinkhof	Oetinger	2005	3789106968	deutsch
16	Eine kurze Weltgeschichte für junge Leser	Ernst H. Gombrich	Dumont	2009	3832174923	deutsch

Tabelle: Mahnung

MahnNr	Ausleihe	Datum	Betrag
1	3	27.12.10	1.50
2	4	4.1.11	1.50
3	6	21.1.11	1.50
4	7	13.2.11	1.50
5	8	13.2.11	1.50

Tabelle: SchülerAdresse

SNR	Straße	PLZ	Ort
1	Ahornstraße 15	56068	Koblenz
2	Buchenweg 23	56127	Urbar
3	Victoriastraße 3	56068	Koblenz
4	Tannenweg 35	56422	Vallendar
5	Lindenstrasse 2	56244	Mendig
6	Fichtenweg 67	56324	Dieblich
7	Finkenweg 4	56068	Koblenz
8	Amselgasse 3	56249	Mayen
9	Drosselgasse 4	56400	Andernach
10	Meisenweg 8	56200	Neuwied
11	Adlerstrasse 7	56068	Koblenz
12	Pfuhlgasse 4	56068	Koblenz

Hier nun die SQL-Abfragen:

Nr.	SQL-Abfrage	Ergebnis der Abfrage
1	SELECT Name, Vorname FROM Schüler;	
2	SELECT * FROM Schüler;	
3	SELECT Titel FROM Buch WHERE Sprache = 'englisch';	
4	SELECT DISTINCT Verlag FROM Buch;	
5	SELECT Name FROM Schüler WHERE Religion IS NULL;	
6	SELECT Titel, Jahr FROM Buch WHERE Jahr < 1990;	
7	SELECT Titel, Jahr FROM Buch WHERE Jahr < 1990 AND Sprache = 'englisch';	
8	SELECT Name FROM Schüler WHERE Geschlecht <> 'm';	
9	SELECT DISTINCT Ort FROM SchülerAdresse ORDER BY Ort;	
10	SELECT DISTINCT Ort FROM SchülerAdresse ORDER BY Ort DESC;	
11	SELECT Titel, Jahr FROM Buch ORDER BY Jahr DESC, Titel	
12	SELECT Titel FROM Buch WHERE Verlag IN ('DTV', 'Springer', 'Hanser');	

Erzeuge nun die Schülerdatenbank in XAMPP und führe dort die SQL-Abfragen durch. Korrigiere, falls nötig, deine Abfragen.
Und nun umgekehrt! Gib jeweils die SQL-Abfrage an und teste sie mit der Schülerdatenbank.

Nr.	SQL-Abfrage	Ergebnis der Abfrage
1		Es werden alle Mahnungen (MahnNr und Betrag) absteigend sortiert nach dem Mahndatum ausgegeben.
2		Es werden die Titel der Bücher ausgegeben, die zwischen 1990 und 2005 (einschließlich dieser Jahre) erschienen sind.
3		Es sollen alle Datensätze der Tabelle „Ausleihe“ ausgegeben werden.
4		Es soll die Postleitzahl von Koblenz ausgegeben werden.
5		Es sollen alle Bücher des DTV-Verlags ausgegeben werden, allerdings primär aufsteigend sortiert nach der Sprache und sekundär absteigend sortiert nach dem Erscheinungsjahr.
6		Es sollen Titel und Verlag von jenen Büchern ausgegeben werden, die nicht vom DTV-Verlag aufgelegt wurden.
7		Es sollen Name und Vorname von jenen Schülerinnen angezeigt werden, die entweder den Vornamen „Sabine“, „Stefanie“ oder „Maria“ haben.
8		Es sollen alle deutschsprachigen Bücher ausgegeben werden, die 2001 oder 2008 erschienen sind.
9		Von der Tabelle „SchülerAdresse“ sollen alle Datensätze absteigend sortiert nach dem Straßennamen ausgegeben werden, deren PLZ mit den Ziffern 562 beginnen.
10		Welche Schülerinnen, gehören keiner Religion an und besuchen die 7c?

Bei der Arbeit mit XAMPP sind dir bereits noch weitere SQL-Befehle begegnet. Vermutlich hast du schon folgenden Befehl gesehen: CREATE TABLE. Mit dem CREATE TABLE-Befehl kannst du Tabellen anlegen. Folgende Dinge müssen dabei näher bestimmt werden:

- Tabellenname
- Spalte(n) (incl. Name und Datentyp)

Allgemein sieht der CREATE TABLE Befehl so aus: CREATE TABLE Name (Spalte1 Datentyp, Spalte2 Datentyp, …)

Wichtige Datentypen:

int	(ganze Zahlen)
float	(Dezimalzahlen)
string	(Zeichenkette beliebiger Länge)
varchar(n)	(Zeichenkette mit maximaler Länge n)
date	(Datumswert)
currency	(Geldbetrag)

Wichtig: Tabellen- und Spaltennamen dürfen keine Sonder- oder Leerzeichen enthalten.

Spaltenbedingungen: NOT NULL

Dieser Befehl legt fest, dass diese Spalte unter keinen Umständen leer sein darf, d.h. bei der Eingabe eines Datensatzes muss diese Angabe gemacht werden.
Bsp.: … Name string NOT NULL

PRIMARY KEY

Ein Primärschlüssel (meist eine Zahl) kann einen Datensatz eindeutig identifizieren. Dieses Attribut muss dann bei allen Datensätzen unterschiedlich sein.
Beispiel: Gewisse Namen können in einer Datenbank mehrmals auftauchen (mehrere Personen haben den gleichen Namen). Die Personalausweisnummer ist aber bei jedem verschieden und darf in der Datenbank nicht mehrmals erscheinen. Deshalb wird sie als Primärschlüssel definiert.
Bsp.: … Name string PRIMARY KEY

Eine Tabelle löschen - DROP TABLE:

DROP TABLE Name

Eine Tabelle ändern - ALTER TABLE:

Mit dem ALTER TABLE Befehl kannst du deine Tabelle ändern, wie z.B. Spalten hinzufügen bzw. löschen.

Hinzufügen von Spalten - ADD

Wenn du eine Spalte deiner Tabelle hinzufügen möchtest musst du natürlich wieder den Spaltennamen und den Datentyp angeben.
Aufbau: ALTER TABLE Name ADD Spalte1 Datentyp, Spalte2 Datentyp …

Löschen von Spalten - DROP

Beim Löschen von Spalten muss nur der Spaltenname und logischerweise nicht der Datentyp angegeben werden.
Aufbau: ALTER TABLE Name DROP Spalte1, Spalte2 …

Aufgaben:

1. Erstelle eine Tabelle Mensch mit den Spalten Name, Vorname, Geburtstag, Groesse, Personalausweisnummer.
2. Erstelle eine neue Tabelle Mensch2 mit den gleichen Spalten wie oben und definiere die Personalausweisnummer als Primärschlüssel und den Vornamen als NOT NULL.
3. Füge der oben erstellten Tabelle Mensch2 eine Spalte namens Hobby hinzu.
4. Lösche die Spalte Hobby wieder.

Übersicht zu SQL-Klauseln

Klausel	Beschreibung
Die FROM - Klausel	Hinter FROM steht der **Name der Tabelle**. Entstammen die Felder verschiedenen Tabellen, so muss jeweils der Tabellenname durch einen Punkt getrennt vorangesetzt werden (z.B.: Kunden.Ortnr).
Die WHERE-Klausel	Damit wird bestimmt, **welche Datensätze (Zeilen einer Tabelle**) ausgewählt werden sollen. **Bedingungsprüfung!** Enthält die Bedingungsprüfung mehrere Bedingungen, werden diese mit logischen Operatoren AND, OR und NOT verknüpft: **AND** ist dann wahr, wenn beide Bedingungen/Kriterien erfüllt sind Beispiel: Alle weiblichen Kunden aus Filderstadt (Ortnr=29740). Geschl = "w" AND Ortnr = 29740 **OR** ist dann wahr, wenn mindestens eine der beiden Bedingungen/Kriterien erfüllt ist. Beispiel: Alle Kunden aus Blaubeuren (Ortnr 39815) und Laichingen (Ortnr 39816). OrtNr = 39815 OR Ortnr = 39816 **NOT** negiert einen Ausdruck. Beispiel: Alle Kunden, die nicht aus Stuttgart kommen. NOT Ort = "Stuttgart"
Die ORDER BY-Klausel	Daten werden nach einem oder mehreren Feldnamen sortiert ausgegeben. Die vorgegebene **Sortierreihenfolge** ist aufsteigend ASC(ending). Soll absteigend sortiert werden, muss DESC(ending) eingegeben werden (⋯ORDER BY wert DESC).
Die GROUP BY-Klausel	Sie dient dazu, die Zeilen einer Tabelle nach bestimmten Feldern zu gruppieren.

MySQL verwendet als Platzhalter für beliebige Zeichen innerhalb einer Zeichenkette das "%"-Symbol.
Die **Grundstruktur** dieser Sprache sieht folgendermaßen aus:

SELECT Merkmale
FROM Tabellen
WHERE Selektionsprädikat

Vergleicht man SQL mit der Relationenalgebra, so entspricht die SELECT-Klausel der Projektion, indem sie eine Liste von Merkmalen angibt. In der FROM-Klausel werden alle benötigten Tabellen aufgeführt.

Übersicht: Vergleichs-Operatoren / Arithmetische Operatoren

SQL kennt die üblichen Vergleichsoperatoren:

=	gleich	<	kleiner
<>	ungleich	>=	größer gleich
>	größer	<=	kleiner gleich

BETWEEN .Wert1..AND..Wert2..	Vergleichswert liegt zwischen Wert1 und Wert2
IN Werteliste	Vergleichswert ist in der angegebenen Werteliste
Like Zeichenfolge	Vergleichszeichen entsprechen der Zeichenfolge
Is Null Feld	Vergleichsfeld hat einen NULL-Wert

Vergleichsoperatoren können verknüpft werden mit den Operatoren AND, OR und NOT.

Übersicht Aggregatsfunktionen (Gruppierungsfunktionen)

AVG(Spalte) = Durchschnittswert
COUNT(Spalte) = Anzahl aller Einträge
MAX(Spalte) = Maximalwert
MIN(Spalte) = Minimalwert
SUM(Spalte) = Summe aller Einträge in einer Spalte

Gruppierungsfunktionen können nur anstelle eines Spaltennamens direkt hinter der SELECT-Anweisung verwendet werden. Sie liefern genau einen Wert, beziehen sich jedoch auf mehrere Tabellenzeilen.

Beispiele:

1. SELECT COUNT(Ort)
   FROM SchülerAdresse
   WHERE Ort = `Koblenz‘
   Es wird die Zahl 5 ausgegeben, weil der Ort Koblenz fünf Mal in der Tabelle SchülerAdresse aufgeführt wird. Allerdings ist die Spaltenüberschrift Count nicht sehr aussagekräftig. Besser ist daher:
   SELECT COUNT(Ort) **AS AnzahlKoblenz**
   FROM SchülerAdresse
   WHERE Ort = `Koblenz‘
   Hier lautet die Spaltenüberschrift nun AnzahlKoblenz.

2. SELECT Ort, COUNT(Ort) AS Anzahl Ort
   FROM SchülerAdresse
   GROUP BY Ort
   HAVING COUNT(Ort) > 1
   Es wird nur Koblenz, 5 ausgegeben, da nur Orte ausgegeben werden, die mehr als einmal in der Tabelle SchülerAdresse vorkommen. Mit dem Schlüsselwort **HAVING** lässt sich eine Gruppierung einschränken.

3. SELECT Vorname, Name, Strasse, PLZ, Ort
   FROM Schüler, SchülerAdresse
   WHERE Schüler.SNR = SchülerAdresse.SNR;
   Es werden alle Schüler mit Namen und Adresse ausgegeben. Die Abfrage entspricht einem Join der Tabelle Schüler und SchülerAdresse. Haben zwei oder mehrere Tabellen das gleiche Attribut (z.B. SNR), dann muss zur eindeutigen Identifizierung der Tabellenname vorangestellt werden
   → Punktnotation, z.B. Schüler.SNR[23]

[23] Alle Aufgaben, die sich auf die Schülerbücherei und SQL beziehen, sind auf das Material von Tim Fruth, Bildungsserver Rheinland-Pfalz Moodle-Kurs zurückzuführen.

Umfassende Übung der Uni Bayreuth Aufgabenblatt zur Datenbank „Fußball-Bundesliga“ (Datenbanken im Informatikunterricht, 2013)

Bearbeite die gestellten Aufgaben und notiere die passenden SQL-Abfragen als Lösung

Die Datenbank enthält folgende Daten über die aktuelle Saison der ersten, zweiten und dritten Bundesliga:

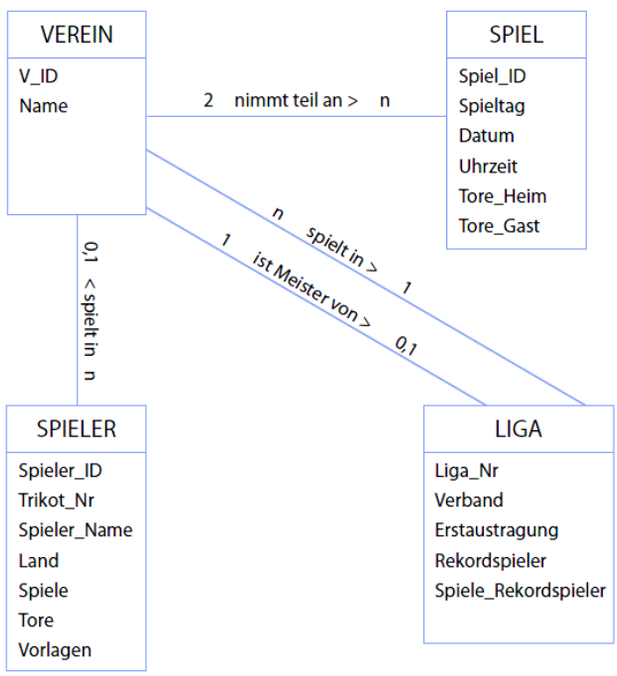
Verein (V_ID: int, Name: varchar, Liga: int)

Spiel (Spiel_ID: int, Spieltag: int, Datum: date, Uhrzeit: time, Heim: int, Gast: int, Tore_Heim: int, Tore_Gast: int)

Spieler (Spieler_ID: int, Vereins_ID: int, Trikot_Nr: int, Spieler_Name: varchar, Land: varchar, Spiele: int, Tore: int, Vorlagen: int)

Liga (Liga_Nr: int, Verband: varchar, Erstaustragung: date, Meister: int , Rekordspieler: varchar, Spiele_Rekordspieler: int)

Das Klassendiagramm soll dir einen Überblick ermöglichen:



Deine SQL-Befehle kannst du auf folgender Website testen: DBup2date.uni-bayreuth.de/bundesliga

Hinweise:

- Unterstrichene Attribute sind Primärschlüssel, Attribute mit einem Überstrich Fremdschlüssel .
- Aufgaben, die mit **(!)** gekennzeichnet sind, können nur mit einer Unterabfrage gelöst werden. Teste diese zuerst allein und füge sie dann in die Haupt-Abfrage ein.
- Setze, wo es erforderlich ist, die Aggregatfunktionen **COUNT**(…), **SUM**(…), **AVG**(…), **MIN**(…) und **MAX**(…) ein, um die Aufgaben zu lösen.

Abfragen über eine Tabelle durch Projektion und Selektion

1. Zeige alle verfügbaren Daten der Tabelle „Verein“ an.

2. Welche Vereine spielen in der ersten Liga?

3. Wann war die Erstaustragung eines Spiels der ersten Fußball-Bundesliga?

4. Wähle Liga_Nr, Verband und Rekordspieler aller drei Ligen aus.

5. Welche Ausgabe wird durch die folgende SQL-Abfrage erzeugt?

   SELECT Liga_Nr, Erstaustragung, Meister
   FROM Liga
   WHERE Spiele_Rekordspieler > 500

6. An welchem Tag fand das erste Spiel in dieser Saison statt?

7. An wie vielen Spielen haben die Rekordspieler aller drei Ligen insgesamt teilgenommen?

8. Welche Spieler haben in dieser Saison bereits mehr als fünf Tore geschossen?
   Ordne Sie absteigend nach der Anzahl ihrer Tore.

9. Wie viele Spieler tragen die Trikot-Nr 12? Benenne die Ergebnisspalte in „Anzahl“ um.

10. Welche deutschen Spieler (Land: D) haben in dieser Saison noch an keinem Spiel teilgenommen?

11. Zeige die Daten aller Spiele an, die am ersten Spieltag aller drei Ligen nach 17 Uhr begonnen haben.

12. Wer ist der Rekordspieler der zweiten Bundesliga und an wie vielen Spielen hat er teilgenommen?

13. Wie viele Tore wurden bisher durchschnittlich von den Spielern geschossen, die schon an mehr als zehn Spielen teilgenommen und mehr als drei Vorlagen geliefert haben?

14. Liste alle Spiele auf, die im August stattgefunden und nach 19 Uhr begonnen haben.

Abfragen über mehrere Tabellen durch Verknüpfung (Join)

Über das **Kreuzprodukt** werden alle Datensätze aller Eingabetabellen miteinander kombiniert. Denke daran, das Ergebnis mit deiner Abfrage so einzuschränken, dass nur sinnvolle Kombinationen übrig bleiben.

15. Welcher Verein ist aktuell Meister der ersten Liga?

16. Wer (Name) hat am ersten Spieltag gegen „Dynamo Dresden“ gespielt? Finde die V_ID von „Dynamo Dresden“ zuvor mit einer eigenen SQL-Abfrage heraus.

17. Welche Spieler spielen für den Verein “FC Bayern München“? Gib auch die Trikotnummer und das Heimatland jedes Spielers sowie die Anzahl seiner Tore mit aus. Ordne die Ergebnisse aufsteigend nach der Trikotnummer.

18. Welche Ausgabe wird durch die folgende SQL-Abfrage erzeugt? **(!)**

```
SELECT Spieler_Name, Land
FROM Spieler, Verein
WHERE Vereins_ID = V_ID AND V_ID = ( SELECT V_ID
                                     FROM Verein
                                     WHERE Name = "FC Augsburg"   )
```

19. Wie viele Spieler hat jeder Verein der ersten Liga? Gib die Ergebnisse mit dem Vereinsnamen aus und ordne sie absteigend nach der Anzahl der Spieler.

20. An welchen Tagen finden die Spiele der ersten Liga statt?
Hinweis: Jedes Datum darf nur einmal ausgegeben werden.

21. Welcher Verein hat bisher die meisten Tore geschossen? **(!)**
Hinweis: Mit dem Vergleich **>= ALL(…Unterabfrage…)** *in der* ***HAVING****-Klausel kann man den größten der durch GROUP BY ermittelten Werte bestimmen.*

22. Wie viele Tore sind bisher in jeder Liga gefallen?

23. Zeige an, welche brasilianischen Spieler, die für Vereine der ersten Liga spielen, bisher an wie vielen Spielen teilgenommen haben. Gib außerdem die Anzahl ihrer Tore und Vorlagen und den Namen des Vereins aus, für den sie spielen.
Hinweis: Finde zuerst heraus, welche Abkürzung in der Datenbank für Brasilien steht.

24. Gib Trikotnummer, Name und die Anzahl der Tore aller Spieler der zweiten Liga, die bisher schon mehr als 10 Tore geschossen haben, geordnet nach der Anzahl ihrer Tore aus.

25. Welche Vereine haben bisher gegen den Verein mit der V_ID 10 gewonnen? Wie lauteten die Ergebnisse dieser Spiele?

26. Welcher Spieler hat bisher für den „1. FC Nürnberg“ die meisten Tore geschossen? **(!)**

27. Welche Vereine haben am ersten Spieltag der ersten Liga gegeneinander gespielt, wie lauten die Ergebnisse?

28. Gegen welche Vereine hat der „FC Schalke 04“ bisher Auswärtsspiele bestritten? **(!)**

29. Wie viele Spiele hat „Hannover 96“ bis heute gewonnen? **(!)**

30. Welche Vereine (Name, Liga) haben bisher schon mindestens fünfmal unentschieden gespielt? Ordne das Ergebnis aufsteigend nach der Liga und absteigend nach der Zahl der Unentschieden.

    *Hinweis: Statt das aktuelle Datum direkt einzugeben kannst du auch die Funktion **NOW( )** verwenden (z.B. „Datum < NOW()“.*
31. Gesucht sind Vereinsname, Spieler_ID, Trikotnummer und Name aller Spieler, die für den Verein spielen, der in dieser Saison die meisten Niederlagen erlitten hat (auch mehrere Vereine mit gleicher Anzahl möglich). **(!)**
    Hinweis: **>= ALL(…)**

32. Gib die aktuelle Spieltabelle der 1. Bundesliga aus. Diese beinhaltet für jeden Verein: Den Vereinsnamen, die Anzahl der bisher gespielten Spiele, die Anzahl der Siege, Unentschieden und Niederlagen, die geschossenen und erhaltenen Tore, die Tordifferenz und die Anzahl der Punkte.
    Hinweis: Bei jedem Spiel gilt: Sieg = 3 Punkte, Unentschieden = 1 Punkt, Niederlage = 0 Punkte

Einfügen, Ändern und Löschen von Datensätzen

Die SQL-Abfragen zu den folgenden Aufgaben können nicht über die Web-Oberfläche, sondern nur bei einer lokal betriebenen Datenbank ausgeführt werden.

[24]

33. Trage dich selbst als Spieler bei deinem Lieblingsverein ein.
    Hinweis: Vorher nachschauen welche Trikotnummern noch frei sind. Die Spieler_ID wird automatisch erzeugt (auto-increment).

34. Trage in die Tabelle „Liga“ die Daten der Regionalliga Süd ein:

    Liganummer: 4, Verband: Süddeutscher Fußball-Verband, Erstaustragung: 4. August 1963, Meister: SV Darmstadt 98, Rekordspieler: Thorsten Bauer, Spiele des Rekordspielers: 34 **(!)**

35. Die 3. Liga hat einen neuen Rekordspieler: „Max Mustermann“ mit 222 Spielen. Passe die Tabelle „Liga“ entsprechend an.

36. Philipp Lahm wechselt zum „1. FC Nürnberg“. Ändere die Tabelle „Spieler“ entsprechend ab. **(!)**

37. Was bewirkt die folgende SQL-Anweisung? **(!)**

```
UPDATE Spiel
SET Uhrzeit = "15:00:00"
```

[24] Man kann die Datenbank downloaden und dann in XAMPP importieren.

WHERE (Heim = (SELECT V_ID FROM Verein WHERE Name = "Hertha BSC") OR Gast = (SELECT V_ID FROM Verein WHERE Name = "Hertha BSC")) AND Spieltag >= 5 AND Uhrzeit > "18:00"

38. Die Trikotnummer 12 soll ab sofort nicht mehr vergeben werden, um den Fußball-Fan als „12. Mann“ zu würdigen. Lösche alle Spieler, die momentan die Trikotnummer 12 tragen.

39. Das Spiel am 38. Spieltag der 3. Liga, an dem der Verein mit den aktuellen wenigsten Toren teilnimmt, wurde abgesagt. Lösche den entsprechenden Datensatz aus der Tabelle „Spiel“. **(!)** *Hinweis:* **<= ALL(…)**

40. Ab sofort sollen keine Spieler-Daten mehr in der Datenbank erfasst werden. Lösche daher die Tabelle „Spieler“ inklusive aller darin enthaltenen Datensätze.

Nachtrag: Optionalitäten von Beziehungen

Wir haben bereits über die Kardinalität von Beziehungen gesprochen. Beziehungen können aber auch daraufhin untersucht werden, ob eine Beziehung zwangsläufig vorliegen muss oder ob es nur die Möglichkeit einer Beziehung gibt. Man unterscheidet daher die MUSS- von der KANN- Beziehung.

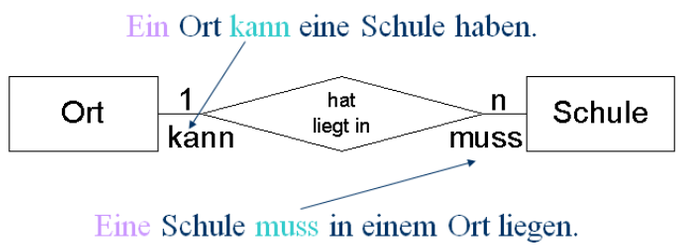
Muss ein Objekt des Typs A mit **mindestens** einem Objekt des Typs B in Beziehung stehen?

- Ja → nicht optional, obligatorisch, Muss-Beziehung
- Nein → optional, fakultativ, Kann-Beziehung

Beispiel:

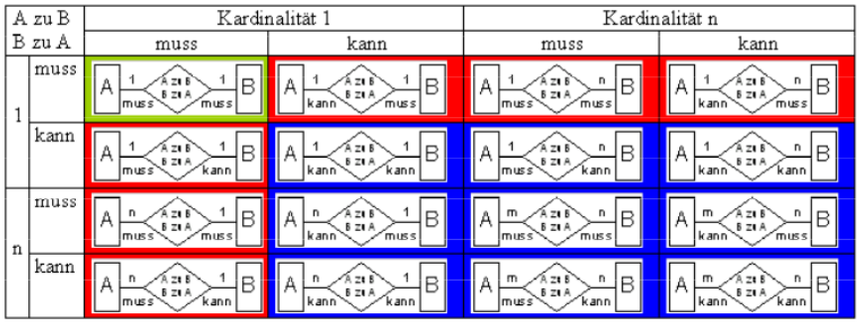
1. Muss eine Schule in mindestens einem Ort liegen?
   Ja, nicht optional, Muss-Beziehung
2. Muss ein Ort mindestens eine Schule haben?
   Nein, optional, Kann-Beziehung

Darstellung:



(Röhner, 2003)

Die Optionalität wird an den Anfang der Beziehung geschrieben. Daraus ergeben sich bei der Übertragung eines ER-Diagramms in das Relationenmodell folgende Anzahlen von Relationen für die entsprechenden Beziehungen:



grün: eine Relation $R(\underline{a_1},..., \underline{b_1},...)$
rot: zwei Relationen $Ra(\underline{a_1},...)\ Rb(\underline{b_1},...,\uparrow a_1)$
blau: drei Relationen $Ra(\underline{a_1},...)\ Rk(\uparrow \underline{a_1}, \uparrow \underline{b_1})\ Rb(\underline{b_1},...)$

Was kann man aus dieser Tabelle ablesen:

1. Egal, welche Optionalitäten eine m:n-Beziehung aufweist, man benötigt für jede m:n-Beziehung eine eigene Tabelle.

2. Liegt eine 1:1-Beziehung vor, gibt es drei Möglichkeiten:
   (1) beidseitige MUSS-Beziehung: Man kann die Tabelle der einen Entität an die Tabelle der zweiten Entität anhängen, da alle Daten immer miteinander verknüpft sind. Man erhält für die Beziehung und die beiden Entitäten im optimalen Fall nur eine einzige Tabelle.
   (2) MUSS-KANN-Beziehung: Die obligatorische Entität (MUSS) erhält den Primärschlüssel der fakultativen Entität, um die Beziehung umzusetzen. Es entsteht keine neue Tabelle für die Beziehung.
   (3) beidseitige KANN-Beziehung: Entweder verfährt man wie in (2) und lässt für den Fremdschlüssel Nullwerte zu oder man erstellt eine eigene Tabelle, in der nur die vorkommenden Beziehungen aufgenommen werden.

3. Liegt eine 1:n-Beziehung vor, gibt es zwei Möglichkeiten:
   (1) beidseitige MUSS-Beziehung oder MUSS-KANN-Beziehung: Die obligatorische Entität (MUSS) erhält den Primärschlüssel der fakultativen Entität, um die Beziehung umzusetzen. Es entsteht keine neue Tabelle für die Beziehung.
   (2) beidseitige KANN-Beziehung: Entweder verfährt man wie in (1) und lässt für den Fremdschlüssel Nullwerte zu oder man erstellt eine eigene Tabelle, in der nur die vorkommenden Beziehungen aufgenommen werden.

Aufgaben:

1. Ergänze die Kardinalitäten und Optionalitäten

2. Ergänze im ER-Diagramm von S. 16 die Optionalitäten

Projekt – Erstellen einer eigenen Datenbank

Anforderungen an die schriftliche Ausarbeitung

Aufbau

- Übersichtliches Deckblatt
  Vor- und Zuname, Titel der Arbeit, betroffenes Unterrichtsfach, Name der Schule, Abgabezeitpunkt, Schuljahr
- Inhaltsverzeichnis
- Einleitung
  Herausarbeiten des Themas mit genauer Fragestellung, evtl. Abgrenzung des Themas, Überblick über den Aufbau, …
- Hauptteil
  - ER-Modell, Überführung in das relationale Modell und die Realisierung in XAMPP, fiktive Datensätze ergänzen, Anfragen
  - Zielgerichtete Dokumentation des Arbeitsprozesses: Vorgehensweise, Probleme, Lernfortschritt, Resultate

  Zusatz: Normalisierung des Datenbankentwurfs
- Schluss
  Zusammenfassung, persönliche Stellungnahme
- Korrektes Literaturverzeichnis (Bücher, Zeitungs- und Zeitschriftenaufsätze, Internetadressen)
- Versicherung: „Ich versichere, dass ich die Arbeit ohne fremde Hilfe angefertigt habe und nur die im Literaturverzeichnis angegebenen Quellen und Hilfsmittel verwendet habe.“ Selbstverständlich persönlich unterschrieben
- Anhang
  Ausdrucke,…

Sprachlicher Aspekt

- Klarer, verständlicher Ausdruck
- Verwendung eines dem Thema angemessenen Sprachstils (auch fachsprachliche Anteile)
- Sicherer Umgang mit Materialien und benutzten Texten (z. B. korrektes Zitieren und sprachliche Einbettung)
- Sprachliche Korrektheit (Grammatik, Rechtschreibung, Zeichensetzung)

Formaler Aspekt

- Fristgerechte Abgabe der Arbeit (sonst: 00 Punkte)
- Vollständigkeit der Arbeit
- Umfang der Arbeit max. 8 Seiten ohne Anhang
- Einsatz von Grafiken etc.
- Gestaltung der Arbeit (ordentliche Mappe, DinA4-Papier,…)

Theoretische Informatik

Warum wird in der Schule „Theoretische Informatik“ unterrichtet?

Viele Bereiche der Informatik sind den Neuerungen unterworfen, so wurde vor Jahren noch Pascal als Programmiersprache unterrichtet, heute sind es die objektorientieren Programmiersprachen Java oder Delphi. Die grundsätzlichen Ideen sind jedoch keinen zeitlichen Veränderungen unterworfen. So behält die Modellierung der Vorgehensweise von Informationsverarbeitung an sich ihre grundlegende Bedeutung. Fragen der Leistungsfähigkeit und der Grenzen von informations-verarbeitenden Maschinen stehen im Mittelpunkt der theoretischen Informatik.

In der Schule spielen drei Bereiche der Theoretischen Informatik eine entscheidende Rolle:

- Die Berechenbarkeit
- Automaten
- Sprachen und Grammatiken

Die Berechenbarkeit

Die Laufzeit

Wenn Anna versuchen würde, die Mathematikaufgabe per Hand zu lösen, würde das Stunden dauern.

Anna: „Jan, kannst du mir schnell ein Computerprogramm schreiben, das die Matheaufgabe löst? Der Computer kann das doch viel schneller als ich.“

Jan: „Der Computer kann auch nicht immer alles schneller. Alle einzelnen Schritte müssen verarbeitet werden, das kostet Rechnerzeit. Zudem sind die Rechenzeiten von verschiedenen Prozessen unterschiedlich und der Computer braucht auch Zeit, um die Prozessabläufe zu organisieren.“

Anna: „Stopp! Du hast doch einen ganz neuen Rechner. Der ist doch wahnsinnig schnell.“

Einführungsbeispiel zur Laufzeitberechnung: Ausschusssitzung von Parlamentsabgeordneten

Allgemeines Problem: Die Parlamentsabgeordneten gehören n verschiedenen Ausschüssen an. Jeder Ausschuss tagt jede Woche genau einmal. Ist ein Abgeordneter Mitglied in zwei verschiedenen Ausschüssen, so dürfen diese nicht zur gleichen Zeit stattfinden. Wir möchten wissen, ob wir mit k verschiedenen Sitzungsterminen auskommen.

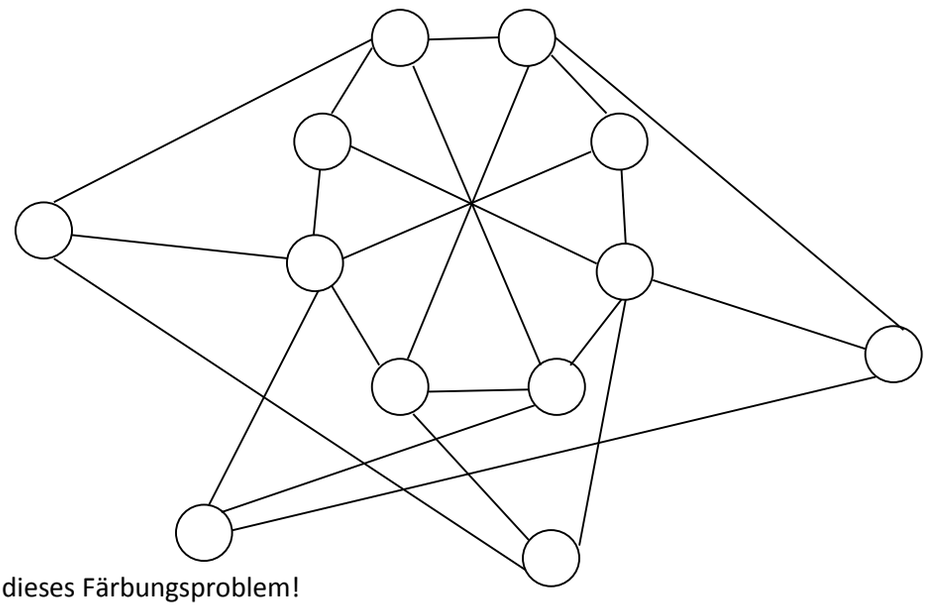
Darstellung des Problems in Form eines Graphen

- Jeder Ausschuss wird durch einen Knoten dargestellt.
- Zwei Knoten sind durch eine Kante miteinander verbunden, wenn es einen Abgeordneten gibt, der in beiden Ausschüssen vertreten ist.

Überführung in das Färbungsproblem

Lässt sich jedem Knoten des Graphen eine von k Farben so zuordnen, dass zwei durch eine Kante verbundene Knoten unterschiedliche Farben besitzen?

Konkretes Beispiel für 12 Ausschüsse und 3 Sitzungstermine



Löse dieses Färbungsproblem!

Algorithmus zur allgemeinen Lösung des Färbeproblems und dessen Laufzeit

Wiederholung:

Definition:
Ein Algorithmus ist eine Verarbeitungsvorschrift aus endlich vielen eindeutig formulierten Befehlen. Die einzelnen Operationen können Methodenaufrufe sein. Algorithmen können in natürlicher Sprache oder auch in einer Programmiersprache abgefasst werden. Elementare Strukturen von Algorithmen sind Sequenzen, Fallunterscheidungen und Wiederholungen.[25]

Algorithmus für das Färbungsproblem:

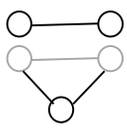
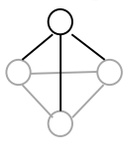
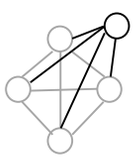
Man überprüft jede Färbemöglichkeit darauf, ob die Forderung erfüllt ist. Wie viel Zeit wird der Computer wohl für dieses Aufgabe benötigen?

n Knoten
⇒ k Färbemöglichkeiten pro Knoten
k Farben

⇒ Es gibt also insgesamt $k \cdot k \cdot \ldots \cdot k = k^n$ Färbemöglichkeiten für den Graphen

[25] Vgl. (Hubwieser & a., 2010, S. 184)

Anzahl der Knotenpaare, die überprüft werden müssen:

2 Knoten		1 Knotenpaar	
3 Knoten		3 Knotenpaare	1 + 2
4 Knoten		6 Knotenpaare	1+2 +3
5 Knoten		10 Knotenpaare	1+2+3 + 4
n Knoten		??? Knotenpaare	1+2+3+…+(n-1) (arithmetische Reihe)

Ein Blick in die Formelsammlung verrät: 1+2+3+…+ n = $\sum_{i=1}^{n} i = \frac{n}{2}(n+1)$ also ist 1+2+…(n-1) = $\frac{n-1}{2}n$
Die Überprüfung sämtlicher Knotenpaare auf Farbübereinstimmung benötigt demnach maximal Schritte, $\frac{n}{2}(n-1)\cdot k^n$ denn es müssen nur Knotenpaare überprüft werden, die durch eine Kante verbunden sind.
Daraus ergibt sich eine Laufzeit von $\frac{n}{2}(n-1)\cdot k^n$. Was bedeutet die Zahl nun?

Aufgabe:

Für k = 3 ergibt sich für die max. Anzahl von Schritten der Term $T(n) = \frac{n}{2}(n-1)3^n$.
Berechne die sich hieraus ergebene Laufzeit für die angegebenen Knotenzahlen, wenn der verwendete Computer (Laufzeit 1) 100.000 Schritte pro Sekunde ausführen kann. Berechne auch die Laufzeit für den Computer mit der Laufzeit 2, der 100.000.000 Schritte pro Sekunde ausführen kann.

Knotenanzahl	12	20	30	50	100
Laufzeit 1					
Laufzeit 2					

Interpretiere deine Ergebnisse!

Vergleich polynomielles und exponentielles Wachstum (Battenfeld & u.a., 1996, S. 7)

	n				
T(n)	20	30	40	50	100
n	0,0002 s	0,0003 s	0,0004 s	0,0005 s	0,001 s
n^2	0,004 s	0,009 s	0,016 s	0,025 s	0,1 s
n^5	32 s	243 s = 4 min 3 s	1024 s = 17 min 4 s	3125 s = 52 min 5 s	100000s = 27 h 47 min
2^n	10 s	3 h	4 Monate	360 a	$4 \cdot 10^{17}$ a

Merke: Algorithmen mit exponentieller Laufzeit sind (schon für relativ kleine Eingabelängen n) praktisch undurchführbar.

Wir unterscheiden Algorithmen hinsichtlich ihrer Laufzeit:

berechenbar und durchführbar (polynomielle Laufzeit)	berechenbar und nicht durchführbar (exponentielle Laufzeit)

Aufgabe:

Ein Mann hat seinen Reichtum in n Goldklumpen angelegt, die die Gewichte $g_1, g_2, \ldots, g_n$ besitzen. Nach seinem Tod soll der Reichtum zu gleichen Teilen an seine beiden Kinder fallen. Im Testament wurde jedoch festgelegt, dass kein Goldklumpen zerschlagen werden darf. Gelingt die exakte Aufteilung nicht, fällt das gesamte Vermögen an die Kirche.
Gibt es eine Aufteilung der Menge $\{ g_1, g_2, \ldots, g_n\}$ in zwei Mengen, so dass die Summen der Zahlen jeder Teilmenge gleich sind?
Betrachte den folgenden konkreten Fall (Gewicht g_i in Gramm):
n = 8 und $g_1 = 10$, $g_2 = 14$, $g_3 = 19$, $g_4 = 30$, $g_5 = 38$, $g_6 = 40$, $g_7 = 45$, $g_8 = 56$
a) Suche eine Lösung für den konkreten Fall.
b) Bestimme allgemein die Laufzeit eines naiven Algorithmus, der alle Möglichkeiten der Aufteilung von n Gewichten auf die beiden Erbteilmengen nach einer Lösung durchsucht.

Beispiel: Wundersame Zahlen
Starte mit einer beliebigen natürlichen Zahl und bestimme die Folgezahlen jeweils wie folgt:

- Ist die Zahl ungerade, so ist das nächste Folgeglied das Dreifache dieser Zahl erhöht um 1.
- Ist die Zahl gerade, so ist das nächste Folgeglied die Hälfte dieser Zahl.

Die Startzahl heißt wundersam, falls die Folge irgendwann auf den Wert 1 stößt.

a) Finde einige wundersame Zahlen (Bleistift und Papier)
b) Teste mit Hilfe des folgenden Java-Programms einige Zahlen auf Wundersamkeit (am Rechner).

```
import java.util.Scanner;

public class wundersam {

  public static void main(String[] args) {
    int startwert, n;
    Scanner tastatur = new Scanner(System.in);
    System.out.println("Gib den Startwert ein: ");
    startwert=tastatur.nextInt();
    System.out.println();
    n=startwert;
    do {
      if (n % 2 == 0) {    //n gerade, Restwertoperator
        n = n/2;
      }
            else {
        n = 3*n +1;
      }
    } while (n>1);
    System.out.println("Die Zahl "+ startwert +" ist wundersam.");
  }
}
```

Zeitkomplexität (Achtung Theorie)

„Der Begriff der Zeitkomplexität eines Algorithmus wurde 1960 von Rabin eingeführt. Sie ist eine arithmetische Funktion, die den Aufwand an Rechenzeit in Abhängigkeit vom Umfang des Problems (Anzahl der Eingabedaten, Grad des Polynoms o. ä.) ausdrückt. Man unterscheidet dabei zwischen dem Aufwand im Mittel (average case) und dem Aufwand im schlechtesten Fall (worst case). Dabei muss man sich meist damit begnügen, den Zeitaufwand größenordnungsmäßig abzuschätzen. Um solche Größenordnungen von Funktionen auszudrücken, hat sich die sogenannte **Groß-Oh-Notation** bewährt:
Ein Algorithmus mit einem Rechenzeitaufwand T(n) hat die Zeitkomplexität **O**(g(n)), sofern es eine Konstante c > 0 gibt, so dass T(n) ≤ c · g(n) $\forall n \in \mathbb{N}$.“ (Breier, S. 1)
Funktionen für g(n) sind zum Beispiel:

- n^k mit k > 0 und $k \in \mathbb{N}$
- log n
- n log n
- 2^n, 3^n,… oder
- n!

Beim Abschätzen können konstante Faktoren vernachlässigt werden, da man nur nach einer Schranke sucht, die nicht überschritten wird.

Probleme und ihre Berechenbarkeit

Was verstehen wir in der theoretischen Informatik unter einem Problem?

Das Beispiel der wundersamen Zahlen kann man als Zuordnung der natürlichen Zahlen auf {0,1} verstehen. Mathematisiert lässt sich dies wie folgt darstellen:

$$f: \mathbb{N} \to \{0,1\}$$
$$x \mapsto \begin{cases} 1 & x \text{ ist wundersam} \\ 0 & sonst \end{cases}$$

Kann man die Funktionswerte berechnen?

- Prinzipiell erkennt das Programm (der letzten Stunde) wundersame Zahlen, also kann f(n) = 1 berechnet werden.
- Für nicht-wundersame Zahlen kann das Programm jedoch kein f(n) = 0 zurückgeben. Es könnte einen anderen Algorithmus geben, der auch für nicht-wundersame Zahlen terminiert, aber bis heute hat niemand einen solchen Algorithmus gefunden.

Verallgemeinerung:

Jede Eingabe und jede Ausgabe (Texte, Zahlen, Bilder,…) werden auf der untersten formalen Ebene als Folge von Nullen und Einsen dargestellt. Dann ist jede Eingabe x eine natürliche Zahl und ebenso jede Ausgabe y.

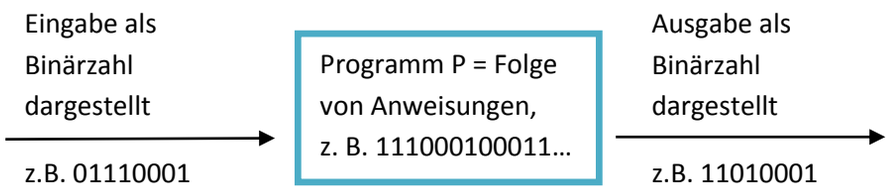
Definition: Ein Problem ist eine Funktion $f: \mathbb{N} \to \mathbb{N}$.

Leitfrage: Gibt es Probleme, die nicht lösbar sind?

Wie berechnet man die Funktionswerte eines Problems?

Für die Berechnung benötigen wir einen Algorithmus.
Ein **Algorithmus** ist eine eindeutige, endliche Folge von Anweisungen. Er lässt sich durch ein Programm darstellen.

Ein **Programm** ist, wie immer es auch formuliert und aufgeschrieben ist, letztendlich eine Folge aus 0 und 1, d. h. als Dualzahl interpretierbar. Ein Programm ist also eine Zahl aus $\mathbb{N}$



Das bedeutet: Es gibt höchstens so viele berechenbare Programme wie Zahlen in $\mathbb{N}$.

Wie viele formale Probleme (keine Alltagsprobleme) gibt es?

Laut Definition gibt es genau so viel Probleme wie Funktionen f von $\mathbb{N}$ nach $\mathbb{N}$. Der Definitionsbereich der Funktionen ist immer eine Teilmenge von $\mathbb{N}$, also gibt es mindestens so viele Funktionen wie Teilmengen von $\mathbb{N}$. Beachtet man noch die verschiedenen Wertebereiche, so gibt es noch viel mehr.

Bleibt also die Frage: Wie viele Teilmengen hat $\mathbb{N}$?

Aufgabe:

Gegeben ist die Menge A = {a, b, c}.

a) Bestimme die Menge aller Teilmengen von A. (Hinweis: Auch die leere Menge ist eine Teilmenge!)
b) Füge der Menge A ein weiteres Element hinzu und bestimme für die so entstehende Menge A‘ wiederum die Menge aller Teilmengen.
c) Eine Menge M habe n Elemente. Wie viele Teilmengen hat M? Begründe!

Definition: Die Menge aller Teilmengen von M heißt Potenzmenge. Bezeichnung: P(M)

Vorsicht mit der Unendlichkeit:

- $\mathbb{N}$ ist unendlich, aber durchzunummerieren.
- $\mathbb{G}$ sei die Menge aller positiven ganzen Zahlen, dann ist $\mathbb{G} \subsetneq \mathbb{N}$ (echte Teilmenge!). Das heißt, dass $\mathbb{G}$ „weniger“ Elemente als $\mathbb{N}$ enthält. Dennoch kann man $\mathbb{G}$ mit Hilfe der natürlichen Zahlen durchnummerieren.

Definition:
Eine Menge M ist „gleichmächtig zu $\mathbb{N}$ “, wenn man die Elemente von M durchnummerieren kann, das heißt M = {m_1, m_2, …}.
Statt „M ist gleichmächtig zu $\mathbb{N}$ sagen wir auch M ist abzählbar unendlich.

Beispiele:

- $\mathbb{Q}$ ist abzählbar mit für $\frac{p}{q}$ $ist\ i = \frac{(p+q-2)^2+3p+q-2}{2}$
- $\mathbb{R}$ ist nicht abzählbar, also überabzählbar.
- Ist die Potenzmenge von $\mathbb{N}$ abzählbar unendlich?

Satz: P($\mathbb{N}$) ist nicht abzählbar unendlich, d. h. überabzählbar.

Beweis durch Widerspruch:

Annahme: P($\mathbb{N}$) ist abzählbar unendlich.

⇒ Man kann die Teilmengen von $\mathbb{N}$ durchnummerieren, also T_1, T_2, T_3, …

$\Rightarrow \forall i : i \in T_i$ oder $i \notin T_i$

Sei $D = \{i \in \mathbb{N};\ i \notin T_i\}$, d. h. D umfasst alle Zahlen, die nicht in ihrer entsprechenden Teilmenge enthalten sind.

⇒ $D \subseteq \mathbb{N}$

⇒ $D \in P(\mathbb{N})$, dann gilt auch $D = T_n$

Was ist nun mit n?

Falls $n \in D$, dann ist $n \notin T_n$ (laut Definition von D) $\Rightarrow n \notin D$ Widerspruch

Falls $n \notin D$, dann ist $n \in T_n$ (laut Definition von D) $\Rightarrow n \in D$ Widerspruch!

Also ist die Annahme falsch und der Satz gilt.

Was bedeutet dies für die Anzahl der formalen Probleme?
Die Menge aller Programme ist eine Teilmenge von $\mathbb{N}$, also abzählbar.
Die Menge aller Probleme ist mind. so groß wie P($\mathbb{N}$), also überabzählbar.

⇒ Es gibt mehr formale Probleme als Programme!

Das Halteproblem

Beispiele für nicht berechenbare Probleme (Paradoxa):

- Der Barbier eines Dorfes ist bartlos. Er rasiert genau die Männer des Ortes, die sich nicht selbst rasieren. Was ist mit dem Barbier selbst?
  Wenn er sich nicht selbst rasiert, rasiert er nicht alle Männer, die sich nicht selbst rasieren. Wenn er sich aber selbst rasiert, rasiert er auch mindestens einen Mann, der sich selbst rasiert.
- „Der nächste Satz ist falsch." „Der vorhergehende Satz ist wahr."
- Aus der Mathematik sind einige Probleme bekannt, die bisher algorithmisch nicht lösbar waren. Es könnte sich um nicht berechenbare Probleme handeln, z. B. die Vermutung von Fermat, die Unendlichkeit der Menge von Primzahlenzwillingen oder die Vermutung von Goldbach.

(Magenheim, 2009, S. 82)

Rückblick zum Beginn des Skripts:
Jan hat für Anna ein Computerprogramm geschrieben. Nun stellt sich die Frage, ob dies Programm auch für jede Eingabe, die Anna tätigt, terminiert.
Also

 oder 

Wenn Jan ein Programm hätte, welches entscheiden könnte, ob sein geschriebenes Programm immer terminiert oder eventuell in eine Endlosschleife gerät, könnte er es vorher überprüfen. Gibt es ein Programm, welches das Halteproblem löst?

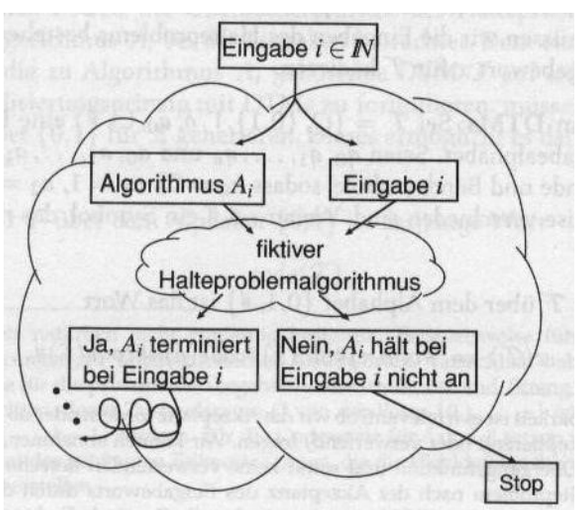
Halteproblem in Funktionsform

$$f: \mathbb{N} \rightarrow \{0,1\}$$

$$P \mapsto \begin{cases} 1 & P \text{ enthält eine Endlosschleife} \\ 0 & \text{sonst} \end{cases}$$

Satz von Turing 1936

Das Halteproblem ist nicht entscheidbar.[26] Es gibt keinen Algorithmus, der entscheiden kann, ob ein beliebiges Programm P mit seiner Eingabe E in eine Endlosschleife gerät oder nicht.

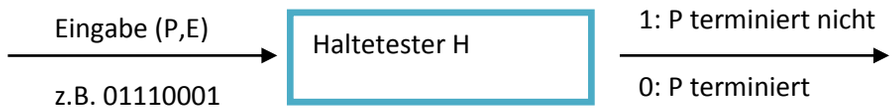


(Asteroth & Baier, 2002, S. 101)

Widerspruchsbeweis:[27]

Annahme: Es gibt ein universelles Programm H (Haltetester), welches für ein beliebiges, fest gewähltes Programm P, den Funktionswert f(P) berechnen kann.

[26] Gibt es für ein Ja-/Nein-Problem einen Algorithmus, der für alle Eingaben terminiert und die korrekte Antwort „Ja“ oder „Nein“ liefert, dann ist das Problem entscheidbar.
[27] Vorsicht, der Beweis kann zu Knoten in den Gehirnwindungen führen. Sollte es dazu kommen, bitte Ruhe bewahren, tief durchatmen, dann lösen sich die Knoten hoffentlich wieder. Ansonsten eine Nacht drüber schlafen.



Das Programm würde vom Grundgerüst so aussehen: (als Beispiel im Java-Quellcode)

```
Haltetester(Binaerzahl P, Binaerzahl E) {
  boolean endlos;
  //hier wird das eingegebene Programm untersucht und je nach
  //Ergebnis die Variable endlos auf true oder false gesetzt

  if (endlos) {
    System.out.println("Das Programm enthält eine Endlosschleife.");
  }
  else {
    System.out.println("Das Programm enthält keine Endlosschleife);
  }
}
```

Mit Hilfe von Haltetester H können wir nun ein Programm Seltsam() erstellen, welches wie folgt arbeitet:

```
class Seltsam {
  static boolean endlos;
  public static void main(String[] args) {
    //hier wird das eingegebene Programm untersucht und je nach
    //Ergebnis die Variable endlos auf true oder false gesetzt, wie
    //im Haltetester!

    if (endlos) {
      System.out.println("Das Programm terminiert nicht.");
    }
    else {
      System.out.println("Das Programm terminiert.");
      while (1 == 1) {
        // Endlosschleife}
      }
    }
  }
}
```

Der Unterschied zwischen den beiden Programmen Haltetester und Seltsam liegt lediglich in der eingebauten Endlosschleife. Terminiert nun Seltsam? Wenden wir Seltsam auf Seltsam an.

- Nehmen wir an, Seltsam terminiert.
  Dann wird zunächst der Testteil ausgeführt und die boolesche Variable endlos auf false gesetzt, weil Seltsam nach Annahme terminiert. Es wird der Text „Das Programm terminiert.“ ausgegeben und das Programm läuft in eine Endlosschleife. Dies steht im Widerspruch zur Annahme.
- Dann muss gelten, dass Seltsam nicht terminiert.

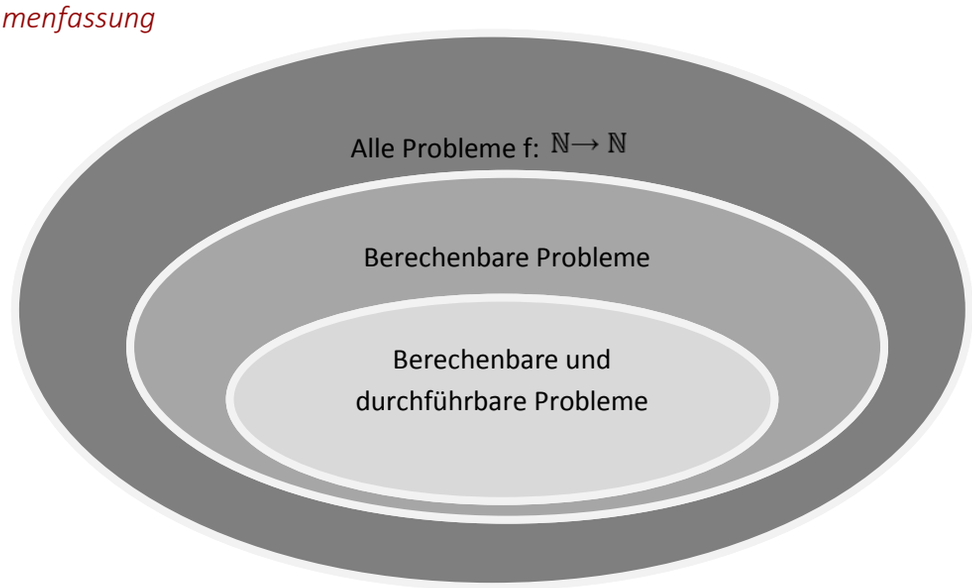
Wieder wird im Testteil überprüft, ob Seltsam terminiert. Die Variable endlos wird diesmal auf true gesetzt. Es folgt die Ausgabe „Das Programm terminiert nicht.“ Und nun stoppt das Programm. Auch dies steht im Widerspruch zur Annahme.

Das Programm Seltsam trägt also seinen Namen zu Recht. Es kann nicht entschieden werden, ob es terminiert oder nicht. Nach den Gesetzen der Logik kann es ein solches Programm nicht geben. Dann kann es aber auch das Programm Haltetester nicht geben, da Seltsam aus Haltetester regelkonform konstruiert wurde. Damit ist der Satz von Turing bewiesen.

Bemerkungen zur Bedeutung des Satz von Turing

- Der Satz von Turing hat sicherlich viele Programmierer davor bewahrt, viel Zeit in die Suche nach einem Haltetester zu investieren. Denn es wäre ja segensreich, wenn man ein solches Programm hätte. Die großen Softwarefirmen hätten sicherlich viel Geld in dies Projekt gesteckt. Der Haltetester hätte in Betriebssysteme integriert werden können, um „schlechte“ Programme zu filtern.
- Ob ein beliebiges Programm nach endlich vielen Schritten anhält oder nicht, lässt sich nur feststellen, indem man das Programm mit den Eingaben E startet und abwartet…
- Alle unentscheidbaren Probleme sind mit Hilfe des Computers nicht lösbar.
- Der Mensch trägt immer die Verantwortung für die Terminierung der erstellten Programme.
- Es gibt kein sicheres Virenprüfprogramm: Satz von Dowling (1989)
  Es ist unmöglich ein Programm zu formulieren, das selber sicher ist (keinen Virus verbreitet) und von jedem beliebigen Programm entscheiden kann, ob dieses virenfrei ist oder nicht.
- Es gibt kein Programm, das für jedes Navigationsprogramm dessen Korrektheit überprüft.
- Gäbe es eine Lösung für das Halteproblem-Programm, könnten viele noch offene mathematische Fragestellungen berechnet werden. Man könnte ein Programm schreiben, das die natürlichen Zahlen nach dem Auftreten von Primzahlzwillingen untersucht und stoppt, wenn es keine weiteren findet. Aber diese Frage wird wohl offen bleiben, wie viele andere auch!

Zusammenfassung



Automaten

(Magenheim, 2009, S. 64)

Endliche Automaten (EA)

Beispiel: Der Getränkeautomat

Ein Getränkeautomat kann Cola und Limo ausgeben. Beide Getränke kosten 1,50 €. Es können Eurostücke und 50-Cent-Stücke eingeworfen werden. Wird der Betrag von 1,50 € überschritten, so fällt die Münze ins Geldausgabefach. Bei korrektem Geldeinwurf kann zwischen Cola und Limo gewählt werden. Zu jedem Zeitpunkt kann die Korrekturtaste betätigt werden, das bereits gezahlte Geld fällt in den Geldausgabeschacht. Der Automat ist immer betriebsbereit und hat alle Getränke vorrätig.

Eingaben / Eingabealphabet	0,50 €, 1 €, Colataste, Limotaste, Korrekturtaste	Welche Eingaben können vom Bediener getätigt werden?
Ausgaben / Ausgabealphabet	0,50 €, 1 €, 1,50 €, Cola, Limo, nichts	Was gibt der Automat zurück?
Zustände / Zustandsmenge	kein Geld, 0,50 €, 1 €, 1,50 €	Welche Zustände erreicht der Automat?
Anfangszustand	kein Geld	Wie findet der Bediener den Automaten vor?
Endzustand	kein Geld	Welchen Endzustand weist der Automat auf?

Darstellungsmöglichkeiten von Automaten

1. Die Zustandstabelle

Eingabe / Zustand	0,50 €	1 €	Cola-Taste	Limo-Taste	Korrekturtaste
kein Geld	0,50 € / N				
0,50 €					
1 €					
1,50 €					

Folgezustand / Ausgabe (N = nichts)

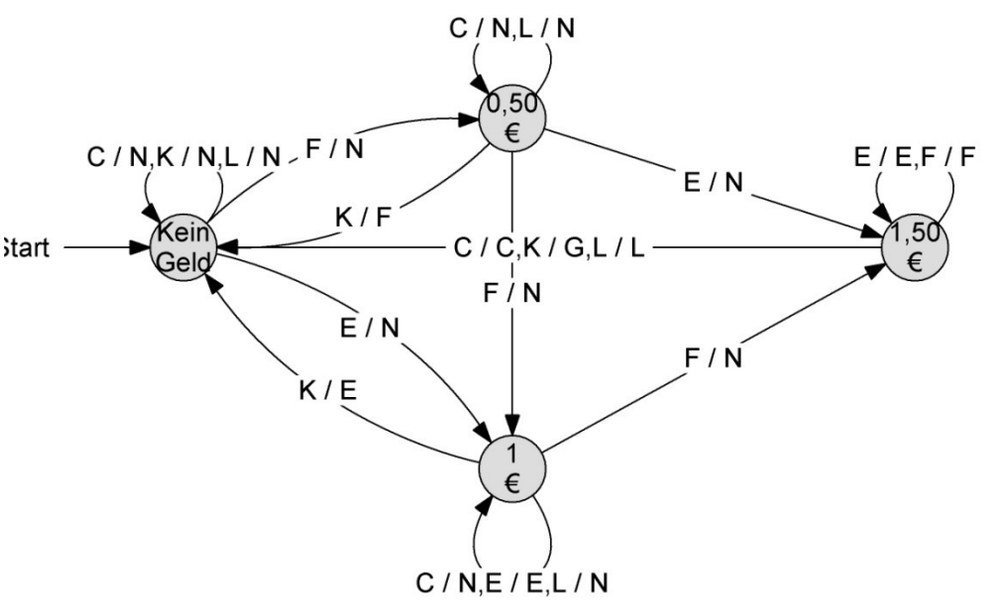
2. Das Zustandsdiagramm / Der Zustandsgraph

Das Verhalten des Automaten kann übersichtlich durch einen Zustandsgraphen beschrieben werden. Zustände werden dabei als Knoten dargestellt. Der Knoten, der dem Anfangszustand entspricht, wird durch einen hineingehenden Pfeil besonders markiert. Der Doppelkreis kennzeichnet alle möglichen Endzustände (beim Getränkeautomat gibt es nur einen Endzustand). Die gerichteten Kanten zeigen mögliche Übergänge von einem Zustand zu einem Folgezustand. Die sind in der Form E / A beschriftet, dabei gibt E an, welche Eingabe den Zustandsübergang herbeiführt. A gibt an, welche Ausgabe der Automat vornimmt. Mehrfachbeschriftungen sind sinnvoll. Sämtliche Eingaben und Ausgaben sollen mit nur einem Buchstaben abgekürzt werden.

Bezeichnungen für den Getränkeautomat:

L : Limo/-taste C: Cola/-taste F: 50 Cent E: 1 Euro K: Korrekturtaste

G: Geld (1,50€) N: nichts

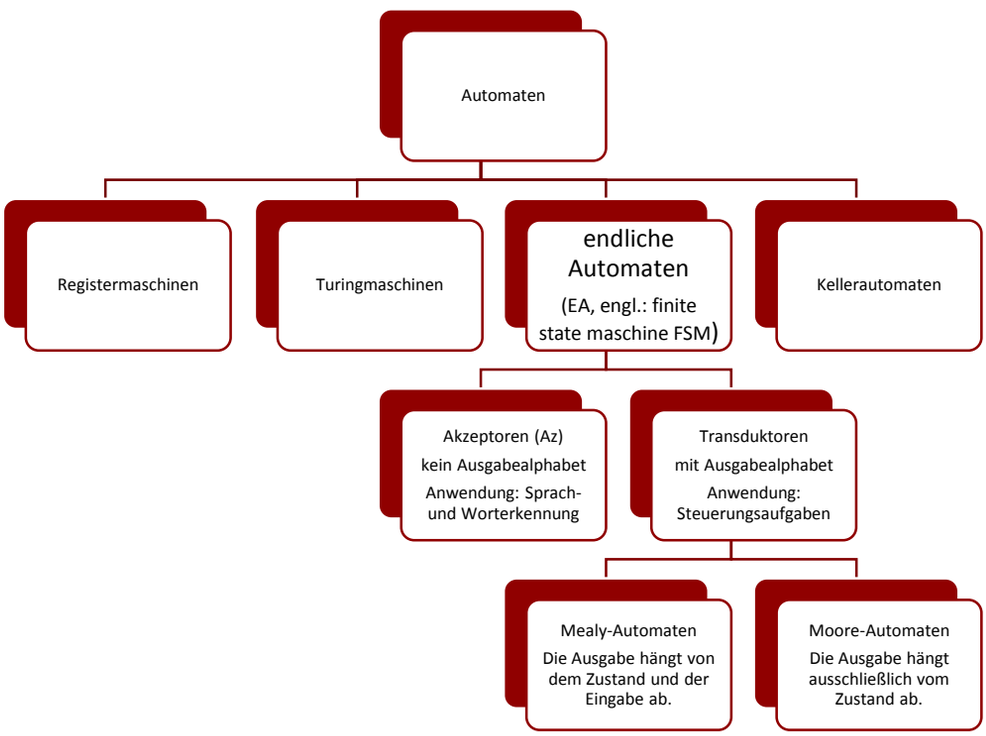


Zur Darstellung mit Hilfe des Computers stehen verschiedene Programme zur Verfügung:

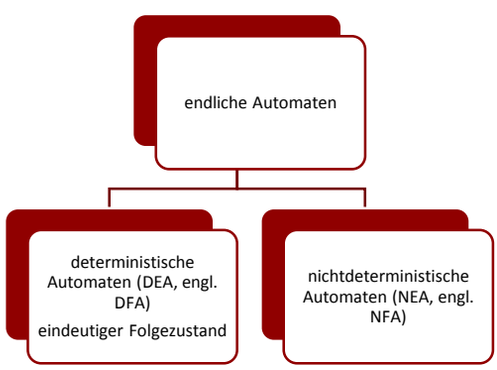
- JFLAP Das Programm kann aus dem Internet kostenlos heruntergeladen werden (englisch). Man kann Automaten darstellen und auch testen, wobei dann eine etwas andere Beschriftung der Kanten gewählt werden muss.
- AutoEdit (siehe oben). Auch hier kann man Automaten konstruieren und testen (deutsch) (AtoCC)

Für die Bedienung dieser Programme, muss man entscheiden können, welchen Automatentyp man konstruieren will, daher folgt hier bereits eine Darstellung der Automatentypen.

Klassifizierung von Automaten



Endliche Automaten lassen sich noch auf eine weitere Art unterscheiden. Betrachtet man die Möglichkeit von einem Zustand in einen Folgezustand zu gelangen, so kann man folgende Differenzierung vornehmen:[28]



[28] Näheres folgt hierzu weiter hinten.

Der deterministische endliche Automat *(DEA)*

Definition:
Ein endlicher deterministischer Automat (DEA, engl. DFA deterministic finite automation) besteht aus

- einer nichtleeren endlichen Menge von Zuständen, der Zustandsmenge Q
- einem festgelegten Startzustand $Z_0 \in Q$
- einer festgelegten Menge von Endzuständen $F \subset Q$
- einem nichtleeren endlichen Alphabet Σ mit
  $\Sigma_1 \subset \Sigma$ und Σ_1 ist die nichtleere Eingabemenge
  und $\Sigma_2 \subset \Sigma$ und Σ_2 ist die Ausgabemenge (bei Akzeptoren ist die Ausgabemenge die leere Menge)
- einer Übergangsfunktion $\delta : Q \times \Sigma \rightarrow Q$, die jedem Eingabezeichen eindeutig einen Folgezustand zuweist.
- Ggf. erweitert um eine Ausgabefunktion, die jedem Eingabezeichen und einem Zustand ein Ausgabezeichen zuordnet.

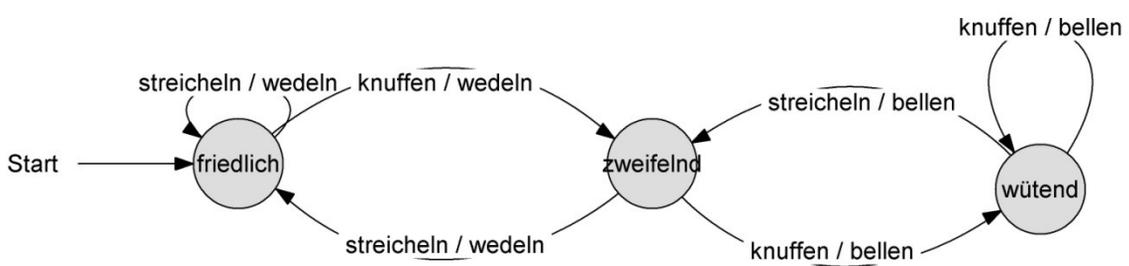
Beispiel für einen DEA mit Ausgabe: Der Hund als Automat
Der Automat Hund wird wie folgt charakterisiert:

- Eingabealphabet = {streicheln, knuffen}
- Zustandsmenge = {friedfertig, zweifelnd, wütend}
- Ausgabealphabet = {wedeln, bellen}
- Sprachliche Beschreibung der Übergangsfunktion:
  Ist der Hund friedfertig, so wedelt er nach dem Streicheln. Wird der friedfertige Hund geknufft, wedelt er weiter, geht aber in den Zustand zweifeln über. Der zweifelnde Hund wedelt und wechselt in den friedfertigen Zustand, wenn er gestreichelt wird. Wird der zweifelnde Hund allerdings geknufft, beginnt er zu bellen und wird wütend. Der wütende Hund lässt sich durch Streicheln beruhigen und wechselt in den zweifelnden Zustand, bellt jedoch weiterhin. Wird der wütende Hund geknufft, bellt er und bleibt wütend.[29]

Formal lässt sich der Automat folgendermaßen notieren:

$$M = (\{friedlich, zweifelnd, wütend\}, \{knuffen, streicheln\}, \{bellen, wedeln\}, \delta, \lambda, friedlich)$$

Das Zustandsdiagramm sieht wie folgt aus.



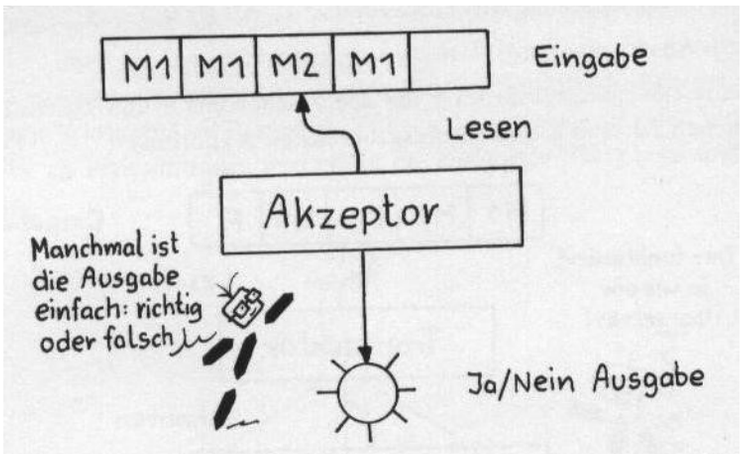
Die Übergangsfunktion und die Ausgabefunktion in Tabellenform:

[29] Der Hundeautomat sieht an dieser Stelle nicht vor, dass der Hund beißt! Dies ist ein deutlicher Unterschied zu einem echten Hund.

δ	knuffen	streicheln
friedlich	zweifelnd	friedlich
zweifelnd	wütend	friedlich
wütend	wütend	zweifelnd

λ	knuffen	streichelr
friedlich	wedeln	wedeln
zweifelnd	bellen	wedeln
wütend	bellen	bellen

Der Akzeptor



(Magenheim, 2009, S. 70)

Einen endlichen Automaten, der keine Ausgabe tätigt und nur prüft, ob am Ende der Eingabe ein Endzustand erreicht ist, nennt man Akzeptor. Akzeptoren werden bei vielen Computerprogrammen verwendet, sie weisen unzulässige Eingaben zurück und akzeptieren nur zulässige.
Einfache Akzeptoren kann jeder Programmierer selbst schreiben, um sein Programm benutzersicher zu machen, z. B. könnte mit einem Akzeptor überprüft werden, ob eine Eingabe eine natürliche Zahl zwischen 0 und 7 ist.
Der Compiler einer Programmiersprache ist ein komplexerer Akzeptor, da er auch die Einhaltung der Syntaxregeln prüfen muss. Akzeptoren, die eingegebene Texte untersuchen, nennt man Parser.

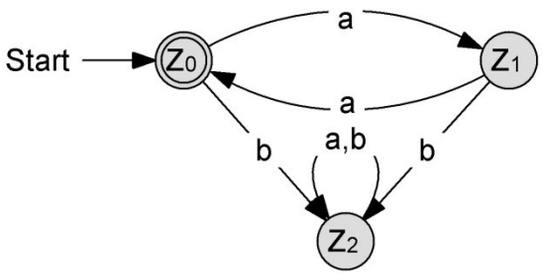
Aufgaben:

1. Erweitere den Getränkeautomaten so, dass er auch zwei Eurostücke akzeptiert. Die Ausgabe besteht dann aus dem gewünschten Getränk und einem 50-Cent-Stück. Vorausgesetzt sei, dass der Automat immer über ein passendes 50-Cent-Stück verfügt.

2. Eine elektrische Zahnbürste lässt sich als Automat mit genau zwei Zuständen darstellen: *bereit* und *läuft* (oder *an* und *aus*). Die auslösende Aktion ist der An-/Ausknopf, die ausgelöste Aktion das Laufen bzw. Stoppen des Motors.

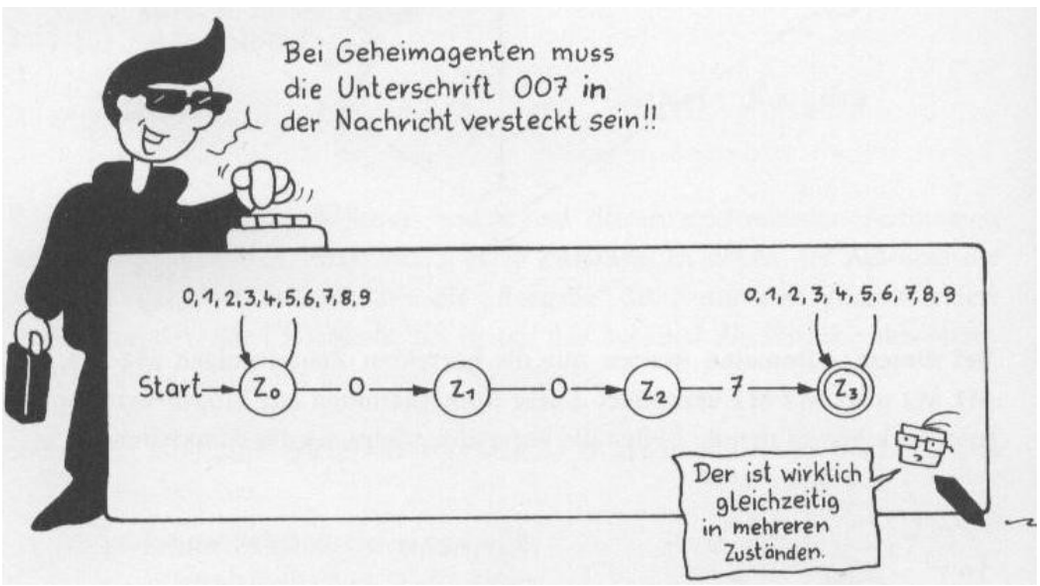
   a. Zeichne das Zustandsdiagramm.
   b. Gib die zugehörige Zustandsübergangstabelle an.
   c. Ergänze das Zustandsdiagramm um einen dritten Zustand *lädt*, in dem sich der Automat befindet, wenn der Akku der Zahnbürste in der Ladestation aufgeladen wird. Welche auslösende Aktion führt zu diesem Zustand?

3. Σ = {a, b} sei das Alphabet eines Akzeptors.



a. Betrachte das Zustandsdiagramm. Welche Eingaben werden vom Automaten akzeptiert? Notiere deine Vermutung.
b. Überprüfe nun die folgenden Eingaben: aba, aaab, a, b, baba, aa, bb, bbbbb, aaaaa
c. Notiere die zugehörige Zustandsübergangstabelle.

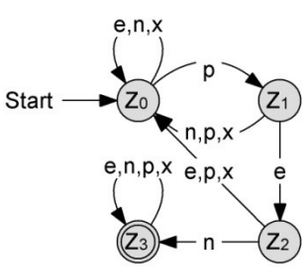
Nichtdeterministische endliche Automaten



(Magenheim, 2009, S. 72)

Beispiel: In einem Text soll die Zeichenkette „pen“ gesucht werden.

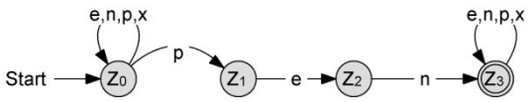
1. Idee



x steht hier für beliebige andere Buchstaben.

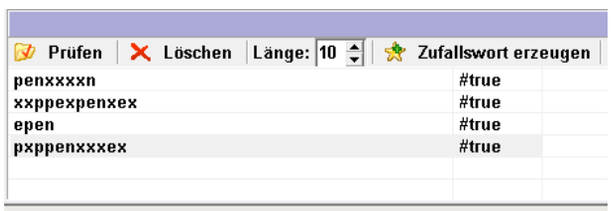
Der Automat arbeitet einwandfrei für folgende Wörter: Pentagon, Doppelpendel, Epen. Aber bei Pappenstiel findet er „pen“ nicht, da er beim ersten P in den Zustand Z_1 wechselt, dann kommt ein zweites p und er kehrt in den Ausgangszustand zurück. Dies ist ein fehlerhafter Akzeptor.

2. Idee



wesentlicher Unterschied: Vom Zustand Z_0 gehen zwei Pfeile aus, die mit dem gleichen Eingabezeichen beschriftet sind.

Es ist also kein deterministischer Automat, d. h. es handelt sich um einen nichtdeterministischen Automaten (NEA). Die Prüfung der Eingabe ist bei diesem Automaten dann positiv, wenn er eine mögliche Zustandsfolge durchlaufen kann, die im Endzustand endet. Für unsere Eingaben liefert der Automat folgende Ergebnisse:



penxxxxp = Pentagon
xxppexpenxex = Doppelpendel
epen = Epen
pxppenxxxex = Pappenstiel

Definition:
Ein nichtdeterministischer endlicher Automat (NEA, engl. NFA) ist bis auf zwei Unterschiede genauso definiert wie ein DEA. Erstens lassen wir eine Menge von Anfangszuständen zu und zweitens ordnet die Übergangsfunktion jeder Eingabe und jedem Zustand eine Menge von möglichen Folgezuständen zu.

Überführung eines NEA in einen DEA

Jeder nichtdeterministische Automat kann in einen deterministischen Automaten überführt werden. Grundsätzlich sind also NEAs nicht leistungsfähiger als DEAs, allerdings fällt es oft leichter zunächst einen NEA zu konstruieren, der dann in einen DEA überführt werden kann.

Verfahren zur Überführung eines NEA in einen DEA:

1. Wir brauchen neue Zustände. Dafür bilden wir Mengen von Zuständen (Teilmengen der Zustandsmenge, max. also die Potenzmenge der Zustandsmenge). Für unser Beispiel des pen-Akzeptors bedeutet dies:
   - Wir haben den Anfangszustand Z_0.
   - Liest der Automat im Zustand Z_0 ein p, so kann er im Zustand Z_0 bleiben oder in den Zustand Z_1 wechseln. Diesen Zustand nennen wir also Z_0Z_1. Es gilt: $Z_0Z_1 = \{ Z_0, Z_1 \}$
   - Liest der Automat im Zustand Z_0 ein anderes Zeichen, so bleibt er im Zustand Z_0.
   - Liest der Automat im Zustand Z_0Z_1 ein p, so betrachten wir die beiden Teilanfangszustände Z_0 und Z_1 getrennt. Vom Zustand Z_0 kommt er in den Zustand Z_0Z_1 (siehe oben) und vom Zustand Z_1 geht es nicht weiter, also bleibt er im Zustand Z_0Z_1.
   - Liest der Automat im Zustand Z_0Z_1 ein e, so ergibt sich aus dem Anfangszustand Z_0, dass der Automat in Z_0 bleibt. Aus dem Anfangszustand Z_1 ergibt sich, dass er in den Zustand Z_2 wechselt. Diesen Zustand nennen wir also Z_0Z_2. Es gilt: $Z_0Z_2 = \{ Z_0, Z_2 \}$
   - Liest der Automat im Zustand Z_0Z_1 ein n, so ergibt sich aus dem Anfangszustand Z_0, dass der Automat in Z_0 bleibt. Aus dem Anfangszustand Z_1 ergibt sich, dass er in den Zustand Z_0 wechselt.
   - Liest der Automat im Zustand Z_0Z_1 ein beliebiges anderes Zeichen, so ergibt sich aus dem Anfangszustand Z_0, dass der Automat in Z_0 bleibt. Aus dem Anfangszustand Z_1 ergibt sich, dass er in den Zustand Z_0 wechselt.
   - ...

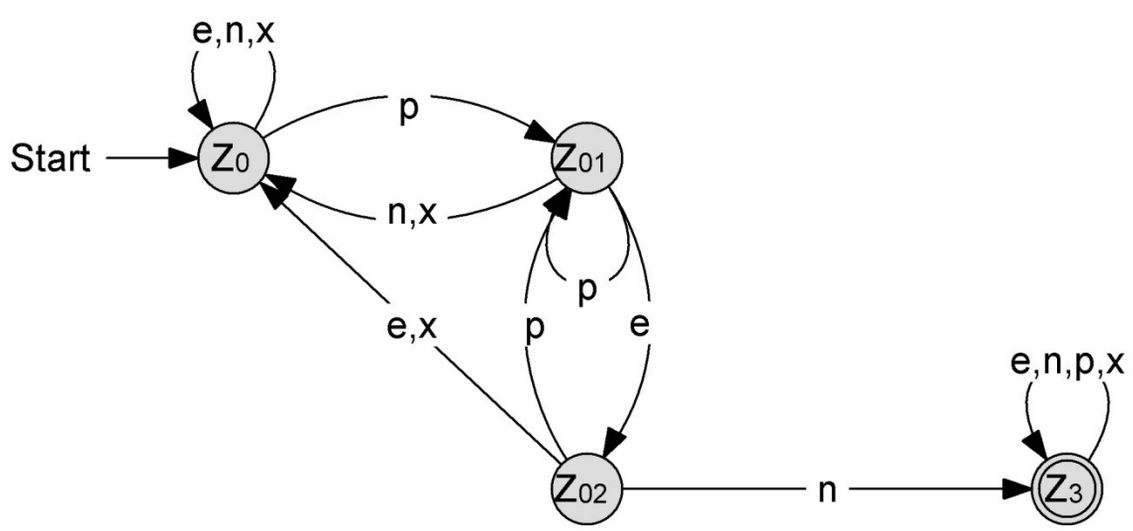
- Alle Mengenzustände, die den Endzustand Z_3 enthalten, können zu einem Mengenendzustand zusammengefasst werden. Die Übergangstafel (Zustandstabelle) sieht dann wie folgt aus:

	p	e	n	x
Z_0	Z_oZ_1	Z_0	Z_0	Z_0
Z_oZ_1	Z_oZ_1	Z_0Z_2	Z_0	Z_0
Z_0Z_2	Z_oZ_1	Z_0	$Z_0Z_3 = Z_3$	Z_0
$Z_0Z_3 = Z_3$	$Z_0Z_1Z_3 = Z_3$	$Z_0Z_3 = Z_3$	$Z_0Z_3 = Z_3$	$Z_0Z_3 = Z_3$
$Z_0Z_1Z_3 = Z_3$	$Z_0Z_1Z_3 = Z_3$	$Z_0Z_2Z_3 = Z_3$	$Z_0Z_3 = Z_3$	$Z_0Z_3 = Z_3$
$Z_0Z_2Z_3 = Z_3$	$Z_0Z_1Z_3 = Z_3$	$Z_0Z_3 = Z_3$	$Z_0Z_3 = Z_3$	$Z_0Z_3 = Z_3$

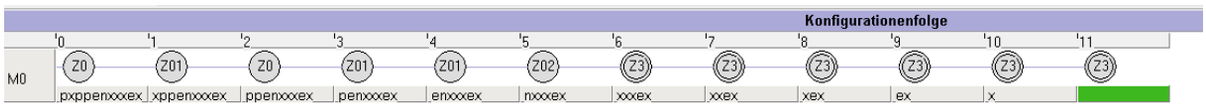
Also kurz gefasst:

	p	e	n	x
Z_0	Z_oZ_1	Z_0	Z_0	Z_0
Z_oZ_1	Z_oZ_1	Z_0Z_2	Z_0	Z_0
Z_0Z_2	Z_oZ_1	Z_0	Z_3	Z_0
Z_3	Z_3	Z_3	Z_3	Z_3

2. DEA konstruieren:

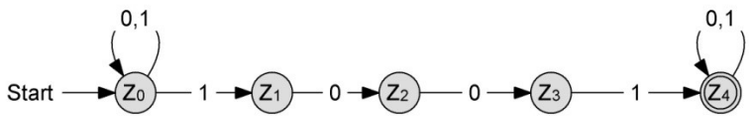


Bei der Simulation von Pappenstiel ergibt sich folgender Durchlauf:



Aufgaben

1. Gegeben ist ein NEA durch das Zustandsdiagramm:



a) Prüfe, ob 10110010 akzeptiert wird. Beschreibe das Durchlaufen des Graphen.
b) Überführe den NEA in einen DEA.

2. Gegeben sei die Übergangstabelle.

	0	1
Z_0	Z_0 oder Z_1	Z_0
Z_1	Z_2	-
Z_2	Z_3	Z_3
Z_3	Z_4	-
Z_4*	Z_4	Z_4

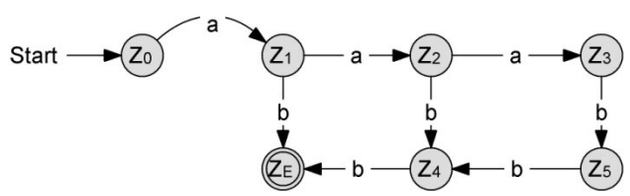
a. Zeichne den Zustandsgraphen und erstelle in einer Simulationsumgebung eine entsprechende Übergangstafel. Teste!

b. Charakterisiere die akzeptierten Worte.

c. Wird 11001101000 bzw. 110011001000 akzeptiert? Beschreibe das Durchlaufen des Graphen.

Grenzen endlicher Automaten

Beispiel:
Konstruiere einen Automaten (in Form eines Zustandsgraphen), der die Sprache $\{a^n b^n \mid 1 \le n \le 3\}$ erkennt. D. h. Es handelt sich um die Sprache aus den a-b-Wörtern, die mit ein bis drei a beginnt und mit genauso vielen b endet.
Lösung:



Wie sieht der Automat für $\{a^n b^n \mid 1 \le n \le 7\}$ aus?

Wann gibt es Probleme für $\{a^n b^n \mid 1 \le n \le N\}$ und warum?
Wir brauchen eine endliche Menge von Zuständen (laut Definition), d. h. z sei die Anzahl der Zustände.

Um $a^N b^N$ zu erkennen, muss der Automat sich merken, dass er a N-mal gelesen hat, dazu benötigt er N Zustände. Entsprechend benötigt er auch für b N Zustände.

Also muss gelten: $z = 2 \cdot N \Longrightarrow N = z \text{ div } 2$

Für N unbegrenzt bräuchten wir also unbegrenzt viele Zustände, dann wäre der Automat nicht mehr endlich. Für solche Probleme benötigt man einen Automaten mit „Gedächtnis“, einen Kellerautomaten.

Grammatiken

Natürliche und formale Sprachen

Seit über 50 Jahren gibt es bereits Übersetzungsprogramme, die das Übersetzen von Texten jeder Art aus einer natürlichen Sprache in eine andere natürliche Sprache ermöglichen sollte. Allerdings waren die ersten Programme dieser Art nur bedingt geeignet, so dass z. B. „Ich weiß nicht“ mit „I white not.“ übersetzt wurde.

In den frühen Jahren der Übersetzungsprogramme arbeitete man vorrangig mit einem zweisprachigen Wörterbuch und eine oberflächlichen Syntaxbetrachtung. Die Ergebnisse zeigten, dass der Wortschatz alleine nicht reicht. Bessere Übersetzungsergebnisse bekommt man, wenn man zusätzlich auch die Analyse der Wortstrukturen und der Semantik (Wortbedeutung im Kontext) hinzuzieht.

Bei natürlichen Sprachen gibt es aber dennoch das Problem der Zweideutigkeit, welches zu zusätzlichen Schwierigkeiten führt, denn wenn der Metzgermeister zu seinem Gesellen sagt: „Hol doch bitte mal den Bullen, der ist jetzt dran.“ Dann will der Metzgermeister hoffentlich nicht einen Polizisten schlachten.

Konsequenz für formale Sprachen:

Für formale Sprachen versucht man dieses Zweideutigkeit zu vermeiden und so eine eindeutige Semantik zu erstellen, d. h. es soll zu jedem Sprachelement eine eindeutig Zuordnung der Bedeutung gibt.

Definition:
Eine formale Sprache ist eine Menge von zulässigen Zeichenketten über ein Alphabet Σ, welches eine endliche Menge einzelner Zeichen ist. Ein Wort (auch Zeichenkette) wird durch Aneinanderreihen von mehreren Zeichen des Alphabetes gebildet. Die Regeln zur Bildung zulässiger Zeichenketten bezeichnet man als Syntax der Sprache, die Bedeutung eines Wortes als Semantik.[30]

Beispiele für formale Sprachen:

1. Einstellungen an einem Backofen
   An einem Backofen kann man die Zeit einstellen, wie lange der Ofen heizen soll (|↦|) und um wie viel Uhr der Ofen sich ausschalten soll (→|). Zusätzlich kann die Temperatur eingestellt werden die Heizart kann über einen Drehknopf gewählt werden.
   Es ergibt sich folgendes Alphabet:

[30] Vgl. (Magenheim, 2009, S. 102f)

Σ = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ~, Y, ♈, =, |↦|, →| }

Die Syntax eines korrekten Wortes (einer richtigen Bedienungsangabe) wird so gebildet, dass zunächst die Heizdauer bzw. die Ausschaltzeit (durch das Symbol gekennzeichnet) mit max. vier Ziffern angegeben wird, gefolgt von der Heizart durch eins der vier Symbole dargestellt und der Temperaturangabe mit Hilfe von drei Ziffern, die zwischen 000 und 299 liegen müssen.

Mögliche Wörter:

a. `|↦|115 →|1230 ♈160‘
b. `|↦|~999999 Y‘
c. `|↦|215 →|817 = 215‘
d. `|↦|876 →|9897 ~ 234‘

zu a: 1 h 15 min lang wird bis 12.30 Uhr bei 160° C mit Umluftgrill geheizt.

zu b: Es handelt sich nicht um ein syntaktisch korrektes Wort.

zu c: 2 h 15 min lang wird bis 8.17 Uhr mit Ober-/Unterhitze bei 215° C geheizt.

zu d: Das Wort ist syntaktisch korrekt, ergibt jedoch semantisch keinen Sinn, da 876 und 9897 beides keine sinnvollen Zeitangaben sind.

Überprüfe die folgenden Wörter auf semantische und syntaktische Korrektheit:

i. `|↦|75 →|2100 ~ 200‘
ii. `|↦|100 Y 220 ♈2115‘
iii. `75 →|2620 ~ 230‘
iv. `|↦|1115 →|90 ~ 240‘

2. Alle Palindrome[31] aus a-b-Wörtern mit ungerader Buchstabenanzahl.

Σ = { a, b }

Regel: Wenn vorne ein Buchstabe steht, muss hinten der gleiche Buchstabe stehen. Diese Regel muss auch erfüllt sein, wenn jeweils vorne und hinten ein Buchstabe weggenommen wird. Die Buchstabenanzahl muss ungerade sein.

Mögliche Wörter: a, b, aab, aba, abba, …

aab und abba sind keine syntaktisch korrekten Wörter, alle anderen gehören zur Sprache.

Notationsmöglichkeiten für die Regeln formaler Sprachen

1. Möglichkeit: Produktionsregeln

Man benötigt zunächst immer genau ein Startsymbol S. Ausgehend von dieser Startvariablen S entstehen durch sukzessives Anwenden der Produktionsregeln syntaktisch korrekte Wörter.

S ⟶ a
S ⟶ b
S ⟶ aSa
S ⟶ bSb

syntaktische Variable und Term, der die syntaktische Variable ersetzen kann

Ein Term kann Zeichen des Alphabets oder syntaktische Variablen enthalten. Wenn eine syntaktische Variable verschiedene Möglichkeiten der Anwendung hat, kann man diese Regeln zusammenfassen, indem man das Zeichen (|) verwendet, welches als alternatives Oder interpretiert wird.

[31] Palindrome lassen sich von vorne und hinten lesen, z. B. OTTO oder ANNA

Also: $S \longrightarrow a \mid b \mid aSa \mid bSb$

Zeichen des Alphabets nennt man Terminale.

Kommt man durch eine Folge von Regelanwendungen zu einem Wort, welches nur aus Terminalen besteht, so bezeichnet man diese Folge als Ableitung.

Beispiele für Ableitungen:

$S \longrightarrow aSa \longrightarrow abSba \longrightarrow abbba$

$S \longrightarrow a$

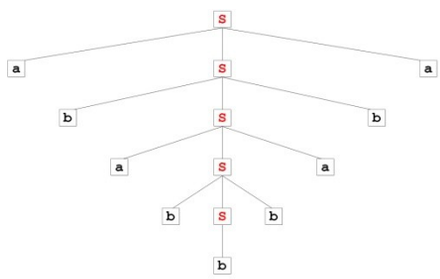
$S \longrightarrow bSb \longrightarrow bbSbb \longrightarrow bbabb$

$S \longrightarrow bSb$ (unvollständige Ableitung)

Alle Wörter, die man ableiten kann, gehören zur entsprechenden Sprache.

Der Ableitungsbaum als übersichtliche Darstellung

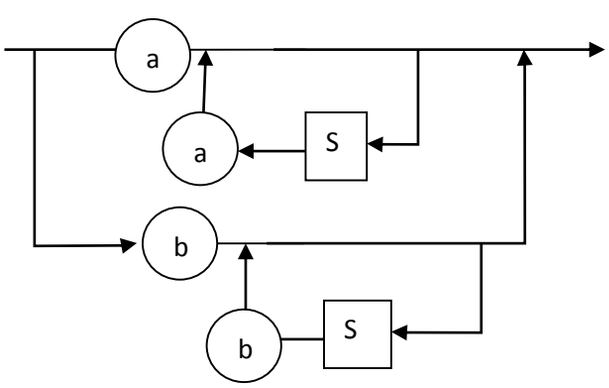
Die Startvariable dient als Wurzel. An den Kindknoten notiert man der Reihe nach die möglichen Terme aus Terminalen und Variablen. An den Blättern stehen schließlich die Terminale des erzeugten Wortes.



2. Möglichkeit: Syntaxdiagramme

Die Regeln der Syntax werden hier in einem Graphen dargestellt. Die Variablen werden als rechteckige Knoten und die Terminale als kreisförmige Knoten dargestellt. Anhand der gerichteten Kanten kann man die Regeln der Syntax ablesen.

Alternativen werden durch Verzweigungen kenntlich gemacht. Kann etwas nur optional eingefügt werden, muss eine alternative Leerkante hinzugefügt werden. Wiederholungen werden durch Rückwärtskanten (Rekursion) dargestellt.



3. Möglichkeit: (erweitere) Backus-Naur-Form (EBNF)

Aufgestellt werden hier auch wieder Produktionsregeln, allerdings ist die Schreibweise etwas verkürzt. Der Produktionspfeil wird durch „=“ ersetzt. Die spitzen Klammern bei den Variablen entfallen und die Terminale müssen in Hochkommata gesetzt werden. Jede Regel endet mit einem Semikolon. Wiederholungen werden in geschweiften Klammern notiert.

S = ‘a’ | ‘b’ | ‘aSa’ | ‘bSb’;

Aufgaben: (Hubwieser & a., 2010, S. 20f)

1. Erstelle ein Syntaxdiagramm für Telefonnummern (Festnetz) für Bad Wildungen
   a. für Ortsgespräche
   b. für Ferngespräche
   c. für Auslandsgespräche

2. Gegeben ist folgende EBNF für die Erzeugung von Smileys.

   Smiley: S = H A N M | A N M | H A M | A M;
   Hut: H = '[' | 'O' | '<';
   Nase: N = '-' | '*' | 'o';
   Augen: A = ':' | ';' | '%' | '8';
   Mund: M = ')' | '(' | '/';

   a. Gib die Menge der Terminale (das Alphabet Σ) und die Menge V der Variablen an.
   b. Gib je eine Ableitung für die Smileys '[;-)' und '<:o)' an.
   c. Wie viele verschiedene Smileys lassen sich insgesamt mit dieser EBNF erzeugen)

Die Grammatik

Noam Chomsky beschäftigte sich als Linguistiker bereits in den 50er Jahren des 20. Jahrhunderts mit der Beschreibung natürlicher Sprachen mit Hilfe einer mathematischen Theorie, dabei definierte er den Begriff Grammatik und kam zu der Erkenntnis, dass man die Grammatiken je nach ihren Produktionsregeln in vier verschiedene Kategorien einteilen kann. Welche Bedeutung diese Erkenntnis für die theoretische Informatik bekommen sollte, konnte er damals nicht ahnen.

Definition: (nach Chomsky, 1959)
Eine Grammatik besteht aus vier Komponenten:

1. einer endlichen Menge von Terminalen, dem Alphabet Σ
2. einer endlichen Menge V von Variablen (auch Nichtterminale genannt) mit $V \cap \Sigma = \{\}$
3. einem Startsymbol $S \in V$
4. einer endlichen Menge P von Produktionsregeln, welche festgelegen, wie man aus bekannten syntaktischen Variablen neue Terme ableiten kann.

Eine Grammatik beschreibt eine Sprache.

Die Chomsky-Hierarchie

Typ 0 Es gibt keine Einschränkungen bezüglich der Produktionsregeln.

Typ 1 kontextsensitiv
Für alle Produktionsregeln gilt: Die Anzahl der Terminale und Variablen auf der linken Seite einer Produktionsregel ist nicht größer als die Anzahl der Terminale und Variablen auf der rechten Seite, d. h. das neue Wort darf nicht kürzer werden.

Typ 2 kontextfrei
Die Grammatik muss vom Typ 1 sein, und es muss für alle Produktionsregeln gelten: Auf der linken Seite darf nur eine einzelne Variable stehen.

Typ 3 regulär

Die Grammatik muss vom Typ 2 sein, und es muss für alle Produktionsregeln gelten: Auf der rechten Seite steht entweder ein einzelnes Terminalsymbol oder ein Terminalsymbol gefolgt von einer Variablen (bzw. eine Variable gefolgt von einem Terminal; rechtslinear bzw. linkslinear).

Reguläre Sprachen

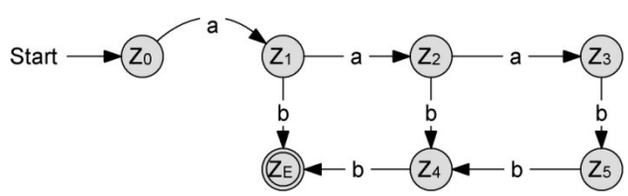
Besonders einfach strukturierte Sprachen sind demnach die regulären Sprachen. In der Informatik sind sie dadurch gekennzeichnet, dass sie von endlichen Automaten erkannt werden können. Zu jedem endlichen Automaten lässt sich eine Grammatik angeben, die genau die Sprache erzeugt, die der endliche Automat (Akzeptor) erkennt.

Konstruktion der zugehörigen Grammatik bei vorgegebenen endlichen Automaten:

1. Die Zustandsmenge Q wird zur Variablenmenge V
2. Das Eingabealphabet Σ_1 entspricht dem Alphabet Σ
3. Der Anfangszustand $Z_0 \in Q$ wird zum Startsymbol S
4. Die Übergangsregel δ (Z_i; x) = Z_j für i,j $\in \mathbb{N}$ Z_i, $Z_j \in$ Q, x $\in \Sigma$ liefert die Produktionsregel:
   <Z_i> = x<Z_j>
5. Für jeden Endzustand Z_E wird zunächst eine Produktionsregeln festgelegt, bei deren Anwendung <Z_E> durch das leere Wort ε (kein Zeichen) ersetzt wird: <Z_E> = ε
   Zwei Produktionsregeln <Z_i> = ‘a’<Z_E> und <Z_E> = ε können dann durch: <Z_i> = ‘a’ ersetzt werden, wenn <Z_E> bei keiner weiteren Produktion auf der linken Seite vorkommt.

Beispiel:

Der bereits bekannt Automat: {$a^n b^n$ | 1 ≤ n ≤ 3}



Hieraus ergibt sich folgende Grammatik

G = {Variablenmenge V, Alphabet, Produktionsregeln, Anfangssymbol}

G = ({Z_0,Z_1,Z_2,Z_3,Z_4,Z_5}, {a,b}, P, Z_0)

P = { Z_0 -> aZ_1
Z_1 -> aZ_2 | b
Z_2 -> aZ_3 | bZ_4
Z_3 -> bZ_5
Z_4 -> b
Z_5 -> bZ_4 }

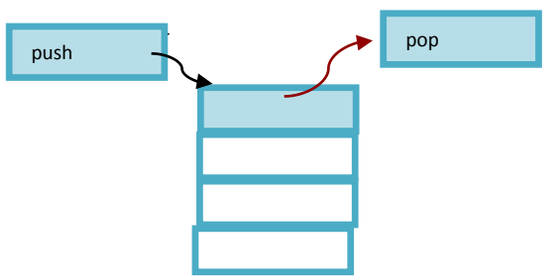
Kontextfreie Sprachen

Kontextfreie Sprachen können von Kellerautomaten erkannt werden.

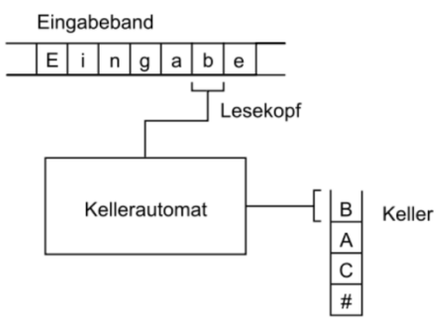
Beispiel:

Betrachten wir die nichtreguläre Sprache $\{a^n b^n \mid n \in N\}$. Im zweiten Teil hatten wir festgestellt, dass ein endlicher Automat nicht ausreicht, um die Sprache zu erkennen. Der entsprechende Automat müsste ein „Gedächtnis“ besitzen. Ein solcher Automat heißt Kellerautomat.

Das „Gedächtnis“ wird durch einen Keller erzeugt. Ein Keller (auch stack genannt) ist eine Datenstruktur, die nach dem LIFO-Prinzip funktioniert, d. h. Last-In-First-Out. Man kann sich die Datenstruktur wie einen Stapel von Büchern vorstellen. Man kann ein Buch oben auf den Stapel legen oder das oberste Buch vom Stapel herunternehmen. Der Stapel kann natürlich auch leer sein.



Der Kellerautomat besitzt also zusätzlich diesen Keller, der mit verschiedenen Zeichen aus dem Kelleralphabet Γ gefüllt sein kann. Die Übergangsfunktion betrachtet nicht nur den aktuellen Zustand und das eingelesene Zeichen sondern auch, welches Zeichen momentan beim Keller ganz oben liegt.

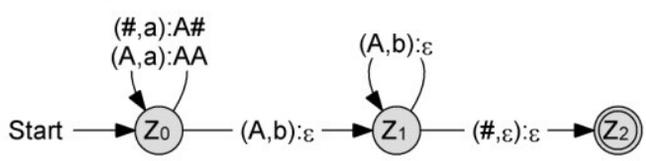


(wikipedia.org, 2014)

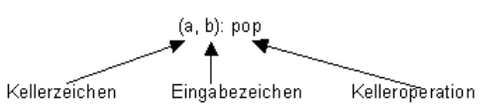
Für unser Sprachbeispiel ergibt sich folgender Kellerautomat:

$$\Sigma = \{a, b,\}, Q = \{Z_0, Z_1, Z_2\}, \Gamma = \{\#, A\}, \delta, Z_0, \#, Z_E = Z_2$$

mit folgendem Zustandsdiagramm



wobei:

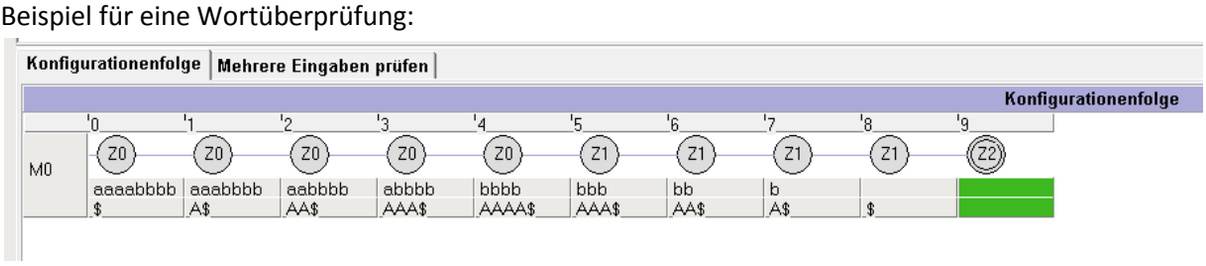


Überführungstabelle:

Zustand	Eingabe	Kellerzeichen	neuer Zustand	Operation	
Z_0	a	#	Z_0	push	A#
Z_0	a	A	Z_0	push	AA
Z_0	b	A	Z_1	pop	ε
Z_1	b	A	Z_1	pop	ε
Z_1	ε	#	Z_2	nop	#

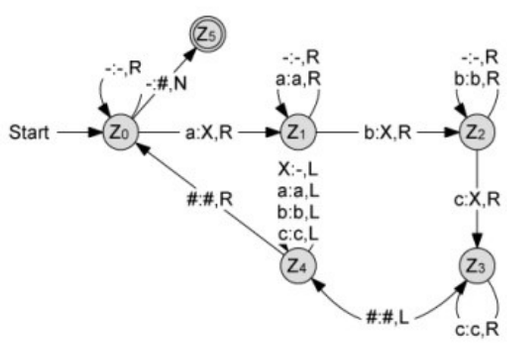
ist ein Vorbelegungszeichen.

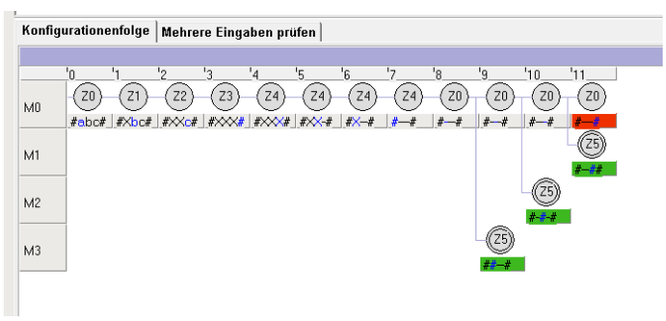
Beispiel für eine Wortüberprüfung:



Sprachen vom Typ 0

Jede Sprache kann von einer Turingmaschine erkannt werden.





Prolog und Künstliche Intelligenz

Prolog (Programming in Logic)

Konzipiert und erstmalig implementiert wurde Prolog 1972 von einer Gruppe von Wissenschaftler um Alain Colmerauer an der Universität in Marseille. Prolog gehört zu den deklarativen Programmiersprachen und basiert auf den Prinzipien der logischen Programmierung. Dies bedeutet, dass die Frage, was zu lösen ist, geklärt werden soll, während bei prozeduralen Sprachen gefragt wird, wie etwas zu lösen ist.

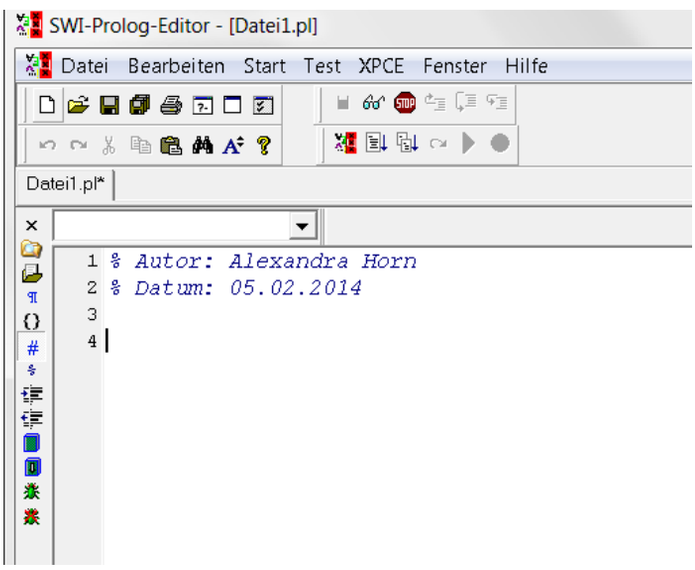
SWI-Prolog-Editor

Hr. Röhner hat einen kostenlosen Editor entwickelt, der auf die Bedürfnisse von Schulen abgestimmt ist. Um diesen Editor nutzen zu können, muss man zunächst SWI-Prolog installieren. Die notwendigen Programme und Hilfedateien findet man auf folgender Seite: http://www.swi-prolog.org/download/stable. Dabei handelt es sich um ein freies und professionelles Prolog-System, welches seit 1987 an der Universität von Amsterdam entwickelt und gepflegt wird. Um den Editor anschließend nutzen zu können, muss die 32-Bit-Version von SWI-Prolog verwendet werden.

Anschließend muss noch der SWI-Prolog-Editor installiert werden und die Konfiguration entsprechend eingestellt werden.

Im Editor können nun ganz komfortabel die Programme geschrieben werden.

Prologprogramme erhalten als Dateiendung „pl“.



Grundelemente

Jedes Prologprogramm besteht aus Fakten, Regeln und Anfragen. Zunächst werden im Programm Fakten und Regel festgehalten, bevor dann Anfragen an das Programm gestellt werden können. Ein Faktum beschreibt dabei eine Eigenschaft eines Objektes oder eine Beziehung zwischen mehreren Objekten. Eine Regel stellt eine Wenn-Dann-Beziehung auf, so dass aus bestehenden Fakten neue Fakten logisch gewonnen werden können.

Kommentare

Kommentare können zwischen „/*“ und „*/“ geschrieben werden.

Fakten:

Um in dem Programm bekannte Informationen einzugeben, werden Fakten verwendet. Sie bestehen aus einem Relationennamen (genannt: **Funktor**) und einem oder mehreren **Argumenten**. Jeder Fakt muss mit einem Punkt abschließen.

Beispiele:

weiblich(erna).	umgangssprachlich: Erna ist weiblich. Funktor: weiblich Argument: erna einstelliges Faktum
elternVon(erna, fritz, karla).	umgangssprachlich: Erna und Fritz sind die Eltern von Karla. Funktor: elternVon Argument: erna, fritz und karla mehrstelliges Faktum

Die Fakten sind in verschiedene **Prädikate** unterteilt. Im Beispiel in die Prädikate weiblich/1 und elternVon/3.

Variablen und Konstanten:

Variablennamen werden groß geschrieben.
Konstanten werden immer klein geschrieben.
Eine anonyme Variable, d. h. deren Name und Inhalt nicht benötigt wird, wird durch einen Unterstrich dargestellt.

Verknüpfungsmöglichkeiten:

Konjunktion (und-Verknüpfung)

Das Komma entspricht dem logischen „Und“, d. h. alle durch ein logisches „Und“ verknüpfte Aussagen müssen übereinstimmen, damit das Ganze stimmt.

Beispiel:

Die Anfrage, ob Max und Erna dieselbe Mutter haben, lässt sich wie folgt ausdrücken:
?-elternVon(Mutter, _ ,erna), elternVon(Mutter, _ ,max).

Disjunktion (oder-Verknüpfung)

Ein Semikolon entspricht dem logischen „Oder“, d. h. wenn nur eine der Teilaussagen richtig ist, stimmt das Ganze.

Das allererste Prologprogramm


```
1  % Autor: Alexandra Horn
2  % Datum: 05.02.2014
3
4  %maennliche-Fakten       KOMMENTAR
5  maennlich(werner).
6  maennlich(paul).
7  maennlich(gerd).
8
9  %weibliche-Fakten
10 weiblich(maja).
11 weiblich(sonja).
12 weiblich(anna).
13
14 %vater(Vater,Kind)-Fakten
15 vater(werner,maja).
16 vater(werner,paul).
17 vater(paul,gerd).
18
19 %mutter(Mutter,Kind)-Fakten
20 mutter(sonja,maja).
21 mutter(sonja,paul).
22 mutter(maja,anna).
```

Der Funktor muss klein geschrieben werden.

werner ist eine Konstante.

Kind ist eine Variable.

Bevor nun Anfragen an das Prologprogramm gestellt werden können, muss es konsultiert werden. Dadurch wird das Programm an SWI-Prolog übergeben. Wenn das Programm fehlerfrei ist, erscheint im SWI-Prolog-Fenster (untere Teil des Editors) folgende Meldung:

```
1 ?- consult('C:/Users/Alja/Documents/Schule/Informatik/Q4 Wahlthemen/Künstliche Intelligenz/Prolog/allererstesPrologprogramm.pl').
% C:/Users/Alja/Documents/Schule/Informatik/Q4 Wahlthemen/Künstliche Intelligenz/Prolog/allererstesPrologprogramm.pl compiled 0.02 sec, 1,944 bytes
true.
```

Nun können im SWI-Prolog-Fenster Anfragen gestellt werden, dabei kann das Programm nur Fragen beantworten, zu dem Informationen vorliegen. Gibt es keine korrekte Information zur Anfrage, so gibt das Programm „false“ aus.

```
4 ?- maennlich(werner).
true.
```

Kommen in der Anfrage nur Konstanten vor, so antwortet das Programm mit „true“ oder „false“.

```
5 ?- maennlich(maja).
false.
```

Im Faktum ist festgelegt, dass werner männlich ist, maja aber nicht.

```
6 ?- maennlich(franz).
false.
```

Über franz gibt es keine Auskunft, daher wird auch hier „false“ ausgegeben. Die Aussage ist nicht beweisbar.

```
7 ?- mutter(sonja,Kind).
Kind = maja ;
Kind = paul.
```

Kommen in der Anfrage Variablen vor, so sucht Prolog nach Werten, die die Anfrage erfüllen. Durch Drücken der Enter-Taste werden weitere Alternativen angezeigt.

Wissensbasis und Anfragen

Alle Fakten und Regeln zusammengefasst ergeben die Wissensbasis des Prolog-Interpreters. Sobald das Programm consultiert worden ist, können Anfragen an diese Wissensbasis gestellt werden. Die Regeln

und Fakten fasst man unter dem Oberbegriff **Klauseln** zusammen. Man kann Anfragen mit Konstanten stellen, dann erhält man als Antwort *True* oder *False*, wobei man dabei beachten muss, dass *False* bedeutet, dass es aufgrund der Wissensbasis nicht beweisbar ist (siehe Beispiel). Kommen in der Anfrage Variablen vor, so sucht der Prolog-Interpreter nach Variablenwerten, die die Anfrage erfüllen. Beispiele:

?- weiblich(erna).	True.	umgangssprachlich: Ist Erna weiblich?
?- weiblich(thea).	False.	(nicht beweisbar)
?-weiblich(Frau).	Frau = erna; *alle weiteren bekannten weiblichen Personen werden ausgegeben* (*Return drücken)*	

?-elternVon(M,V,karla).
M = erna, V = fritz

Wichtig!
Die Reihenfolge der Klauseln steuert das Suchverhalten des Prolog-Interpreters. Der Prolog-interpreter durchforstet beim Beweisversuch einer Anfrage die Wissensbasis nach der Strategie der Tiefensuche von oben nach unten und innerhalb eines Regelrumpfes von links nach rechts. (siehe Backtracking weiter hinten)

Aufgaben:

1. Öffne die **Datei strauss.pl** im SWI-Prolog-Editor.

```
% Autor: Alexandra Horn
% Datum: 20.03.2012
% Grundlagen – Blumenstrauß

%Fakten
%farbe: Prädikat = Eigenschaft

rot(rose).
gelb(tulpe).
weiss(nelke).
blau(vergissmeinnicht).
blau(veilchen).
```

2. Consultiere diese Datei und stelle folgende Anfragen an die Wissensbasis. Notiere die Ausgaben.
   a. ?- rot(rose).
   b. ?- gelb(veilchen).
   c. ?-gelb(primel).
   d. ?-lila(veilchen).
   e. ?- rot(X).
   f. ?- rot(Blume).

3. Welche Anfrage musst du stellen, wenn du den Namen aller blauen Blumen erhalten willst?

4. Öffne die **Datei fruehst.pl** im SWI-Prolog-Editor.

```
% Autor: Alexandra Horn
% Grundlagen
% Frühstücksgewohnheiten

mag(papa,muesli).
mag(papa,brot).
mag(mama, kuchen).
mag(mama, brot).
mag(oma, brot).
mag(max, muesli).
mag(max, kuchen).

hasst(papa, kuchen).
hasst(papa, muesli).
hasst(oma, muesli).
hasst(oma, kuchen).
hasst(max, brot).
```

5. Consultiere diese Datei und stelle folgende Anfragen an die Wissensbasis. Notiere die Anfragen und Ausgaben.
   a. Mag Papa Kuchen?
   b. Wer hasst Müsli?
   c. Was mag Oma?
   d. Wer mag was?
   e. Wer hasst Kuchen und mag Müsli?
   f. Wer mag Kuchen und Brot?
   g. Wer mag Brot oder Kuchen?

h. Was mögen sowohl Papa als auch Mama?
i. Wer mag Kuchen und hasst Müsli?

6. Übersetze die folgenden Sätze in eine Prolog-Datenbasis. Nenne die Datei **liebe.pl**. (D. h. schreibe ein Prologprogramm, welches die folgenden Informationen enthält).

   Peter liebt Susi.
   Hans liebt Susi und Sabine.
   Sabine liebt Peter und hasst Hans.
   Susi liebt Peter und Felix.
   Susi hasst Sabine.
   Peter hasst Felix.
   Felix liebt sich selbst.

   Stelle nun die folgenden Anfragen:
   a) Wen liebt Sabine?
   b) Wer liebt Sabine?
   c) Wer liebt wen?
   d) Wer liebt jemanden, der ihn auch liebt?
   e) Wessen Liebe wird mit Hass gedankt?

Regeln – Wenn-Dann-Beziehungen

Mit einer Regel können aus bekannten Fakten neue Fakten logisch gefolgert werden. Eine Regel besteht aus einem **Regelkopf** (der Folgerung), dem **Prolog-Atom :-** und einem **Regelrumpf** (den Voraussetzungen). Der Regelkopf stellt die logische Schlussfolgerung (Konklusion) dar, welche sich aus dem Regelrumpf ergibt.

Beispiel:
X ist eine Schwester von Y, falls X weiblich ist und X und Y dieselben Eltern haben.
in Prolog:

```
istSchwesterVon(X,Y) :-
        weiblich(X),
        elternVon(M,V,X),
        elternVon(M,V,Y),
        X \== Y.
```

Die verwendeten Variablen beziehen sich lediglich auf die Regel, in der sie auftreten, d. h. sie haben außerhalb der Regel keine Gültigkeit. Manchmal benötigt man den Operator \==, der *nicht identisch* bedeutet, denn sonst wäre z. B. jede Tochter ihre eigene Schwester. Prolog kann nämlich die beiden unterschiedlichen Variablen X und Y mit dem gleichen Wert instanzieren.

Das erste Prologprogramm mit Regeln

Wir erweitern das allererste Prologprogramm um einige Regeln.

```
% Autor: Alexandra Horn
% Datum: 05.02.2014

%maennliche-Fakten      KOMMENTAR
maennlich(werner).
maennlich(paul).
maennlich(gerd).

%weibliche-Fakten
weiblich(maja).
weiblich(sonja).
weiblich(anna).

%vater(Vater,Kind)-Fakten
vater(werner,maja).
vater(werner,paul).
vater(paul,gerd).

%mutter(Mutter,Kind)-Fakten
mutter(sonja,maja).
mutter(sonja,paul).
mutter(maja,anna).

%Regeln
elternteil(Elter,Kind):-vater(Elter,Kind).
elternteil(Elter,Kind):-mutter(Elter,Kind).
sohn(Kind, Elter):- maennlich(Kind),elternteil(Elter,Kind).
grossmutter(Oma,Enkel):- mutter(Oma,Elter),elternteil(Elter,Enkel).
bruder(X,Y):-maennlich(X),elternteil(E,X),elternteil(E,Y), X\==Y.
```

Auch an dieses Programm können wir nun Anfragen stellen.

```
4 ?- elternteil(E, paul).
E = werner ;
E = sonja ;
false.
```

Pauls Eltern sind Werner und Sonja.

```
5 ?- grossmutter(C,E).
O = sonja,
E = anna ;
O = sonja,
E = gerd ;
false.
```

Sonja ist die Oma von Anna und Gerd, denn Sonja ist die Mutter von Maja und Paul. Maja ist die Mutter von Anna und Paul ist der Vater von Gerd.

```
6 ?- bruder(X,Y).
X = paul,
Y = maja ;
X = paul,
Y = maja ;
false.
```

Paul und Maja sind Geschwister. Die Ausgabe erfolgt doppelt, weil einmal über Werner und einmal über Sonja gesucht wird.

```
7 ?- sohn(werner,Sohn).
false.
```

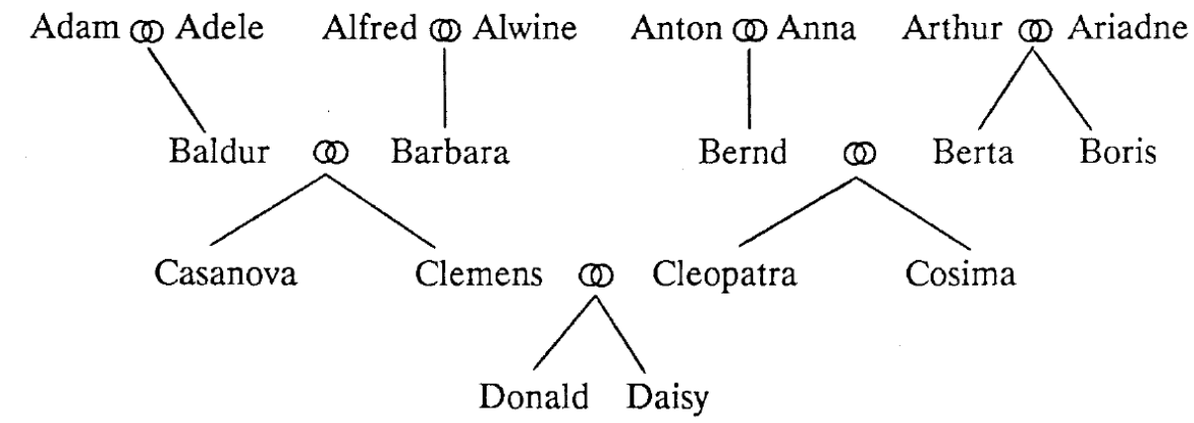
Diese Anfrage liefert kein Ergebnis, weil an erster Stelle der Sohn stehen muss und als zweites Argument der Vater. Werner hat im System jedoch keinen Vater.

```
8 ?- sohn(Sohn,werner).
Sohn = paul ;
false.
```

Nun bekommt man den Sohn von Werner genannt.

Aufgaben:

Dies ist der Stammbaum von Donald und Daisy:



Als Prolog-Datenbasis findest du diesen Stammbaum in der Datei **stammb.pl.**

1. Stelle folgende Anfragen:
   a. Wer sind die Eltern von Daisy?
   b. Mit wem ist Baldur verheiratet?
   c. Wie heißen die Kinder von Adam?
   d. Wie heißt die Mutter von Cosima? Es gibt zwei Möglichkeiten.
   e. Wie heißt der Vater von Daisy? Verwende beide Möglichkeiten.
   f. Wie heißen die Großeltern von Donald?

2. Füge im Programm Regeln hinzu. Schreibe vor jedes Prädikat einen erläuternden Kommentar.
   a. mutter(X,Y) :- elter(X,Y), weibl(Y).
   b. vater(X,Y)…
   c. kind(X,Y)…
   d. schwiegermutter(X,Y) :- verheiratet(X,Z), mutter (Z,Y).
   e. bruder(X,Y)…
   f. schwester(X,Y)…
   g. schwager(X,Y) :- verheiratet(X,Z), bruder(Z,Y).
      schwager(X,Y) :- schwester(X,Z), verheiratet(Z,Y).
   h. sohn(X,Y)…
   i. tochter(X,Y)…
   j. großeltern(X,Y)…

3. Überprüfe, ob Prolog die Antworten gibt, die du aufgrund des Stammbaums erwartest.

Diese Regeln haben wir verwendet, um neue Prädikate mit Hilfe von bekannten zu definieren. Regeln kann man auch verwenden, um den Gültigkeitsbereich von schon bekannten Prädikaten zu erweitern.

4. Betrachten wir die Datei fruhst.pl. Es ist bekannt, dass Opa alles mag, was Oma hasst. Diese Regel lautet in Prolog: mag(opa, X) :- hasst (oma, X).
   a) Nimm diese Regel in dem Prologprogramm auf.

b) Welche Antworten erwartest du bei den folgenden Fragen? Überlege zunächst, prüfe dann mit Hilfe einer Prolog-Anfrage, ob das Ergebnis mit deinen Erwartungen übereinstimmt.
   a. ?- mag(opa,X).
   b. ?- mag(X; kuchen).
   c. ?- mag(opa, muesli).
   d. ?- hasst(opa, X).

5. Öffne nun das Programm **liebe.pl.**
   Definiere ein Prädikat idealesPaar, das auf (X,Y) zutrifft, falls X von Y und Y von X geliebt wird.

„Erzeugen einer Sprache“

1. Öffne die **Datei grammatik.pl** im SWI-Prolog-Editor.

2. Überprüfe, ob die folgenden Sätze zu der Sprache des Programms gehören.
   a. ?- satz(der, jaeger, bellt).
   b. ?- satz(flieht, der, hund).

3. Verwende das Prädikat Satz, um alle möglichen Sätze dieser Sprache zu **erzeugen**. Wie viele verschiedene Sätze erwartest du? (Tipp: Variablen verwenden!)

4. In einer Gaststätte gibt es:

   Vorspeisen: Tomatensuppe, Lauchsuppe, Fleischbrühe mit Backerbsen.
   Hauptgerichte: Sauerbraten mit Spätzle, Leberkäse mit Kartoffeln, Hackbraten mit Reis.
   Nachspeisen: Eis, Obstsalat, Bienenstich.

   Ein Menü besteht aus Vorspeise, Hauptgericht und Nachspeise.

   Schreibe ein Programm, das ein dreistelliges Prädikat menue enthält. Dieses Prädikat soll Menüvorschläge überprüfen und erzeugen können.

Suchstrategie Wie sucht Prolog?

Der junge Prinz sucht die schöne Tänzerin der vergangenen Nacht. Auf der Flucht hat sie ihren goldenen Schuh verloren. Mit diesem besucht er nun die Töchter des Landes, um nachzuschauen, bei welcher der Fuß in den Schuh passt. Die Suche wäre weniger mühsam, wenn die Daten der Untertanen schon auf dem Computer verfügbar wären. Es sei etwa auf dem königlichen Hofcomputer eine PROLOG-Datenbasis **aschenputtel.pl** abgelegt.

schuhgroesse(adelheid,34).

```
schuhgroesse(agnes,28).
schuhgroesse(aschenputtel,26).
schuhgroesse(brunhilde,44).
schuhgroesse(kunigunde,28).
schuhgroesse(walburga,38).
```

Anfragen mit Konstanten:
Was passiert, wenn die Anfrage
?- schuhgroesse(aschenputtel,26)
gestellt wird?

Der Prolog-Interpreter vergleicht die Fakten der Datenbasis der Reihe nach mit der Anfrage. Beim dritten Faktum wird eine Deckung erreicht, d. h. die Anfrage und dieses Faktum stimmen überein. Man sagt die Anfrage und das Faktum **matchen**.
Prolog arbeitet also genau so wie der Prinz. Die Anfrage (der Schuh) wird mit einem Faktum (einem Fuß) verglichen. Passen beide nicht zusammen, so geht die Suche zum nächsten Faktum (Fuß). Passen sie, wird das dem Benutzer mit true mitgeteilt. Wird die gesamte Datenbasis ohne Erfolg durchlaufen, so gibt Prolog false zurück.

Anfragen mit Variablen:
?- schuhgroesse(X,26).

Eine Variable kann mit jeder Konstante matchen.

Die Anfrage matcht ebenfalls mit dem dritten Faktum, dabei wird X mit aschenputtel belegt. Das Ergebnis der erfolgreichen Suche wird mit *X = aschenputtel* ausgegeben.
Verlangen wir vom System weitere Antworten auf diese Frage, so löst Prolog die Variable X von der Konstanten aschenputtel und setzt die Suche fort. Da die Datenbasis keine weiteren Möglichkeiten des Matchens findet, gibt es die Antwort false aus.
Würde es noch weitere Antworten geben, so würde diese dann der Reihe nach ausgegeben.

Die Suche erfolgt von oben nach unten, dies nennt man Tiefensuche.

Suchstrategie Wie sucht Prolog bei mehrteiligen Anfragen?

Betrachte das Beispiel des Stammbaums. Wir wollen nach dem Vater von Daisy suchen.
?- elter(daisy, X), maennl(X).

Ziel ist es, dass beide Teilziele für ein X erfüllt werden, da die Anfrage über eine und-Verknüpfung gestellt wurde.

Schritt 1: Betrachtung des ersten Teilziels *elter(daisy, X)*
Mit Hilfe der Tiefensuche matcht die Anfrage mit *X = clemens*.
Die Variable X ist damit instantiiert mit clemens.
Schritt 2: Betrachtung des zweiten Teilziels mit X = clemens: *maennl(clemens).*
Diese Forderung wird beim Durchsuchen bestätigt, damit sind beide Teilziele erreicht worden und der Benutzer erhält die Antwort: *X = clemens*.

Was passiert, wenn die Anfrage in der anderen Reihenfolge gestellt wird?
?- maennl(X), elter(daisy, X).

Schritt 1: Betrachtung des ersten Teilziels *maennl(X)*
Mit Hilfe der Tiefensuche matcht die Anfrage mit *X = adam*.
Die Variable X ist damit instantiiert mit adam.
Schritt 2: Betrachtung des zweiten Teilziels mit X = adam: *elter(daisy, adam).*
Dies kann mit Hilfe der Datenbasis nicht bestätigt werden. Die Belegung der Variablen X führt also nicht zum Ziel, somit wird sie rückgängig gemacht. Die Variable X wird wieder frei gegeben.
Schritt 1: Betrachtung des ersten Teilziels *maennl(X)*, bei dem folgend Faktum von maennl(adam)
Mit Hilfe der Tiefensuche matcht die Anfrage mit *X = alfred*.
Die Variable X ist damit instantiiert mit alfred.
Schritt 2: Das zweite Teilziel kann wieder nicht erreicht werden.
Schritt 1: Diese Schritte wiederholen sich bis X mit der Konstanten clemens belegt wird.
Schritt 2: Betrachtung des zweiten Teilziels mit X = clemens: *elter(daise, clemens).*
Diese Forderung wird beim Durchsuchen bestätigt, damit sind beide Teilziele erreicht worden und der Benutzer erhält die Antwort: *X = clemens*.

Dies Verfahren nennt man **BACKTRACKING** (Rücksetzen).

Ein weiteres Beispiel: Verwandtschaftsverhältnisse

Folgende Wissensbasis sei vorhanden.

weiblich(erna).
weiblich(thea).
weiblich(martha).
weiblich(theresa).
maennlich(franz).
elternVon(erna, thea). umgangssprachlich: Erna ist ein Elternteil von Thea.
elternVon(thea, martha).
elternVon(thea, theresa).
elternVon(franz,martha).
elternVon(franz,theresa).

Was passiert bei folgenden Anfragen:

1. *?- weiblich(theresa).*
   Prolog vergleicht diese Anfrage der Reihe nach mit den Fakten der Wissensbasis. Beim vierten Faktum erreicht es eine Deckung, d. h. die Anfrage und dieses Faktum **matchen**. Die Ausgabe ist *true*.
2. *?- weiblich(doris).*
   Bei dieser Anfrage wird die gesamte Wissensbasis durchlaufen, ohne eine Übereinstimmung zu finden, die Ausgabe ist *false.*
3. *?- weiblich(X).*

Die Variable kann mit jeder Konstanten matchen. Somit matcht X mit erna. Das Ergebnis der erfolgreichen Suche wird ausgegeben: X = erna. Verlangt man vom Programm weitere Antworten, so löst Prolog die Variable X von der Konstanten erna und setzt die Suche fort. Beim zweiten Faktum wird die Variable X nun mit der Konstanten thea verbunden, und es erfolgt die Ausgabe: X = thea. usw.

4. *?- maennlich(X),elternVon(X, martha).* (Wir suchen den Vater von Martha.)
   Ziel ist es, beide Forderungen zu erfüllen. Im ersten Schritt wird versucht, die erste Forderung *maennlich(X)* zu erfüllen. Nach einigen Vergleichen wird die Möglichkeit *maennlich(franz)* gefunden, so wird X mit der franz belegt. Das zweite Teilziel lautet nun *elternVon(franz, martha).* Die Überprüfung der Klausel ergibt, dass diese Forderung richtig ist, somit ergibt sich die Ausgabe *X = franz*.
5. *?- elternVon(X, martha), maennlich(X).* (Wir suchen den Vater von Martha.)
   Zunächst wird das erste Teilziel *elternVon(X, martha)* angestrebt. Nach einigen Vergleiche wird die Möglichkeit *elternVon(thea, martha)* gefunden. Die Variable X wird mit thea instanziiert (belegt). Das zweite Teilziel lautet dann *maennlich(thea)*. Diese Anfrage kann nicht bestätigt werden, so dass Prolog nach dem Durchlauf der gesamten Datenbasis die Instanziierung von X rückgängig macht und die Variable X wieder frei gibt. Das erste Teilziel matcht nun *elternVon(franz,martha).* Die Variable X wird mit franz belegt und erneut wird das zweite Teilziel mit *maennlich(franz)* überprüft. Dies kann bestätigt werden. Erst jetzt, wenn beide Teilziele erreicht sind, wird die Antwort *X = franz* ausgegeben.

   Die beiden Anfragen 4 und 5 sind vom deklarativen (beschreibenden) Standpunkt aus gleichwertig, sie sind aber verschieden, wenn man sie unter prozeduralen Gesichtspunkten betrachtet, d. h. ihre Abarbeitung verfolgt. Deklarativ kann man die 5. Anfrage übersetzen mit „Wer ist Elternteil von Martha und männlich?“ oder prozedural mit „Suche ein Elternteil X von Martha, suche solange, bis du ein männliches X mit dieser Eigenschaft findest.“
   Um die Zwischenergebnisse für den Anwender sichtbar zu machen, kann man das Prädikat *write(X)* einfügen.
   Beispiel: *?- elternVon(X, martha), write(X), nl, maennlich(X).*
   Das Prädikat *nl* bedeutet „new line“ und bewirkt einen Zeilenvorschub.

Die Methode mit der der Prolog-Interpreter nach einer Lösung sucht, bezeichnet man als **Backtracking (Rücksetzen)**. Dabei handelt es sich um einen leistungsstarken Grundalgorithmus. Entscheidungen, die in eine Sackgasse führen, werden wieder rückgängig gemacht und es wird die nächste Möglichkeit ausprobiert.
Die Reihenfolge der Klausel ist hierbei entscheidend. Bei der Überprüfung wird die Wissensbasis **von oben nach unten (Tiefensuche)** durchsucht. Innerhalb eines Regelrumpfes werden die einzelnen Teilziele von links nach rechts abgearbeitet. Ist die Voraussetzung erfüllt, so wird die Variable mit der entsprechenden Konstanten verknüpft. Der „Ort“ der Bindung, d. h. die entsprechende Klausel, ist ein sogenannter „Backtrack-Punkt“, wenn zum gleichen Prädikat weitere Klauseln existieren, die abgearbeitet werden müssen. Im fünften Beispiel ist *elternVon(thea, martha)* ein solcher Backtrack-Punkt. Die weitere Überprüfung verläuft erfolglos, so dass nun das Zurücksetzen (Backtracking) beginnt, d. h. alle Variablenbindungen werden bis zum letzten Backtrack-Punkt gelöst und es wird nach einer weiteren Variablenbelegung gesucht.

Aufgaben:

1. Gib in der Datei **stammb.pl** folgende Anfrage ein:

   ```
   ?- elter(daisy,X), write(X), nl, weibl(X).
   ?- weibl(X), write(X), nl, elter(daisy,X).
   ```

   Jetzt kannst du die Zwischenergebnisse verfolgen.

Arithmetik

Prolog enthält das Prädikat **is**, dabei handelt es sich um ein partielles Prädikat, d. h. bei seiner Verwendung das Argument rechts von is einen Wert haben muss.

X is Ausdruck ist wahr, wenn X zum Wert des arithmetischen Ausdrucks passt, d. h. mit dem Wert matcht.

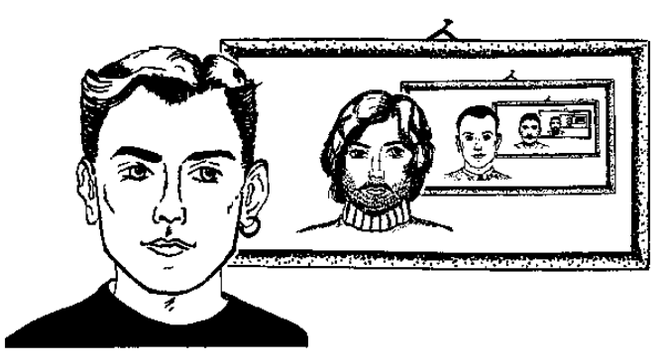
Vergleichsoperatoren in Prolog:

X = Y	die Zahlen X und Y matchen
X\= Y	ungleich
X < Y	kleiner (=< kleiner oder gleich)
X > Y	größer (=> größer oder gleich)

Arithmetische Operatoren in Prolog:

X + Y	Summe
X – Y	Differenz
X * Y	Produkt
X / Y	Quotient, ganzzahlig
X mod Y	Rest bei Division

Rekursion



Donald (siehe Stammbaum) hat sich vor dem Bild seines Vaters fotografieren lassen. Das Bild von seinem Vater wurde vor 25 Jahren vor dem Bild von Donalds Opa aufgenommen. Und sein Opa stand vor 50 Jahren vor dem Bild von Donalds Uropa. (usw.) Auf diese Weise ist in einem Bild eine ganze Galerie von Vorfahren eingefangen.

Um festzustellen, ob jemand ein Vorfahr von jemanden ist, genügt es zu wissen, dass derjenige ein Vorfahr eines Elternteils ist.

In Prolog lässt sich dieser Zusammenhalt folgendermaßen ausdrücken:

(1) vorfahr(X,Y) :- elter(X,Y).
(2) vorfahr(X,Y) :- elter(X,Z), vorfahr(Z,Y).

Das bedeutet: Vorfahren von X sind die Eltern von X und die Vorfahren der Eltern von X. Die Regel vorfahr greift also teilweise auf sich selbst zurück.

Betrachtet man die beiden Regeln deklarativ (beschreibend), so greift die Regel (1) für den einfachsten Fall. Die Regel (2) umfasst alle Regeln für die Prädikate *grosselter, urgrosselter,ururgrosselter,…* Die Definition von *vorfahr* ist also logisch richtig. Prozedural (Abarbeitung verfolgend) betrachtet löst die Regel (2) nie eine Anfrage direkt, sondern dient dazu, die Suche auf eine einfachere, gleichartige Suche zurückzuführen. Es wird z. B. die Lösung der „schwierigen" Aufgabe ?- *vorfahr(donald,baldur).* gesucht. Die Schwierigkeit der Aufgabe liegt darin, dass die Suche nicht mit Hilfe von Regel (1) zum Ziel führt. Prolog greift nun auf Regel (2) zurück und belegt Z mit Clemens. Das erste Teilziel der rechten Seite ist erfüllt. Nun muss noch das zweite Teilziel vorfahr(clemens,baldur) überprüft werden. Dies matcht mit Regel (1).

Kann eine Anfrage der obrigen Form positiv beantwortet werden, so wird zuletzt stets die Regel (1) angewendet, deshalb nennt man sie **REKURSIONSAUSSTIEG**.

Aufgaben:

1. Öffne die **Datei stammb.pl** im SWI-Prolog-Editor.

2. Ergänze die beiden Vorfahr-Regeln. Stelle folgende Anfragen an die Wissensbasis. Notiere die Ausgaben.
   a. ?- vorfahr(donald,V).
   b. ?- vorfahr(daisy,cosima).
   c. ?- vorfahr(cosima, X).
   d. ?- vorfahr(X,bernd).

3. Welche Anfrage musst du stellen, wenn du die Nachfahren von Anna suchst?

4. Man kann also die Nachfahren mit Hilfe von *vorfahr* suchen. Besser ist es jedoch, ein eigenes Prädikat *nachfahr* zu verwenden. Definiere ein solches Prädikat unter Verwendung des Prädikates *kind*.

5. Um die Suchstrategie von Prolog nachzuvollziehen, kann man den TRACE-Modus einschalten. Gib dazu *?-trace.* Ein (*notrace* schaltet ihn wieder ab). Wird im Trace-Modus eine Anfrage an Prolog gestellt, so gibt Prolog die vollständigen Informationen über deren Abarbeitung aus.

   CALL: Prolog versucht ein Ziel zu erfüllen.
   FAIL: Das Ziel scheitert.
   REDO: Prolog versucht das aktuelle Ziel erneut zu erfüllen (z. B. durch Weitersuchen in der Datenbasis oder durch Backtracking)

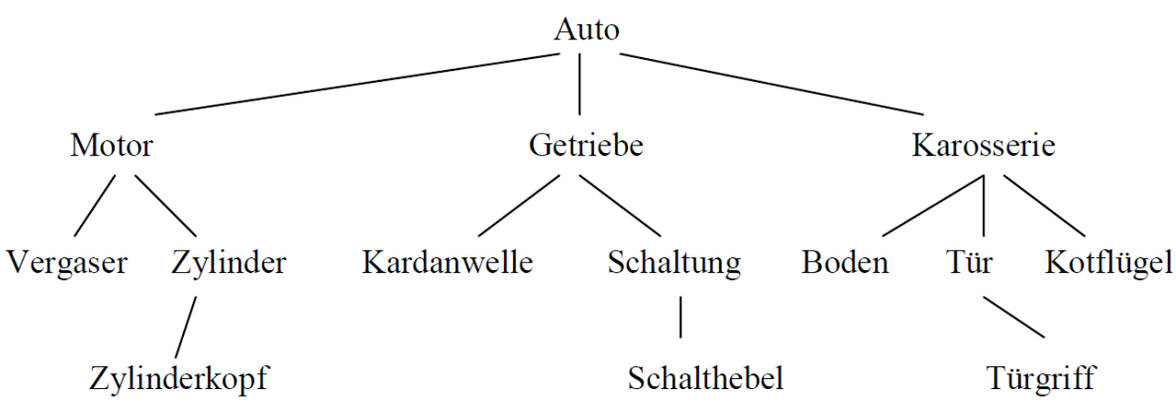
EXIT: Das Ziel wurde erfüllt.

Für die Anfrage *vorfahr(donald,V)* ergibt sich folgende Ausgabe (nur der Anfang):

Betrachte die Schritte die Prolog durchführt und mache dir klar, wie Prolog sucht.

```
4 ?- trace.
true.

[trace] 4 ?- vorfahr(donald,V).
   Call: (6) vorfahr(donald, _G515) ? creep
   Call: (7) elter(donald, _G515) ? creep
   Exit: (7) elter(donald, clemens) ? creep
   Exit: (6) vorfahr(donald, clemens) ? creep
V = clemens ;
   Redo: (7) elter(donald, _G515) ? creep
   Exit: (7) elter(donald, cleopatra) ? creep
   Exit: (6) vorfahr(donald, cleopatra) ? creep
V = cleopatra ;
   Redo: (6) vorfahr(donald, _G515) ? creep
   Call: (7) elter(donald, _G585) ? creep
   Exit: (7) elter(donald, clemens) ? creep
   Call: (7) vorfahr(clemens, _G515) ? creep
   Call: (8) elter(clemens, _G515) ? creep
   Exit: (8) elter(clemens, baldur) ? creep
   Exit: (7) vorfahr(clemens, baldur) ? creep
   Exit: (6) vorfahr(donald, baldur) ? creep
V = baldur ;
   Redo: (8) elter(clemens, _G515) ? creep
   Exit: (8) elter(clemens, barbara) ? creep
   Exit: (7) vorfahr(clemens, barbara) ? creep
   Exit: (6) vorfahr(donald, barbara) ? creep
V = barbara ;
   Redo: (7) vorfahr(clemens, _G515) ? creep
   Call: (8) elter(clemens, _G585) ? creep
   Exit: (8) elter(clemens, baldur) ? creep
   Call: (8) vorfahr(baldur, _G515) ? creep
```

6. Das Diagramm soll einen Überblick über die Teile eines Autos geben:



   a. Lege den Inhalt des Diagramms in einer PROLOG-Datenbasis ab und definiere ein Prädikat teil, wobei teil(X,Y) bedeuten soll, dass Y Teil von X ist.
   Es soll gelten: Ist A ein Teil von B und B ein Teil von C, so ist A Teil von C.
   b. Welche Antworten auf folgende Anfragen erwartest du?
      i. ?- teil(A,motor).
      ii. ?- teil(schalthebel,B).
      iii. ?- teil(tür,motor).
      iv. ?- teil(A,getriebe).
   c. Beschreibe mit deinen eigenen Worten, wie Prolog bei der Suche ?- teil(A, getriebe). vorgeht.
   d. Schicke das fertige Programm und die Antwort von 6c per Mail an mich.

7. Die Türme von Hanoi

Das Problem der Türme von Hanoi ist weit verbreitet. Auf drei Holzstäben können goldene Scheiben wie Perlen aufgefädelt werden. Auf dem ersten Stab ist eine bestimmte Anzahl von Scheiben gesteckt. Diese Scheiben sind der Größe nach angeordnet, so dass die größte Scheibe ganz unten liegt und die kleinste ganz oben. Die Aufgabe besteht nun darin, alle Scheiben von A nach C zu versetzen, wobei nur die oberste Scheibe auf einmal versetzt werden darf und eine größere nie über einer kleineren Scheibe liegen darf.
Schreibe ein Prologprogramm, welches dieses Problem löst. Schicke das Programm per Email an mich.

Hilfen:

- Wenn noch Scheiben auf dem Ausgangsstab sind, dann muss der Turm ohne die untere Scheibe auf den Hilfsstab in der Mitte verlegt werden, dann muss die untere Scheibe zum Zielstab transportiert werden und schließlich muss der auf dem Hilfsstab befindliche Turm zum Zielplatz gebracht werden.
- Wenn alle Scheiben am auf dem Zielstab aufgefädelt sind (d. h. keine Scheiben mehr …), muss ein Abbruch mit Hilfe eines Cuts erfolgen.
- Eine Variable kann reduziert werden, indem man *X1 is X -1* verwendet. Der arithmetische Term im rechten Argument von ‚is' darf nur instanziierte Variablen enthalten.
- Lies die Tipps!

Tipp 1 für die Türme von Hanoi

Probiere es auf dem Papier aus, beginne mit einer Scheibe und notiere dir jeweils die nötigen Schritte. Wie müssen die Türme verschoben werden? Du solltest bei drei Scheiben auf 7 Schritte kommen. Bei 4 Scheiben sind 15 Schritte notwendig.

Hintergrundwissen: Die Anzahl der durchzuführenden Einzelanweisungen beträgt genau 2^N-1, wenn N die Anzahl der Scheiben angibt. Es handelt sich demnach um eine exponentielle Laufzeit, die immer nicht besonders günstig ist. Leider kann man jedoch beweisen, dass man mindestens 2^N-1 Verschiebungen benötigt. Es handelt sich also um eine untere Schranke.

Für die Originalversion dieses Spiels bedeutete dies leider nichts Gutes. Tibetische Mönche sollten einen Turm aus 64 Scheiben mit Hilfe von 3 Stäben versetzen. Es waren also 18.446.744.073.709.551.515 (≈ 18,5 Trillionen) Versetzungen notwendig. Vorausgesetzt die Mönche könnten 1 Millionen Scheiben pro Sekunde versetzen, dann würden sie knapp 600.000 Jahre benötigen, um die Aufgabe zu lösen. Aber selbst 1 Millionen Versetzungen pro Minute sind unrealistisch. Die Mönche glaubten, die Welt ginge unter, bevor sie fertig würden.

Wichtig für dich ist also, dass du zum Programmtesten nur kleinere Zahlen für N verwendest!

Tipp 2 für die Türme von Hanoi

Lass dein Programm eine Ausgabe (mit Hilfe von write) tätigen, damit du kontrollieren kannst, ob die richtige Scheibe auf den richtigen Stab gelegt wird.

Mögliche Ausgabe:

```
ausgeben(N,Von, Nach) :-
   writeln('Bewege den Ring '),write(N),
   write(' von  Platz '), write(Von),
   write(' nach Platz '), write(Nach).
```

Tipp 3 für die Türme von Hanoi

Der Lösungsweg für drei Scheiben lässt sich rekursiv wie folgt angeben:

- Transportiere zwei Scheiben vom Ausgangsstab zum Hilfsstab.
- Verschiebe die dritte Scheibe vom Ausgangsstab zum Zielstab.
- Transportiere die zwei Scheiben vom Hilfsstab zum Zielstab.

Schritt drei muss wieder in die drei oben genannten Schritte unterteilt werden, allerdings mit einer Scheibe weniger (→Rekursion).

Bei einer anderen Idee, geht man davon aus, dass die drei Stäbe im Kreis stehen. Der Algorithmus ist dann sehr einfach: Führe folgende Schritte solange aus, bis Schritt zwei nicht mehr möglich ist:

- Bewege die kleinste Scheibe auf den im Uhrzeigersinn nächsten Stab.
- Führe die einzige mögliche zulässige Bewegung mit einer anderen Scheibe als der kleinsten aus.

Stopp!

Schritt 2 kann nicht ausgeführt werden, wenn bereits alle Scheiben in der richtigen Ordnung auf einem anderen Stab verschoben wurden, da dann nur die kleinste irgendwo oben liegt, wenn Schritt 2 ausgeführt werden kann, dann gibt es genau eine Möglichkeit hierfür.

Tipp 4 für die Türme von Hanoi

Welche rekursiven Aufrufe erfolgen bei drei Scheiben?

Die Rekursion besteht in jeder Ebene aus 3 Anweisungen: verlege, Ausgabe, verlege.

A: Ausgangsstab, H: Hilfsstab, Z: Zielstab

Aufruf im Hauptprogramm: verlege(3,A,H,Z) (Ebene 0)
→ Da 3 ≠ 0, erfolgt der Aufruf: verlege (2,A,Z,H) (Ebene 1) (**)
→ Da 2 ≠ 0, erfolgt der Aufruf: verlege(1,A,H,Z) (Ebene 2) (*)

→ Da 1 ≠ 0, erfolgt der Aufruf: verlege(0,A,Z,H) (Ebene 3)
→ Da 0 = 0, stoppt die Rekursion und die zweite Anweisung wird ausgeführt, d. h. die kleinste Scheibe wird von A nach Z gelegt (Ausgabe mit write)
→ Nun muss verlege(0,H,A,Z) aufgerufen werden (3. Anweisung), dies führt zum Rekursionsabbruch. (Es erfolgt der Wechseln zurück in Ebene 2.)

→ Es wird bei verlege(1,A,H,Z) weitergearbeitet. (siehe *)
Die zweite Anweisung der angefangenen Rekursion in Ebene 2 wird fortgesetzt, d. h. die zweite Scheibe wird von A nach H verlegt. (Ausgabe)
→ Es erfolgt der Aufruf: verlege(1,Z,A,H). (3. Anweisung in Ebene 2).

→ Da 1 ≠ 0, erfolgt der Aufruf: verlege(0,Z,H,A) (Ebene 3)

→ Da 0 = 0, stoppt die Rekursion und die kleinste Scheibe wird von Z nach H gelegt (Ausgabe) Der Restturm steht nun bei H. (Schritt 1 von Tipp 3 ist abgearbeitet)
→ Nun muss verlege(0,A,Z,H) aufgerufen werden, dies führt zum Rekursionsabbruch.

Es erfolgt der Wechseln zurück in Ebene 2. Die 3. Anweisung ist nun abgearbeitet, damit erfolgt der Wechsel zurück in Ebene 1.
→ Es wird bei verlege(2,A,Z,H) weitergearbeitet. (siehe **) Die zweite Anweisung der angefangenen Rekursion in Ebene 1 wird fortgesetzt, d. h. die dritte Scheibe wird von A nach Z verlegt. (Ausgabe) (Schritt 2 von Tipp 3 ist abgearbeitet)
→ Es erfolgt der Aufruf: verlege(2,H,A,Z). (3. Anweisung in Ebene 1).

→ Da 2 ≠ 0, erfolgt der Aufruf: verlege(1,H,Z,A) (Ebene 2) (***)
→ Da 1 ≠ 0, erfolgt der Aufruf: verlege(0,H,A,Z) (Ebene 3)
→ Da 0 = 0, stoppt die Rekursion und die kleinste Scheibe wird von H nach A gelegt (Ausgabe)
→ Nun muss verlege(0,Z,H,A) aufgerufen werden, dies führt zum Rekursionsabbruch.

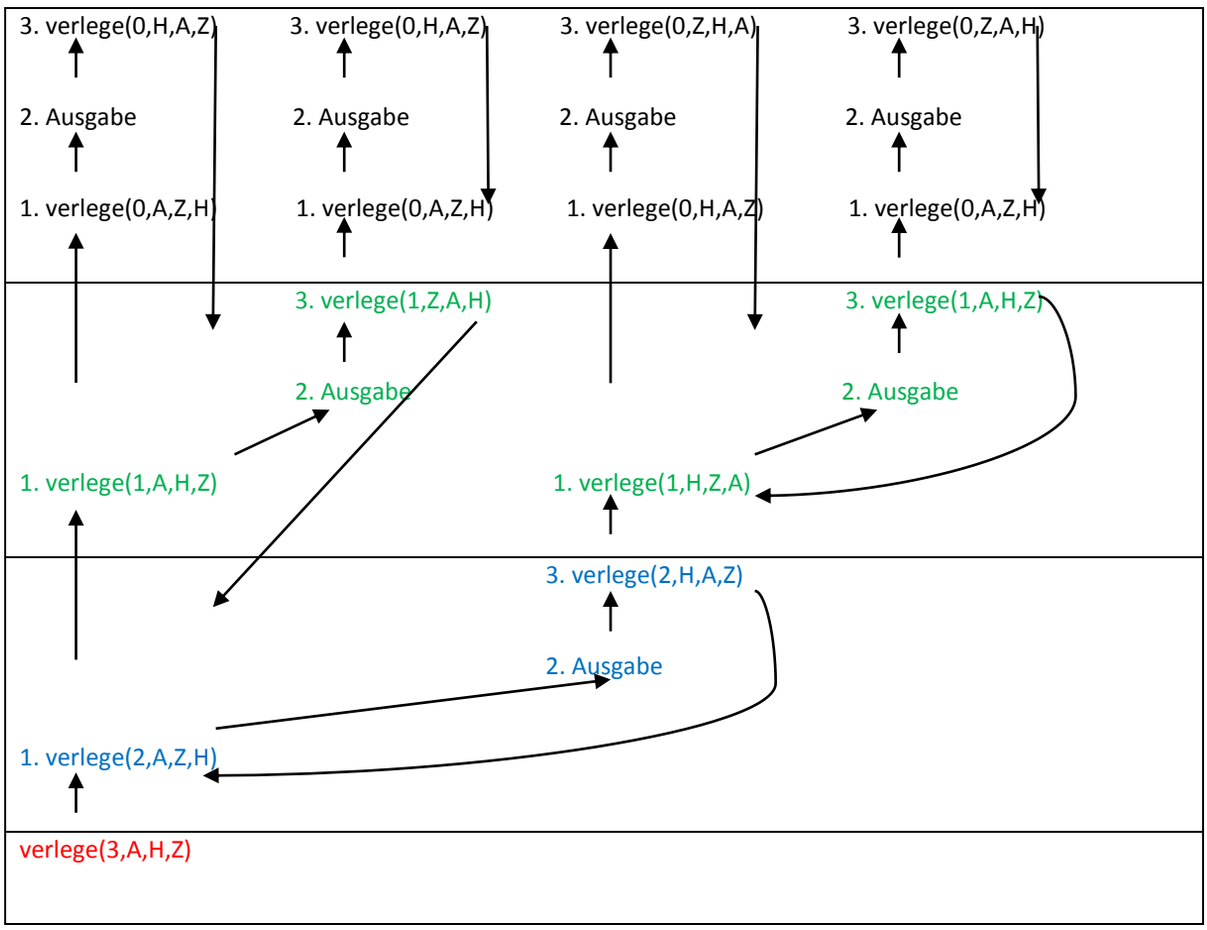
(Es erfolgt der Wechseln zurück in Ebene 2.)
→ Es wird bei verlege(1,H,Z,A) weitergearbeitet.(siehe ***) Der zweite Schritt der angefangenen Rekursion in Ebene 2 wird fortgesetzt, d. h. die zweite Scheibe wird von H nach Z verlegt. (Ausgabe)
→ Es erfolgt der Aufruf: verlege(1,A,H,Z). (3. Anweisung in Ebene 2).

→ Da 1 ≠ 0, erfolgt der Aufruf: verlege(0,A,Z,H) (Ebene 3)
→ Da 0 = 0, stoppt die Rekursion und die kleinste Scheibe wird von A nach Z gelegt (Ausgabe) (Der 3. Schritt von Tipp 3 ist abgearbeitet)
→ Nun muss verlege(0,Z,A,H) aufgerufen werden, dies führt zum Rekursionsabbruch.

(Es erfolgt der Wechseln zurück in Ebene 2.)
Die 3. Anweisung in Ebene 2 ist nun abgearbeitet, also erfolgt der Wechsel in Ebene 1.

Die 3. Anweisung in Ebene 1 ist abgearbeitet, es folgt der Wechsel in Ebene 0.

In Ebene 0 erfolgte der erste Aufruf, hier gab es noch keine drei Schritte, so dass nun alles abgearbeitet ist.



Gewonnenes Wissen weiterverwenden – Wissensbasis erweitern

Bisher haben wir gesehen, dass man aus bekannten Fakten neu Erkenntnisse erlangen kann, wenn man auf diese Fakten die Regeln anwendet. Nun wäre es sehr schön, wenn diese neuen Erkenntnisse auch der Wissensbasis hinzugefügt werden könnten, damit diese sich erweitert und nicht immer nach den gleichen Fakten neu suchen muss.

Beispiel: Fibonacci-Zahlen

Die ersten Fibonacci-Zahlen lauten: 1 1 2 3 5 8 13 21

Die ersten beiden Fibonacci-Zahlen sind 1 und die n-te Fibonacci-Zahl ist die Summe ihrer beiden Vorgängerzahlen, d. h. fib(n) = fib(n-1)+fib(n-2)

Wie man sieht, müsste man ein und dasselbe Ergebnis für höhere Fibonacci-Zahlen mehrfach berechnen. Die Methode ist also höchst ineffektiv, da durch die doppelte Rekursion die Anzahl der rekursiven Aufrufe exponentiell wächst.

Um dir dies klar zu machen, überlege, wie oft die 4. Fibonacci-Zahl berechnet werden muss, um die 8. Fibonacci-Zahl zu berechnen? Wie oft muss die 3. Fibonacci-Zahl berechnet werden, um die 8. Fib-Zahl zu berechnen?

Es liegt also nahe, die berechneten Zwischenergebnisse in der Wissensbasis abzuspeichern, um die erneute Berechnung überflüssig zu machen.

Wie kann man in Prolog neu gewonnene Fakten zur Wissensbasis hinzufügen?

Wie wir bereits festgestellt haben, ist es wichtig, an welcher Position gewisse Klauseln stehen, daher gibt es unterschiedliche Befehle, um die neuen Erkenntnisse einzufügen:

- assert(Klausel) speichert eine Klausel irgendwo
- asserta(Klausel) fügt die Klausel am Anfang der Datenbasis hinzu
- assertz(Klausel) fügt die Klausel am Ende der Datenbasis hinzu
- retract(Klausel) löscht eine Klausel
- listing nach Beendigung des Programms kann man sich alle neu hinzugewonnenen Fakten anzeigen lassen
- retractall(Name(Variablenname)) löscht alle Klauseln zum Namen

Prädikate, die aus Dateien konsultiert werden, können nur dann um Klauseln ergänzt werden, wenn sie zuvor mit *:- dynamic funktor/arität* als **dynamisch** deklariert wurden.

Prolog-Programm für die Fibonacci-Zahlen:

```
% Berechnung der Fibonacci-Zahlen durch Abspeicherung der Zwischenergebnisse
% Man kann nur Ergebnisse aus dynamisch deklarierten Klausel abspeichern.
:- dynamic fib/2.

% Rekursionsaussteig
% fib(X,Y) bedeutet die X.Fibonacci-Zahl lautet Y
fib(1,1):-!.
fib(2,1):-!.

% Rekursion
fib(N,Fib):-
  N > 2,
  N1 is N - 1, fib(N1,Fib1),
  N2 is N - 2, fib(N2,Fib2),
  Fib is Fib1 + Fib2,
  asserta(fib(N,Fib)),!.

  /*Der Cut ist wichtig, da sonst Prolog nach  weiteren Lösungen suchen würde.*/
```

Aufgaben:

1. Öffne das Programm **fibonacci.pl**.
   a. Lass die 8. Fibonacci-Zahl berechnen.
   b. Überprüfe nun die neu eingetragenen Fakten mit dem listing-Aufruf.
   c. Warum sollten die neu gewonnenen Fakten nicht am Ende der bereits bestehenden Klausel eingefügt werden?

2. Wähle eine der beiden Aufgaben aus.
   a. Bekanntlich ist die 2 die kleinste Primzahl (*prim(2)*). Eine größere Zahl als zwei ist Primzahl, wenn sie durch keine kleinere ohne Rest teilbar ist (*Rest is Zahl mod Teiler*). Entwickle entsprechend dieser Definition ein Programm zur Ausgabe aller Primzahlen bis zu einer vorgegebenen Zahl.
   b. Die Binomialkoeffizienten können rekursiv definiert werden:
      bin(n,0) = 1,
      bin(n,n) = 1,
      bin(n, k) = bin(n-1,k-1) + bin(n-1,k).
      Entwickle ein Programm zur Berechnung von Binomialkoeffizienten einmal ohne und einmal mit assert. Vergleiche die beiden Lösungen.

3. Schicke das fertige Programm von Nr. 2 und die Antwort von 1c per Email an mich.

4. **Zusatz für die Schnellen/ Guten:** Das Spiel Nimm ist für zwei Personen gedacht. Am Anfang liegt ein Haufen Streichhölzer auf einem Tisch. Abwechselnd nimmt jeder Spieler höchstens 3 aber mindestens 1 Streichholz weg. Gewonnen hat derjenige, der die letzten Streichhölzer wegnimmt. Der folgende Zugberater sucht den kompletten Spielbaum ab, um einen Gewinnzug zu finden. Dies dauert schon bei kleinen Streichholzhaufen lange. Erkläre dies und verbessere den Zugberater durch das Lernen von Gewinn- und Verlustpositionen. Den unten angegebenen Quellcode findest du in der Datei **nimm.pl**. (Lösung per Email an mich)

```
% Zugberater für Nimm
:-dynamic verlust/1, gewinn/1.

verlust(0).
verlust(X) :- not(gewinn(X)).

gewinn(1).
gewinn(2).
gewinn(3).

gewinn(X) :-
  X > 3,
  (X1 is X - 1;   % Semikolon bedeutet "oder"
   X1 is X - 2;
   X1 is X - 3),
   verlust(X1).

zug(X) :-
   (X1 is X - 1;
    X1 is X - 2;
    X1 is X - 3),
   verlust(X1),
   Wert is X - X1,
   write('Nimm '), write(Wert), writeln(' Streichhölzer').

zug(X) :-
  write('Der Gegner kann gewinnen '), writeln('nimm beliebig.').
```

Dynamische Prolog-Programme sind nicht unproblematisch

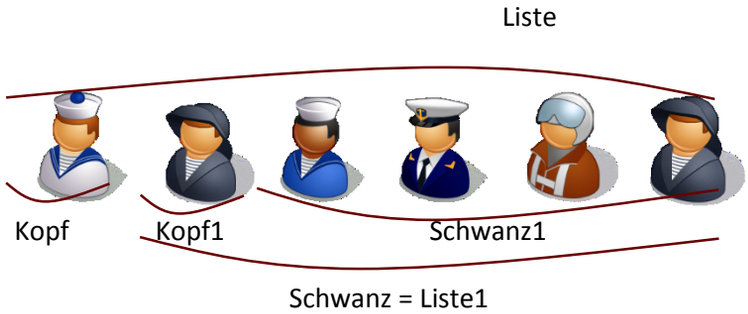
Assert und retract erlauben es, ein Prolog-Programm dynamisch zu ändern. Aus der Sicht der Programmentwicklung ist das sehr problematisch, weil es schwer ist, Fehler in sich ändernden Programmen zu lokalisieren. Andererseits hat man damit Möglichkeiten, effiziente oder auch selbst lernende Programme zu schreiben und das braucht man für KI-Programme!

Der Datentyp Liste

Listen sind Datenstrukturen, die umgangssprachlich als Folge von Objekten beschrieben werden können. Listen spielen eine fundamentale Rolle im Bereich der abstrakten Datentypen. Eine Möglichkeit der Definition einer Liste in Prolog ist folgende:

Eine Liste ist

- **die leere Liste**, dargestellt durch das Atom **[]** oder
- eine Struktur mit den zwei Komponenten ‚Kopf' und ‚Schwanz', wobei der Kopf das erste Element der Liste ist und der Schwanz aus den restlichen Elementen der Liste besteht. Daher wird der Schwanz auch häufig als Restliste bezeichnet.



Eine Liste ist eine geordnete Folge von Elementen beliebiger Länge, d. h.

- es gibt ein erstes, zweites, drittes usw. Element,
- dynamische Datenstruktur, wie viele Elemente die Liste enthält wird erst während des Programmlaufs entschieden.

Wie sehen Listen in Prolog aus?

Es gibt unterschiedliche Darstellungsmöglichkeiten.

- als Aufzählung: **[Element1, Element2, Element3,…]**, wobei Element1 = Kopf
- als Kopf und Restliste: **[Element1 | Restliste]**, wobei ‚|' der Listenoperator ist.
- als Kombination: **[Element1, Element2, Element3 | Restliste]**

Listen können in Prolog unterschiedliche Datentypen als Elemente enthalten, d. h. als Elemente kommen Variablen, Konstanten, aber auch wieder Listen in Frage.

Beispiel: [a, b, c] = [a | b, c] = [a, b | c] = [a, b, c | []]

Grundoperationen auf Listen

Mitgliedschaft

Ein Element X ist in einer Liste L enthalten, wenn

- X der Kopf der Liste L ist oder
- X Element der Restliste ist.

```
% entspricht Systemprädikat member
mitglied(X, [X|_]).
mitglied(X, [_|Y]):-
  mitglied(X, Y).
```

Ein Element anhängen

Ein Element X soll an eine Liste L angehängt werden.

```
haengean(X,[],[X]).
haengean(X,[Y|Rest],[Y|Rest_mit_X]) :-
  haengean(X,Rest,Rest_mit_X).
```

- Ist L = [], so ist die neue Liste, die Liste mit dem Kopf X und der leeren Restliste.
- Ist die Liste L nicht leer, so wird X an den Rest der Liste angehängt. Durch den rekursiven Aufruf, wird das Element X am Ende der Liste angehängt.

Zwei Listen verketten

Eine Liste L2 soll an die Liste L1 angehängt werden, als Ergebnis erhält man die Liste L3. Natürlich kann man mit dieser Methode auch nur ein einzelnes Element an die Liste L1 anhängen, wenn L2 =[X] ist.

```
% entspricht Systemprädikat append
anhaengen([], L, L).
anhaengen([X|L1], L2, [X|L3]):-
  anhaengen(L1, L2, L3).
```

Das Prädikat *haengean* ist damit überflüssig.

- Ist L1 = [], so ist L3 = L2 (erste Klausel).
- Ist L1 nicht die leere Liste, so gilt: L1=[X| Rest]. Die neue Liste soll ebenfalls X als Kopf haben, also ist L3 = [[X| Rest], L2] = [X| [Rest,L2]]
  also kann man L2 an die Restliste von L1 anhängen und schließlich den Kopf von L1 vor die erzeugte neue Liste hängen. Das hört sich zunächst kompliziert an, funktioniert aber.

Ein Element löschen

Das Prädikat loesche enthält drei Argumente: das zu entfernende Element, die zu bearbeitende Liste und die Ergebnisliste.

```
%löschen eines Elementes
loesche(X,[],[]).
loesche(X,[X|Rest],Rest):-!.
loesche(X,[Kopf|Rest],[Kopf|Rest_ohne_X]) :-
  !,loesche(X,Rest,Rest_ohne_X).
```

Folgende Fälle sind zu unterscheiden:

- aus einer leeren Liste kann nichts gelöscht werden
- ist das zu löschende Element der Kopf der Liste, so bleibt die Restliste erhalten
- andernfalls muss das Element in der Restliste gelöscht werden

Aufgrund des Cuts wird nur das erste gefundene Exemplar des zu löschenden Elements aus der Liste entfernt.

```
3 ?- loesche(a,[a,b,a],Ergebnis).
Ergebnis = [b, a].
```

Bei fehlenden Ausrufezeichen wird Backtracking nicht verhindert, man würde alternative Lösungen erhalten.

Länge einer Liste

Bei dem zweistelligen Prädikat *laenge(L,N)* soll die Länge N einer Liste L ermittelt werden.

- Die leere Liste hat die Länge 0.
- Ist die Liste nicht leer, kann man sie in Kopf und Rest aufteilen. Die Länge des Kopfes ist 1, die Länge der Restliste ergibt sich unter Ausnutzung der Rekursion.

```
% entspricht Systemprädikat length
laenge([], 0).
laenge([X|Y], N):-
  laenge(Y, N1),
  N is N1 + 1.
```

Liste umdrehen

Mit dem Prädikat *umdrehen(Liste1, Liste2)* soll eine Liste umgekehrt werden. Die algorithmische Idee besteht darin, den Kopf der Liste abzutrennen, die Restliste umzukehren und abschließend den Kopf an das Ende der umgekehrten Restliste anzuhängen.

```
% entspricht Systemprädikat reverse
umdrehen(Liste1, Liste2):-
  Liste1 = [Kopf|Rest],
  umdrehen(Rest, Rest1),
  anhaengen(Rest1, [Kopf], Liste2).
umdrehen([], []).
```

Aufgaben:

1. Welche Antworten liefern folgende Anfragen?
    a. [X| Y] = [rhein, elbe, weser, mosel].
    b. [X| [weser, mosel]] = [elbe, weser, mosel].
    c. [X| [weser, mosel]] = [rhein, elbe, weser, mosel].
    d. [Z| Rs] = [1, 2, 3, 4, 5].
    e. [3, 4, 5] = [X| Rs].
    f. [Kopf|Rest] = [a].
    g. [X| Rest] = [].
    h. [X| Rest] = [[]].
2. Die Listenprädikatsammlung der Datei **LISTEN.pl** soll um einige Prädikate ergänzt werden. *Füge die Prädikate hinzu und schicke die fertige Datei an mich per Email.*
    a. Das Prädikat *gleich* soll prüfen, ob zwei Listen elementweise gleich sind, d. h. die Abfrage *gleich([a,b,c],[a,b,c])* soll bestätigt werden.
    b. Mit dem Prädikat *praefix* soll festgestellt werden können, ob eine erste Liste den Beginn einer zweiten Liste darstellt.
    c. Das Prädikat *erstes_Element* soll den Kopf einer Liste ausgeben.
    d. Das Prädikat *letztes_Element* soll das letzte Element einer Liste ausgeben.

Von der Wissensbasis zum Expertensystem

Hinzufügen und Löschen von Klauseln

Die Wissensbasis kann mit neuen Fakten und Regeln ergänzt werden, wenn die einzelnen Klauseln dynamisch deklariert werden.

Lösungsmengen bestimmen

In manchen Anwendungsfällen möchte man nicht einzelne Lösungen, sondern eine Lösungsmenge ausgegeben bekommen. Beispielsweise interessiert uns, wie viele Kinder jemand hat und man möchte die Kinder in einer bestimmten Reihenfolge ausgeben.

Das Prädikat, welches alle Lösungen in einer Liste speichert heißt: findall. Wie kann man findall realisieren?

1. Alle Lösungen müssen erzeugt und gespeichert werden. Zum Speichern verpackt man eine Lösung in ein Faktum: gefunden.

    In Prolog:
    % findall(+Term, +Ziel, -Lösungsliste)
    Findall(Loesung, Ziel, _) :- call(Ziel), assertz(gefunden(Loesung)), fail.

    Fail sorgt hier dafür, dass durch Backtracking auch die nächste findall-Klausel verwendet wird.

2. Einsammeln aller gefundener Lösungen und gleichzeitiges Löschen der Lösungen aus der Wissensbasis, damit kein unnötiger Speicherplatz vergeudet wird.
    Findall(_, _, Liste) :- sammeln(Liste).

Sammeln endet genau dann, wenn keine gefunden-Klausel mehr vorhanden ist. Dies kann mit dem Systemprädikat clause überprüft werden.
Sammeln([]) :- not(clause(gefunden(_), true)).

Künstliche Intelligenz

Was versteht man unter künstlicher Intelligenz?

„Künstliche Intelligenz (KI, englisch artificial intelligence, AI) ist ein Teilgebiet der Informatik, welches sich mit der Automatisierung intelligenten Verhaltens befasst. Der Begriff ist insofern nicht eindeutig abgrenzbar, da es bereits an einer genauen Definition von Intelligenz mangelt. Dennoch findet er in Forschung und Entwicklung Anwendung. Im Allgemeinen bezeichnet „künstliche Intelligenz“ oder „KI“ den Versuch, eine menschenähnliche Intelligenz nachzubilden, d. h., einen Computer zu bauen oder so zu programmieren, dass dieser eigenständig Probleme bearbeiten kann. Oftmals wird damit aber auch eine effektvoll nachgeahmte, vorgetäuschte Intelligenz bezeichnet, insbesondere bei Computerspielen, die durch meist einfache Algorithmen ein intelligentes Verhalten simulieren soll.“ (wikipedia.org, 2014)

Es ist also nicht so einfach, genau zu beschreiben, was genau man unter „Künstlicher Intelligenz“ versteht. Zumal der Begriff einerseits eine Eigenschaft und andererseits eine wissenschaftliche Disziplin widerspiegeln soll. Dennoch gibt es einen großen Forschungsbereich zu diesem Thema.

Helbig versucht mit folgenden Bedingungen für Computerleistungen dem Kern der künstlichen Intelligenz näher zu kommen:
„a) ihre Hervorbringung verlangt nach allgemeinem Verständnis menschliche Intelligenz,
b) für ihre Realisierung liegen keine speziell angepassten Algorithmen vor.“ (Helbig, 1996, S. 11)
Aufgaben, die von einem Computer geleistet werden und der künstlichen Intelligenz zugeordnet werden sollen, müssen also zum einen komplex genug sein, so dass sie nicht trivial von jedem gelöst werden können und dürfen nicht auf allgemein bekannten Algorithmen beruhen, z. B. Integralbildung in der Analysis.

Wir werden versuchen, dem Verständnis der künstlichen Intelligenz näher zu kommen.

Der Dialog zwischen der Bombe und dem Astronauten

Das Raumschiff „Dark Star“, das dem Film den Titel leiht, ist seit zwanzig Jahren im Weltall unterwegs. Das Bordbuch schreibt das Jahr 2200, und die Technik hat sich so weit entwickelt, dass sogenannte instabile Planeten mit speziellen Bomben, die über künstliche Intelligenz verfügen und kommunikationsfähig sind, zerstört werden können. Dies ist die Mission der „Dark Star“.

Die Besatzung besteht aus vier mehr oder minder zurechnungsfähigen Astronauten nebst ihrem tödlich verunglückten und daher tiefgefrorenen Kapitän, dessen Gehirn jedoch in besonderen Situationen immer noch um Rat gefragt werden kann. In der Regel interessiert es

sich aber nur für die neuesten Baseballresultate. An Bord ist außerdem noch ein außerirdisches Wesen, das deutlich an einen quietschenden, hüpfenden Gummiball erinnert und zusätzlich für Turbulenzen sorgt. Die Situation im Schiff spitzt sich zu: Der Schlafsaal ist unbrauchbar, das Toilettenpapier verbraucht und die Luke des Bombenschachts, in dem sich eine scharfe Bombe befindet, klemmt. Da fast gar nichts mehr funktioniert, droht die Explosion der Bombe das Raumschiff zu vernichten. In ihrer Not wenden sich

die Astronauten an das Gehirn des toten Kommandanten, der Leutnant Doolittle den Rat gibt, die Bombe zu überzeugen, nicht zu detonieren. Doolittle verlässt das Schiff und beginnt eine Diskussion mit der Bombe.

A: Hallo, Bombe, hörst Du mich?

B: Selbstverständlich.

A: Bist Du bereit, ein paar Zusammenhänge zu erörtern?

B: Ich bin Vorschlägen gegenüber immer empfänglich.

A: Fein, dann denke mal darüber nach: Woher weißt Du, dass Du existierst?

B: Natürlich existiere ich!

A: Aber woher weißt Du, dass Du existierst?

B: Es ist eine intuitive Erkenntnis.

A: Intuition ist kein Beweis. Was für konkrete Beweise hast Du dafür, dass Du existierst?

B: Hm, nun, ...ich denke, also bin ich.

A: Das ist gut! Das ist sehr gut! Aber woher weißt Du, dass außer Dir etwas existiert?

B: Meine sensorischen Apparaturen vermitteln es mir.

A: Ah, richtig.

B: Das macht Spaß.

A: Jetzt hör‘ mal gut zu! Hier kommt die große Frage: Woher weißt Du, dass die Erkenntnis, die Deine Sinnesapparaturen Dir vermitteln, korrekt ist? Ich will auf Folgendes hinaus: Die einzige Erfahrung, die Dir direkt zur Verfügung steht, sind Deine sensorischen Daten. Und diese Daten sind lediglich eine Reihe elektrischer Impulse, die Dein Rechenzentrum stimulieren.

B: Mit anderen Worten: Alles was ich wirklich über die Außenwelt weiß, wird mir über meine elektrischen Verbindungen vermittelt.

A: Genau!

B: Aber das würde ja bedeuten, dass ich überhaupt nicht mit absoluter Sicherheit weiß, wie das Universum um mich herum ist.

A: Genau, genau das ist es.

B: Interessant! Ich wünsche, ich hätte mehr Zeit dieses Thema zu diskutieren.

A: Wieso hast Du nicht mehr Zeit?

B: Weil ich in 75 sec. detonieren muss!

A: Also Bombe, denk über die nächste Frage gut nach! Was ist der einzige Zweck Deiner Existenz?

B: Zu explodieren, natürlich.

A: Und das kannst Du nur einmal, richtig?

B: Das stimmt.

A: Und Du würdest doch wohl nicht auf der Grundlage falscher Daten explodieren wollen, oder?

B: Natürlich nicht.

A: Ich stelle fest: Du hast bereits zugegeben, dass Du keinen wirklichen Beweis für die Existenz der Außenwelt hast.

B: Na schön.

A: Also hast Du auch keinen Beweis dafür, dass Pinback Dir befohlen hat zu explodieren.

B: Ich erinnere mich ganz deutlich an den Detonationsbefehl. Mein Gedächtnis ist in solchen Dingen sehr gut.

A: Selbstverständlich erinnerst Du Dich daran. Aber alles, woran Du Dich erinnerst, ist eine Reihe sensorischer Impulse, von denen Du jetzt weißt, dass sie keine eindeutige Verbindung mit der äußeren Realität haben.

B: Richtig, aber da es so ist, habe ich auch keinen Beweis dafür, dass Sie mir das alles wirklich sagen.

A: Darum geht es doch überhaupt nicht! Wenn Zusammenhänge logisch sind, dann sind sie es unabhängig von ihrem Ursprung.

B: Hm.

A: Wenn Du detonierst...

B: In 9 Sekunden.

A: ... willst Du das doch nicht auf der Grundlage falscher Daten tun.

B: Ich habe keinen Beweis dafür, dass es falsche Daten sind.

A: Du hast keine Beweise dafür, dass es richtige Daten sind.

B: Ich muss weiter darüber nachdenken.

Die Bombe fährt zurück

A: Also dann, Bombe, mach Dich bereit, neue Befehle zu empfangen.

B: Sie haben falsche Daten! Daher werde ich sie ignorieren.

A: Hallo, Bombe.

B: Falsche Daten können mich nur verwirren. Deshalb werde ich mich weigern, mich weiter danach zu richten.

A: Hey, Bombe!

B: Das einzige, was existiert, bin ich selbst.

Aufgaben:

1) Welche Erkenntnisse hat die Bombe und wie gelangt sie zu diesen Einsichten?
2) Ein Schlüsselsatz des Philosophen René Descartes (1596 bis 1650) lautet „Ich denke, also bin ich!“ Wie weit ist die Bombe in der Entwicklung ihres Bewusstseins fortgeschritten?
3) Hältst Du Maschinen, die auf diese Art und Weise wie die Bombe „In der Welt sind“, zukünftig für möglich?
4) Stell dir vor, du wärest an der Stelle der Bombe? Hättest du genauso regiert?

Künstliche Intelligenz? – Was gehört dazu?

Roboterexperiment

Auf dem Fußboden ist der Startpunkt für einen Roboter gekennzeichnet. Auf dem Tisch vor dem Startpunkt befindet sich ebenfalls eine Markierung, auf der ein Stift liegt. Auf dem Stuhl neben dem Tisch befindet sich eine dritte Markierung. Deine Aufgabe ist es nun, eine Folge von Anweisungen aufzuschreiben, so dass ein Roboter beim Startpunkt beginnt, den Stift vom Tisch nimmt und ihn anschließend auf dem Stuhl ablegt. Dabei kann der Roboter nichts sehen und nichts fühlen. Verwende möglichst klare und kurze Anweisungen und nummeriere sie durch, wie zum Beispiel: 1. „Gehe einen Schritt nach vorne“ oder 5. „Greife zu.“

a) Suche dir einen Partner, mit dem du nun die Anweisungen testen kannst. Jeder von euch ist einmal Roboter und einmal derjenige, der die Anweisungen gibt. Beachte:

Anweisungen für den Roboter: Lasse dir die Augenbinde anziehen, so dass du nichts mehr sehen kannst. Führe die gehörten Anweisungen genauso aus, wie DU sie verstehst. Nachfragen kannst du nicht. Wenn du eine Anweisung nicht verstehst, dann beweg dich nicht. Warte mit der Ausführung der Anweisung bis du das Wort „**AUSFÜHREN**“ hörst. Jedes Mal, wenn du eine Anweisung ausgeführt hast, sagst du „**FERTIG**“.

Anweisung für den Vorleser: Lies deine Anweisung vor und beende jede Anweisung mit dem Wort „**AUSFÜHREN**“. Bevor du eine neue Anweisung vorliest, musst du warten bis der Roboter FERTIG meldet.

Welche Probleme gab es bei der Durchführung und wie kann man sie lösen?

b) Wir tauschen den einfachen Roboter gegen einen hochentwickelten Roboter, der den menschlichen Fähigkeiten sehr nahe kommt. Formuliere für einen solchen Roboter die Anweisung. Welche Sensoren bzw. Geräte braucht der Roboter von Aufgabe c)?

c) Ergänze den Lückentext mit folgenden Begriffen: Kamera, Gehirn, Sensoren, Muskeln, Motoren, Computer, Gehirn, Mikrophon.

Einem Roboter ohne _____ ist es nicht möglich seine Umwelt wahrzunehmen. Er braucht zum Beispiel eine _____, um sehen zu können oder ein _____, um hören zu können. Aber auch das Laufen will erst einmal gelernt sein. Die Steuerung übernimmt ein Computer. Der Computer ist das _____des Roboters. Ein zweibeiniger oder vierbeiniger Roboter hat verschiedene Motoren, die seine Beine bewegen. Damit ein Roboter laufen kann, müssen diese _____zur richtigen Zeit an- und wieder abgeschaltet werden und sich natürlich auch in die richtige Richtung drehen. Das alles steuert der _____ und muss dafür sehr viel beachten, dass der Roboter nicht einfach umkippt. Das geht uns Menschen übrigens auch ähnlich: als wir laufen gelernt haben, musste unser _____ auch erst mühsam lernen, welche _____in den Beinen wie bewegt werden müssen.

ELIZA

Öffne folgende Seite und starte dort ELIZA (http://bs.cyty.com/menschen/e-etzold/archiv/science/rat.htm). Experimentiere mit dem Programm, indem du versuchst eine Unterhaltung über die Tastatur zu führen. Gehe dabei auf die Fragen des Programms ein. Beantworte anschließend die folgenden Fragen:

a) Was passiert, wenn du zweimal hintereinander die gleiche Antwort eingibst?
b) Wie reagiert ELIZA, wenn du sie nach etwas Persönlichen befragst?
c) Kannst du ein System hinter den Antworten von ELIZA erkennen?
d) Wie intelligent ist ELIZA?

ELIZA ist ein von Joseph Weizenbaum entwickeltes Computerprogramm. Das Programm simuliert oberflächlich einen Psychotherapeuten als Gesprächspartner, indem es die eingetippten Sätze klassifiziert. Als Joseph Weizenbaum sein Programm 1966 schreibt, glaubt er, dass jeder sofort merken würde, dass er sich mit einem Computer unterhält. Doch dann stellt er fest, dass seine Sekretärin sich mit dem Programm beschäftigt und intime Details mit dem Computer besprach. Daraufhin veröffentlichte er 1976 sein Buch „Die Macht der Computer und die Ohnmacht der Vernunft“. ELIZA funktioniert nach einigen wenigen Grundideen:

- Die Aussagen des menschlichen Gesprächspartners werden in eine Frage umformuliert, um ein Interesse am Gesagten vorzuheucheln.
- Schlüsselwörter, wie z. B. Vater werden erkannt und in Beziehung gesetzt. „Erzählen Sie mir etwas über ihre Familie.“

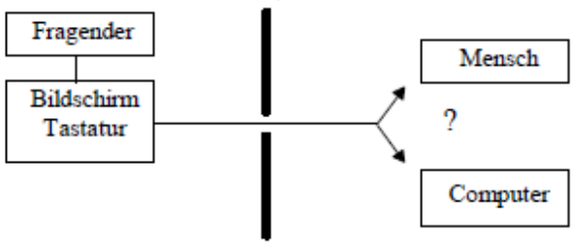
Alan M. Turing

a) Lies den Text und beschreibe mit eigenen Worten den Intelligenz-Test von Alan M. Turing!
b) Ist ELIZA nach Turing intelligent ist?

Alan Mathison Turing (* 23. Juni 1912 in London; † 7. Juni 1954 in Wilmslow) war ein britischer Logiker, Mathematiker und Kryptoanalytiker und legte die theoretischen Grundlagen für die moderne Informations- und Computertechnologie.
Turing gilt heute als einer der einflussreichsten Theoretiker der frühen Computerentwicklung und Informatik. Das von ihm entwickelte Berechenbarkeitsmodell der Turingmaschine bildet eines der Fundamente der theoretischen Informatik.
Während des Zweiten Weltkrieges war er maßgeblich an der Entzifferung der mit der Enigma verschlüsselten deutschen Funksprüche beteiligt. Der Großteil seiner Arbeiten blieb nach Kriegsende jedoch unter Verschluss. Turing entwickelte 1953 eines der ersten Schachprogramme, dessen Berechnungen er mangels Hardware selbst durchführte. Nach ihm wurde der Turing-Preis benannt, die bedeutendste Auszeichnung in der Informatik, sowie der Turing-Test zum Nachweis künstlicher Intelligenz.

Der Turing-Test

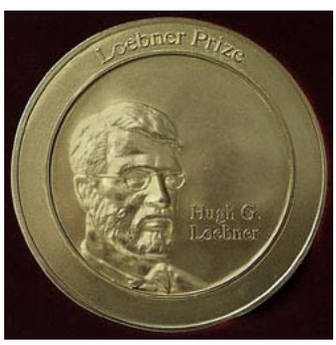
Die Beurteilung der Intelligenz einer Person ist sehr schwierig - insbesondere, da der Begriff Intelligenz sich nur sehr schwer definieren lässt. Dies gilt entsprechend für die Frage nach der Intelligenz einer Maschine. Die Problemstellung wurde von dem Mathematiker Alan Turing (1950) letztlich in folgende Frage umgewandelt: „Lässt sich das (Sprach-) Verhalten von Menschen und Maschinen unterscheiden?“ Alan Turing schlug so vor, über ein Experiment eine Entscheidung zu finden, ob eine Maschine intelligent ist oder nicht. Beim Turing-Test wird ein Computer zusammen mit einem menschlichen Freiwilligen vor den Blicken einer Versuchsperson oder einer Jury versteckt. Die Versuchsperson muss in einem Gespräch herausfinden, wer von beiden der Computer und wer der Mensch ist. Die Gesprächsbeiträge werden jeweils auf unpersönliche Weise übertragen, zum Beispiel per Tastatur und Bildschirm.

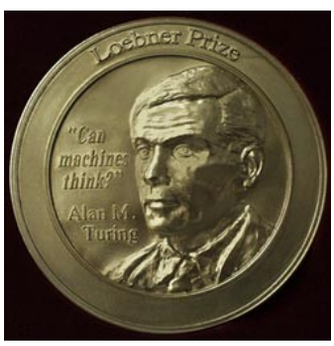


TURINGS VORSCHLAG: Wenn bei einem Gespräch über Tastatur und Bildschirm die Versuchsperson vor dem Bildschirm nicht herausbekommt, ob der Dialogpartner ein Mensch oder ein Computer ist, dann ist der Computer intelligent.

Was ist der Loebner-Preis?

Im Jahr 1990 stiftet der Soziologe, Erfinder und NewYorker Geschäftsmann Hugh Gene Loebner einen Preis, der nach ihm benannt ist. Der Hauptpreis besteht aus einer Goldmedaille und 100.000 $, einer Silbermedaille mit einer Prämie von 25.000 $ und einer Bronzemedaille mit einem Preisgeld von 2.000 $. Seit 1991 wird in jedem Jahr ein Wettbewerb durchgeführt.





(Loebner.net, 2012)

Bei dem Wettbewerb müssen die Programme sich einem Chat mit einem menschlichen Prüfer stellen. Der Prüfer weiß nicht, ob am anderen Ende der Datenleitung ein Mensch oder ein Chatbot sitzt. Nach fünf Minuten entscheidet er, ob er sich mit Mensch oder Maschine unterhält.

Die Goldmedaille erhält das Programm, das die Hälfte der Preisrichter nach 5 Minuten überzeugt, ein Mensch zu sein – mit Grafik und Sound! Die Silbermedaille gibt es, wenn die Ein- und Ausgabe nur über Texte erfolgt. Die Bronzemedaille geht an das „menschliche“ Programm mit den meisten Punkten. Bislang wurden nur Bronzemedaillen vergeben.

Für den Loebner-Wettbewerb haben die meisten Wissenschaftler nur Spott übrig: Beim gegenwärtigen mangelhaften Forschungsstand sei von der Software bestenfalls primitive Effekthascherei zu erwarten – aber auch bedeutende Wissenschaftler wurden von ihren Zeitgenossen verspottet!

Welche Programme gewannen in den letzten Jahren den Loebner-(Trost)-Preis?

2001 mussten die Juroren beim Loebner-Preis sieben Programme beurteilen, zusätzlich waren auch zwei menschliche Kommunikationspartner dabei. Jeder Juror musste ein 5.Minuten-Gespräch mit jedem Programm und den beiden menschlichen Kommunikationspartnern führen. Von den Computerprogrammen hat – wie schon im Jahr 2000 – ALICE von Dr. Richard Wallace am besten abgeschnitten (14 Punkte), die beiden menschlichen Gesprächspartner erhielten (24 bzw. 19 Punkte). Von einigen Juroren wurde ALICE besser als einer der beiden Menschen beurteilt; insgesamt hat die Jury aber Computer und Mensch noch deutlich unterscheiden können. Die Dialoge der Prüfer können im Internet (http://loebner.net/Prizef/2001_Contest/loebner-prize-2001.html) nachgelesen werden.

Im Jahr 2002 gewann ein Programm namens ELLA; ALICE belegte diesmal nur den 3. Platz. Leider ist ELLA nicht im Internet zu erreichen. Im Jahr 2003 gewann jabberwacky von Jürgen Pirner aus Hamburg; ALICE rutschte auf Platz 9 ab, um dann 2004 in verbesserter Version wieder zu gewinnen. In den Jahren 2005 und 2006 gewann Rollo Carpenter, im Jahr 2007 bekam Robert Medeksza den Preis, 2008 gewannen Fred Roberts und Artificial Solution die Bronzemedialle mit Elbot. Aus diesem Haus stammt auch der Chatbot Anna von IKEA. Elbot gibt ironischerweise gar nicht vor, ein Mensch zu sein, sondern stellt sich als Roboter vor. Dies war aber so überzeugend, dass ein Viertel der Preisrichter glaubten, am andern Ende der Leitung säße ein Mensch. In den Jahren 2010 und 2011 gewann den Trostpreis Rosette von Bruce Wilcox (http://labs.telltalegames.com/rosette).

Der Minsky Loebner Prize Revocation Prize

Marvin Minsky, so wie auch viele andere KI-Forscher, lehnen den Loebner-Preis als billige Effekthascherei ab. Minsky hat daher 1995 ein Preisgeld von 100 $ für denjenigen ausgelobt, der Loebner den Wettbewerb ausreden kann. Loebner hat darauf so reagiert, dass er Minsky zum Förderer seines Preises gemacht hat, da der Loebner-Preis nur beendet werden kann, wenn die Goldmedaille vergeben wird. Somit erhält derjenige, der die Goldmedaille gewinnt, sowohl das Preisgeld als auch die 100 $ von Minsky.

Aufgaben:

1. Wie erfolgt der Test der Chatbots?
2. Wofür wird Gold, Silber und Bronze vergeben?
3. Wie erfolgreich waren die Programme, die sich bisher um den Loebner-Preis beworben haben?
4. Wie wird er Preis von Wissenschaftlern gesehen?
5. Wodurch lassen sich ALICE oder ELBOT überführen?
6. Kann der Turing-Test eine Antwort darauf geben, ob Maschinen denken können?
7. Welche anderen Möglichkeiten sind deiner Meinung nach denkbar, um die Intelligenz einer Maschine nachzuweisen? Was ist Intelligenz?

Expertensysteme

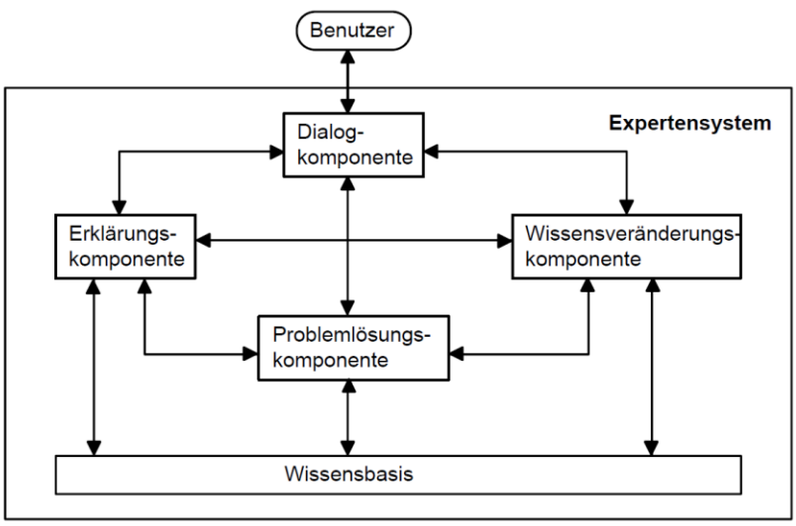
Definition

Ein Expertensystem ist ein Programmsystem, das „Wissen“ über ein spezielles Gebiet speichert und ansammelt, aus dem Wissen Schlussfolgerungen zieht und zu konkreten Problemen des Gebietes Lösungen anbietet.

Expertensysteme können demnach:

- große Mengen Wissen repräsentieren
- aus dem Wissen auf logischem Wege Schlussfolgerungen ziehen und neues Wissen gewinnen
- im Benutzerdialog zu gegebenen Problemen Lösungen finden und den Lösungsweg erläutern

Prinzipielle Struktur eines Expertensystems

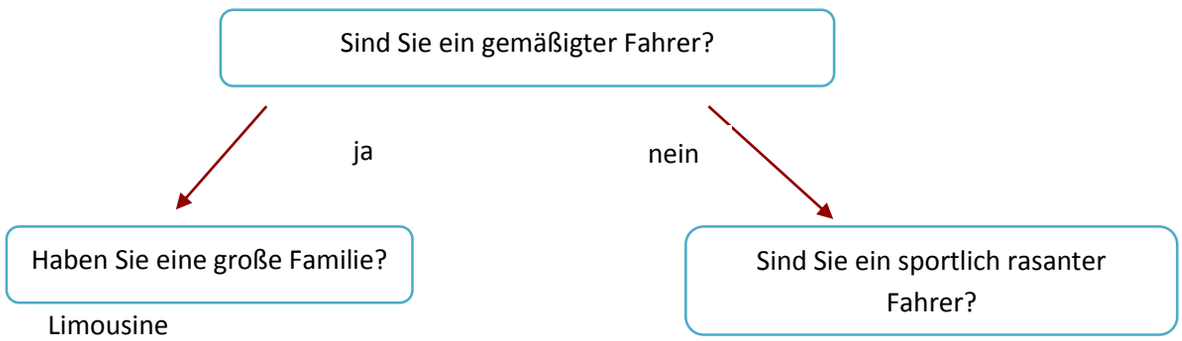


Die Wissensbasis enthält dabei zunächst einmal alle Fakten und Regeln und bildet somit die Grundlage (vgl. Prolog). Zusätzlich gibt es im Experten die Problemlösekomponente, die eine Anfrage vom Benutzer auswertet, indem Fakten und Regelns nach einer vorgegebenen Strategie miteinander verknüpft werden, so dass ein oder mehrere Ergebnisse geliefert werden können. Diese Ergebnisse werden dann dem Anwender mithilfe der Erklärungskomponente erläutert, so dass der Weg dorthin deutlich wird. Die Dialogkomponente begleitet den Lösungsprozess, indem die richtigen Fragen an den Benutzer weitergeleitet werden, so dass das Expertensystem alle notwendigen Informationen vom Anwender erhalten kann. Damit das Expertensystem hinzulernen kann, benötigt es noch eine Wissensveränderungskomponente, so dass der Wissensbasis neue Erkenntnisse hinzugefügt werden können bzw. Änderungen von Fakten und Regeln aufgenommen werden können. Ein Expertensystem ist daher keine reine Datenbank, die Wissen zur Verfügung stellt. Ein Expertensystem zieht Schlussfolgerungen aus Regeln und Fakten, ist lernfähig und selbsterklärend.

Test: Kaufberatung für ein Auto

Im Prologprogramm „Autos.pl“ ist ein Expertensystem für die Kaufberatung eines Autos dargestellt.

a) Gib zunächst unterschiedliche Antworten ein, um das Expertensystem kennen zu lernen.
b) Ergänze nun den Baum für das Programm (auf einem DinA3-Blatt!!!)



c) Ergänze das Programm so, dass der Bediener einen Vorschlag erhält, wenn er
   a. sich als flott, aber nicht sportlich rasant bezeichnet.
   b. nach einem günstigen Sportwagen sucht.
d) Verändere das Programm so, dass der Ferrari exklusiv und der Porsche nicht exklusiv angeboten wird.
e) Fülle die Tabelle aus: (im Heft)

Fakten	Regeln

f) Erläutere den Programmablauf, wenn die Anfrage berate. Eingegeben wird.

ZIEL: Erstelle ein Werkstatt-Expertensystem in PROLOG

```
/*WISSENSBASIS*/
fakt(f(a,a)).
/*Form von Fakten im Arbeitsspeicher, z.B. fakt(hat(licht,ja)).*/

regel(1, hat(treibstoff, nein),
  [reagiert(springtan, nein),
  hat(licht, ja),
  zeigt(tankleer, ja)]).

regel(2,hat(defekteBatterie,ja),
  [hat(licht, nein),
  reagiert(springtan, nein)]).

regel(3,hat(defekteGluehkerzen,ja),
  [ist(diesel,ja),
  hat(defekteBatterie, nein), /*alternativ: hat(licht, ja) */
  hat(treibstoff, ja),
  reagiert(springtan, nein)]).

regel(4,hat(defekteZuendkerzen,ja),
```

```
  [ist(diesel, nein),
  hat(defekteBatterie, nein),
  hat(treibstoff, ja),
  reagiert(springtan, nein)]).

fragbar(springtan, 'Springt der Wagen an').
fragbar(licht, 'Funktioniert das Licht').
fragbar(tankleer, 'Zeigt die Tankanzeige einen leeren Tank an').
fragbar(diesel, 'Ist der Wagen ein Diesel').

/*KOMMUNIKATIONSMÖGLICHKEITEN */
nenneGrund(Fakt, Warum) :-
  asserta(fakt(Fakt)),            /* Fakt speichern */
  regel(RegelNr,Warum,Bedingungen), /* Regel aufrufen */
  enthalten(Fakt,Bedingungen),    /* Falls der Fakt Teil der Bedingungen,*/
  pruefe(RegelNr,Bedingungen),    /* sind die Bedingungen zu testen */
  !.

beweise(Defekt) :-
  regel(Nr,Defekt,Bedingungen),    /* Suche entsprechende Regel */
  pruefe(Nr,Bedingungen), write('bestätigt')./* Überprüfe die Bedingungen*/

enthalten(Fakt,[]) :- !, fail.
enthalten(Fakt,[Fakt|R]) :-!.
enthalten(Fakt,[B|R]) :-              /* enthalten(Fakt,[_,R] :- */
enthalten(Fakt,R).

/* Prüfe, ob alle Bedingungen der Liste erfüllt sind */
pruefe(RegelNr,[]).
pruefe(RegelNr,[F1|Fakten]) :-
  F1 =.. L,
  bestaetige(RegelNr,L),!,
  pruefe(RegelNr,Fakten).

bestaetige(Nr,[P,Frage,B]) :- /* liegt bereits als Fakt vor */
  /*HIER FEHLT NOCH ETWAS */

bestaetige(RegelNr,[P,Frage,B]) :- /* es soll gefragt werden */
  fragbar(Frage, Text),!,
  /*HIER FEHLT NOCH ETWAS */

/* Bestätige, dass eine Bedingung erfüllt ist oder auch nicht */
bestaetige(_,[P,Frage,B]) :-   /* Regelueberpruefung */
  T1 =.. [P,Frage,Bool],       /* Bestätigung pos/neg*/
  regel(Nr,T1,Liste),
  asserta(fakt(T1)),
  pruefe(Nr,Liste),!,
  gleich(B,Bool).

bestaetige(_,[P,Frage,B]) :- /* Regelueberpruefung */
```

```
  T1 =.. [P,Frage,Bool], /* Fehlschlag */
  fakt(T1),
  retract(fakt(T1)),
  kontra(Bool,Bk),
  T2 =.. [P,Frage,Bk],
  asserta(fakt(T2)), gleich(Bk,B).
  gleich(A,A).
  kontra(ja,nein).
  kontra(nein,ja).

/* Testaufrufe: */
go(X) :- nenneGrund(reagiert(springtan,nein),X).
do :- beweise(hat(defekteZuendkerzen,ja)).
```

Aufgaben:

1. Es gibt typische Anwendungsbereiche von Expertensystemen, z. B. in der Medizin, bei der Steuerung von Maschinen, mathematische Beweisführungen oder auch in der Konfigurierung von Computersystemen. Welche Vor- und Nachteile kann die Anwendung eines Expertensystems bieten? Wann kann der Einsatz problematisch sein? Welche?

Literaturverzeichnis

Asteroth, A., & Baier, C. (2002). *Theoretische Inforamtik; Eine Einführung in Berechenbarkeit, Komplexität und formale Sprachen mit 101 Beispiel.* München: pearson-studium.

Battenfeld, G., & u.a. (06 1996). Theoretische Informatik. *Planung eines Kurses in der Jahrgangsstufe 13 I*. Weilburg.

Breier, P. D. (kein Datum). Algorithmisch lösbare und unlösbare Probleme. *Empfehlungen für eine Unterrichtseinheit zum Wahlthema "Theoretische Informatik" in der Jahrgangsstufe 13*. Greifswald.

Burkert, J., Lächa, R., & Meyer, D. M. (Version 1.000001). *Datenbanken in der Sekundarsteufe II; Theorie und Praxis.*

Datenbanken im Informatikunterricht. (2013). Von http://dbup2date.uni-bayreuth.de/ abgerufen

Dr. Engelmann, L. (Hrsg.). (2006). *Duden; Informatik; Lehrbuch S II.* Berlin: duden-paetec.

Gallenbacher. (2008). *Abenteuer Informatik.* Heidelberg: Spektrum.

Helbig, H. (1996). *Künstliche Intelligenz und automatische Wissensvermittlung.* Berlin: Verlag Technik GmbH.

Heusel, H. (kein Datum). *Präsentation: Der Einsatz von MySQL-Datenbanken (mit XAMPP).* Von http://informatik.bildung-rp.de/fileadmin/user_upload/informatik.bildung-rp.de/Fortbildung/FB_Wahlfach/WF-120214-Heusel-MySQL_Praesentation.pdf abgerufen

Hubwieser, P., & a., u. (2010). *Informatik 5; Lehrwerk für Gymnasien.* Stuttgart: Klett.

Kempe, T., & Löhr, A. (Hrsg.). (2012). *Informatik 2, Modellierung, Datenstrukturen und Algorithmen.* Braunschweig: Schöningh.

Kempe, T., & Tepaße, D. (Hrsg.). (2010). *Informatik 1; Softwareentwicklung mit Greenfoot und BlueJ.* Paderborn: Schöningh.

Loebner.net. (19. 02 2012). Von http://www.loebner.net/Prizef/loebner-prize.html abgerufen

Magenheim, J. u. (2009). *Informatik macchiato; Cartoon-Informatikkurs für Schüler und Studenten.* München: Pearson Studium.

Matzke. (2000). *Datenbanken.* Augsburg.

(Nr. 128/129). *LOG IN*, S. 93ff.

Preckel, E. (2012). *JAVA; Einstieg in das objektorientierte Programmieren.* Berlin: Cornelsen.

Rau. (07. 09 2014). Von http://ww4w.herr-rau.de/wordpress/2010/01/das-wasserfallmodell.htm abgerufen

Röhner, G. (03 2003). *Zur Rolle der Sprache bei der Modellierung von Datenbanken.* Präsentation.

wikipedia.org. (21. 09 2014). Von http://de.wikipedia.org/wiki/Kellerautomat abgerufen

wikipedia.org. (05. 02 2014). Von http://de.wikipedia.org/wiki/K%C3%BCnstliche_Intelligenz abgerufen

Zabel, F., & Hempel, T. (2000). Expertensysteme, Seminar zur Didaktik der Informatik. Greifswald.