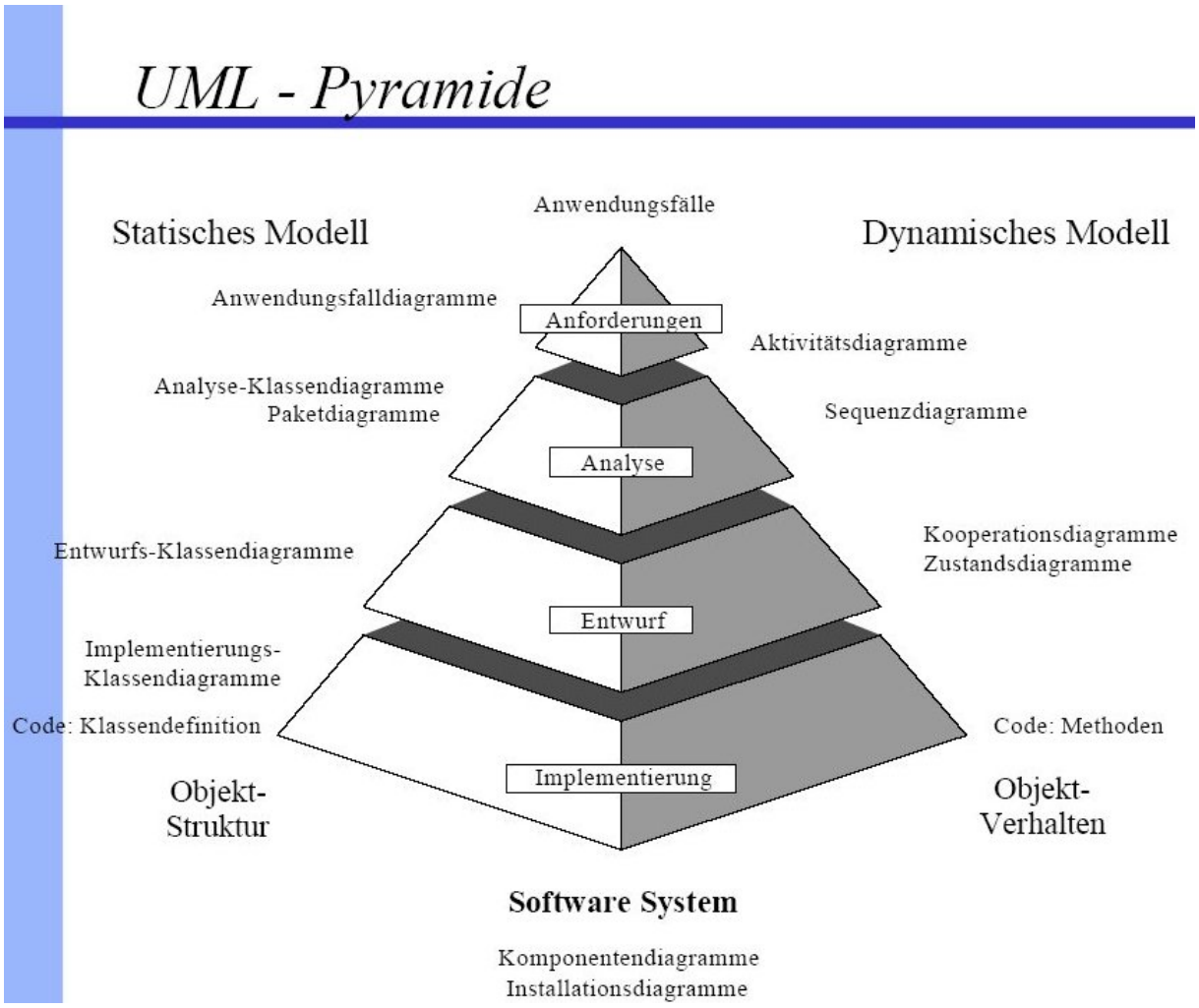
1 Programmiersprachen und -paradigmen

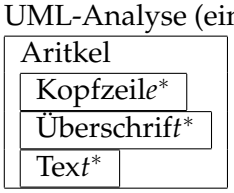
1.1 Grundlagen von Programmierung und Modellierung

1.1.1 Statische oder Strukturanalyse

Anforderungsanalyse ↔ Analyse ↔ Entwurf ↔ Implementierung ↔ Test



UML-Analyse (eines Artikels):



1.1.2 Grammatik

- Grammatik: Menge von Regeln, die bestimmen, welche Sätze zur Sprache gehören
  - Terminalsymbole (T): Zeichen (was man sieht)
  - Variablen (Nichtterminale N): werden durch T ersetzt
  - Regeln (R): Ersetzungsregeln für alle Variablen
  - Startsymbol (S): ausgezeichnete Variable
- EBNF: Erweiterte Backus-Naur-Form

  Zur Darstellung der Syntax (kontextfreier Grammatiken)

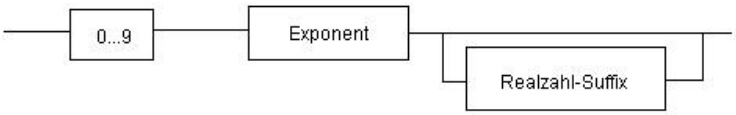
  <ZifferAußerNull> ::= ”1” | ”2” | ”3” | ”4” | ”5” | ”6” | ”7” | ”8” | ”9” ;

  <Ziffer> ::= ”0” | ZifferAußerNull ;

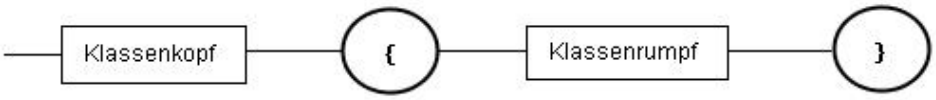
  <NatuerlicheZahl> ::= ZifferAußerNull (Ziffer)*

x	x?	x^*	x^+	x|y
1	0 oder 1	0 bis ∞	1 bis ∞	x oder y

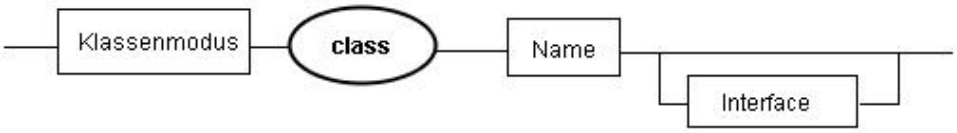
- Syntaxdiagramm



Klasse:



Klassenkopf



1.1.3 reguläre Ausdrücke

dienen der Beschreibung von Zeichenketten

- Gleitkommazahl GKZ
  $[0-9]([0-9]^*)|\epsilon$. $[0-9]([0-9]^*)|\epsilon$ $((e[0-9]([0-9]^*))|\epsilon)$
- Zeichenklassen
  * bel Zeichen
  $\backslash n$ Zeilenende
  \\ \
  [abc] = a | b | c
- Zeilenrandmarkierung
  ^Anfang
  $ Ende
- Quantoren

x^*	x^+	x(n)	x(n,m)	x(n,)
0 bis ∞	1 bis ∞	genau n-mal	zwischen n und m mal	min m-mal

x?	x|y
optional	Alternative

- auch regexp in Java

1.1.4 Markieren

farbig oder mit Marken

1.1.5 Ablaufmodellierung

- Szenario beschreibt einen Vorgang als Folge von Aktivitäten
- Grammatik
  Szenario ::= Aktivitäten*
  Aktivitäten ::= einfache Aktivität | bedingte Aktivität | Verzweigung
  bedingte Aktivität ::= Bedingung einfache Aktivität
  Verzweigung ::= Szenario | Szenario

1.1.6 Automaten

wie in Theoretische Informatik

1.2 Programmierparadigmen im Überblick

1.2.1 Imperativ

- Programm: Folge von Befehlen oder Anweisungen
- zentrales Konzept: Behältervariablen
- zeitliche Abfolge ist die Reihenfolge der Befehle
- Nachteile

  schlechte Erfassung von Problemen ohne offensichtlichen Algorithmus, unflexibel, schwer veränderbar
- Makros (Zeichenfolgen werden durch andere ersetzt)

  Hintereinander Ausführen von Kommandos (z.B. für PP)
- Skripte
  - Sprache wird nicht explizit kompiliert oder mit Bibliotheken verbunden
  - Typdeklarationen für Variablen freiwillig oder nicht möglich
  - Höhere Skriptsprachen (Perl, Python) bieten komplexe Datenstrukturen (z.B. Listen)

1.2.2 objektorientiert

- Objekte sind eigenständige Einheiten, die einen Zustand haben
- Objekte sind Exemplare einer Klasse, die gemeinsame Attribute und Aktionen zusammenfasst
- Programm:
  - Folge von Objekterzeugungen
  - Austausch von Nachrichten
  - Objekte zur Ausführung von Methoden
- Methode: Folge von Anweisungen
- Datenstruktur: Objekt, Datentyp: Klasse

1.2.3 funktional (Datenstrom-basiert)

- Programme als mathematische Funktionen
- keine Behältervariablen, kein Zustandbegriff
- Rekursion
- Vorteile: klare Semantik
- Nachteile: kein Gedächtnis

1.2.4 Relationales Programmieren

- Daten: Menge von Relationen
- Operationen: Selektion, Projektion, Natürlicher Verbund, Mengenoperationen
- Prolog (logik-basiert)
  Regeln dienen dazu aus Fakten neue Fakten herzuleiten
  Abfrage der Form Ja/Nein
  keine Planung des Kontrollfluss, kein Gedächtnis

1.3 Funktionale Programmierung

- Programme als mathematische Funktionen (Ausdrücke)
- kann man als partielle Abbildungen von Ein- und Ausgabedaten auffassen
- Funktionen def. durch
  - Elementarfunktionen
  - Funktionsaufruf
  - Alternative
  - Rekursion
- Jede Funktion liefert Ausgabewerte
- keine Behältervariablen, kein Zustandbegriff
- Jeder vollständige Teilausdruck kann getrennt vom Rest ausgewertet werden
- kommt ohne Schleifen und Zuweisungen aus
- Vorteile: klare Semantik
- Nachteile: kein Gedächtnis
- Java funktional:
  - Verwende nur eine Klasse
  - Funktionen public static, Parameter nur final
  - verwende nur return Anweisung
  - keine Schleifen, sondern Rekursion
  - keine lokalen Variablen

1.4 Imperative Programmierung

- Programm: Folge von Befehlen oder Anweisungen
  - Zuweisungen
  - Verzweigungen
  - Schleifen
- zentrales Konzept: Behältervariablen (Zustand des Programms)
- Charakteristisch: Zuweisugen (Änderungen von Werten)
- Datenkapselung(in Module oder Klassen)
- zeitliche Abfolge ist die Reihenfolge der Befehle
- Vorteile
  - leichte Implementierung auf von-Neumann-Rechner
- Nachteile
  - schlechte Erfassung von Problemen ohne offensichtlichen Algorithmus,
  - unflexibel, schwer veränderbar

1.5 OOP

- konkrete Datentypen
  - Wertebereich uner Operationen
  - Klasse (Attribute und Methoden)
  - Datenkapselung
  - Klasseninvariante
- abstrakte Datentypen (unabhängig von konkreter Implementierung)
  - Interfaces
  - Listen (Zeiger auf linear verkettete Datenstruktur)

  Es werden Eigenschaft festgelegt, die für jede konkrete Implementierung gelten müssen (Axiome)

  Universalität

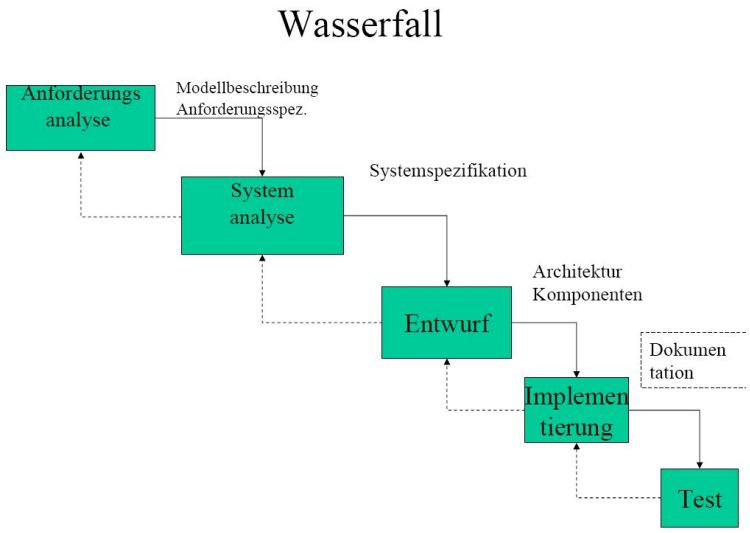
  Kapselung (was ADT tut)

  Präzise Beschreibung
- Objekte sind eigenständige Einheiten, die einen Zustand haben
- Objekte sind Exemplare einer Klasse, die gemeinsame Attribute und Aktionen zusammenfasst
- Programm:
  - Folge von Objekterzeugungen
  - Austausch von Nachrichten
  - Objekte zur Ausführung von Methoden
- Methode: Folge von Anweisungen
- Datenstruktur: Objekt, Datentyp: Klasse
- Vererbung:

  Spezialisierung, Generalisierung, Hierachien, Unterverträge
- es gibt Unter- und Oberklassen
- Unterklasse ist Spezialisierung, also Teilmenge und erbt alle Attribute und Methoden der Oberklasse
- ererbte Methoden können modifiziert werden
- Merkmal: wiederverwendbaren, weiterverwertbaren Programmcode

2 Objektorientierte Softwareentwicklung

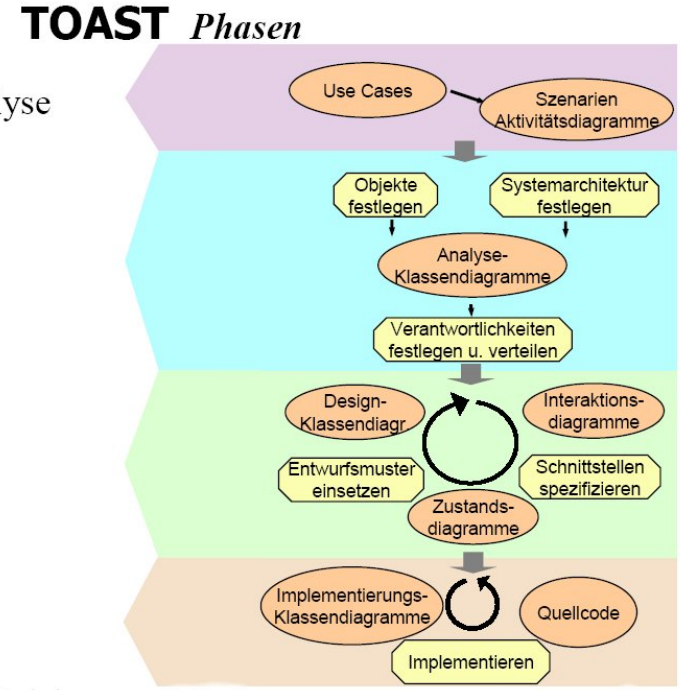
2.1 Pyramide



- System
  beliebiger Ausschnitt aus der realen Welt & Software-System
- Modellbildung (Software zu komplex um im Ganzen verstanden zu werden)
  - abstrahieren von Details
  - betrachten Teile, Aspekte der Software
  - Modelle für unterschiedliche Entwicklungsphasen
- statisches Modell
  - wer agiert
  - was es tut
  - welche Verantwortlichkeit es übernimmt
  - mit wem kooperiert es
  - von wem werden Dienste in Anspruch genommen
- dynamisches Modell
  - anwendungsbezogene, zustandsorientierte Sicht auf System
  - Zusammenspiel von Objekten
  - wie werden Verantwortlichkeiten ausgeführt
  - wann (Ablauf)
  - in welchem Kontext (Kooperation)

- Anford.analyse ↔ Analyse ↔ Entwurf ↔ Implementierung ↔ Test

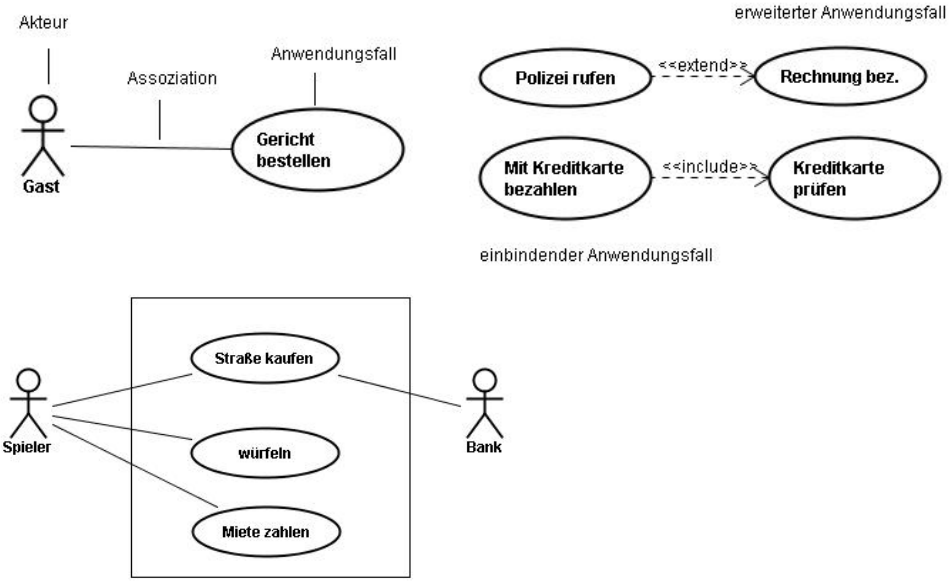
  ist iterativer Prozess (läuft mehrmals ab)
- TOAST (Traceable Object-oriented Approach for Software Tevelopment)



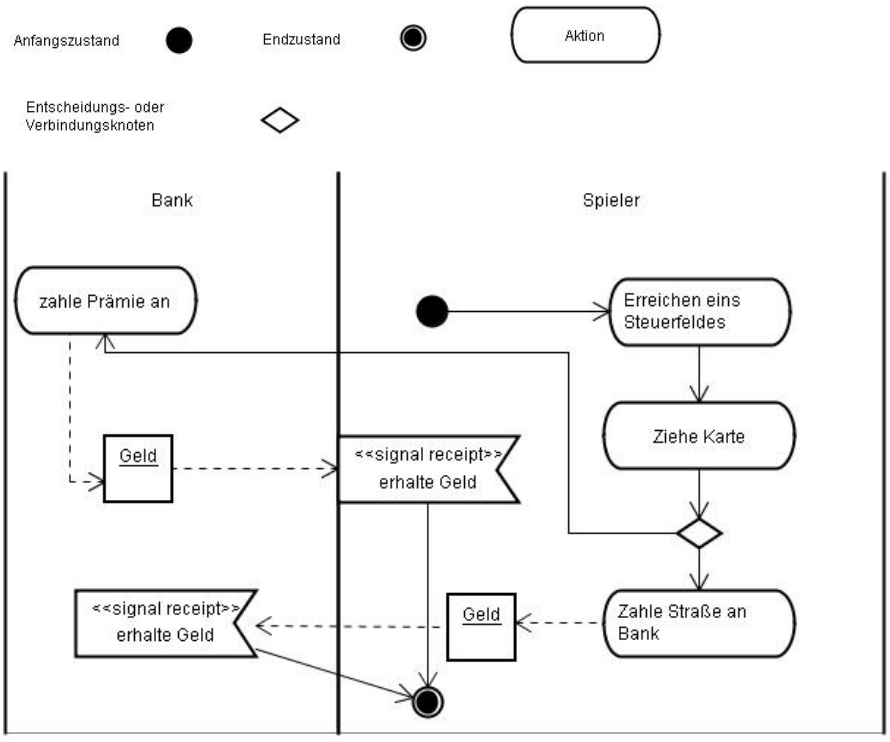
- Anforderungsanalyse
  - Use Cases (Szenario für Use Case beschreiben; textuell)
  - Aktivitätsdiagramm
  - ggf. Sequenzdiagramm
- Analyse
  - Lege Objekt fest, Klasse
  - Lege Verantworltichkeit fest
  - Analyse - Klassendiagramme
- Entwurf
  - Klassenentwurf
- Implementierung
  - Klassen und Methoden definieren

Struktur: Klassendiagramm
Verhalten: Use Case Diag, Aktivitätsdiag., Sequenzdiag., Zustandsdiag.

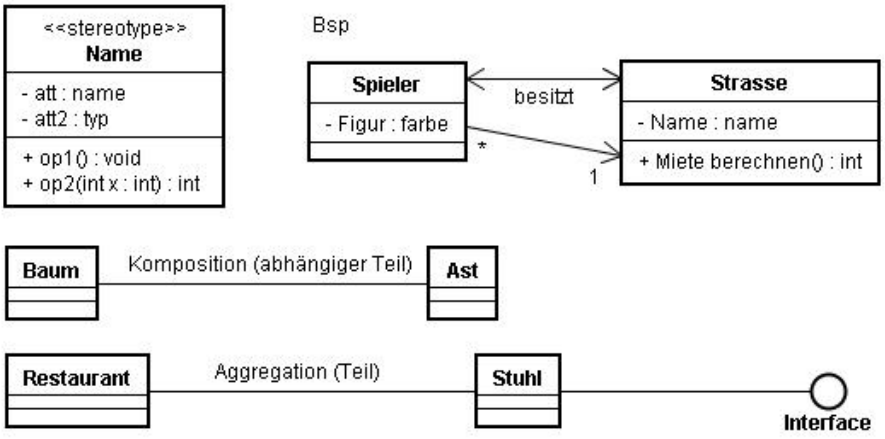
Use Cases (Anforderungsanalyse - statisch)



Aktivitätsdiagramme (Anforderungsanalyse - dynamisch)



Klassendiagramm (Analyse - statisch - Entwurf - dynamisch)

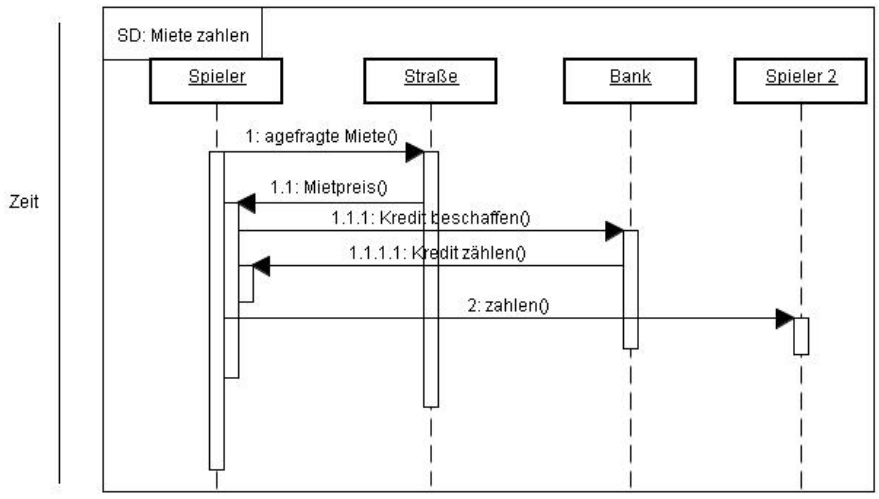


- Analyse statisch: Mit Methoden als Text
- Entwurf dynamisch: Methodennamen und Rückgabewerte (wie hier)

CRC: Class Responsibility Colaboration (Analyse - statisch)

- Responsibility: Verantwortlichkeiten der Klasse
- Collaboration: welche anderen Klassen dies betrifft

Sequenzdiagramme



2.2 OOA (Objektorientierte Analyse)

- Ziel
  Wünsche und Anforderungen des Auftraggebers ermitteln und beschreiben
  (Aspekte der Implementierung bewusst ausklammern)
- Analysephase
  - Pflichtenheft
  - OOA-Modell (Anwendungsfälle)
  - Prototyp der Benutzungsoberfläche
- Pflichtenheft
  1. Zielbestimmung (Kann, Muss-Kriterien)
  2. Einsatz
  3. Umgebung
  4. Funktionalität
  5. Daten
  6. Leistungen
  7. Benutzungsoberfläche
  8. Qualitätsziele
- statisches Modell
  - Anwendungsfälle
  - Klassen
  - Attribute
- dynamisches Modell
  - Geschäftsprozesse und Anwendungsfälle (Aufgaben)
  - Szenarios: wie Objekte kommunizieren
  - Aktivitätsdiagramme
- Trennung von Fachkonzept und Benutzeroberfläche
  Fachkonzept: welche Informationen
  Benuteroberfläche: in welchem Format

- Use Case (Geschäftsprozess)

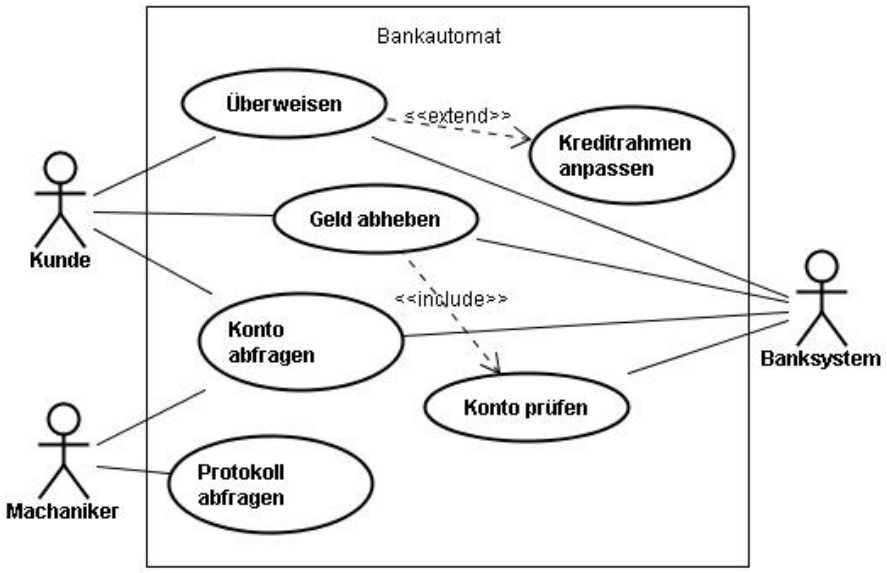
  besteht aus mehrern zusammenhängenden Aufgaben, die von einem Akteur durchgeführt werden, um ein Ziel zu erreichen

  Sequenz von zusamenngehörenden Transaktionen (Möglichkeiten der Benutzung des Systems)

  Akteur: hat Einluss auf System

  Definition:

  - Auslösendes Ereignis
  - Beschreibung (Aktionen: primär, sekundär, optional)
  - Erweiterungen der Aktionen
  - Alternativen



- Szenario

  Szenario := Aktiv*

  Aktiv := einfAktiv | bedAktiv | Verzweigung

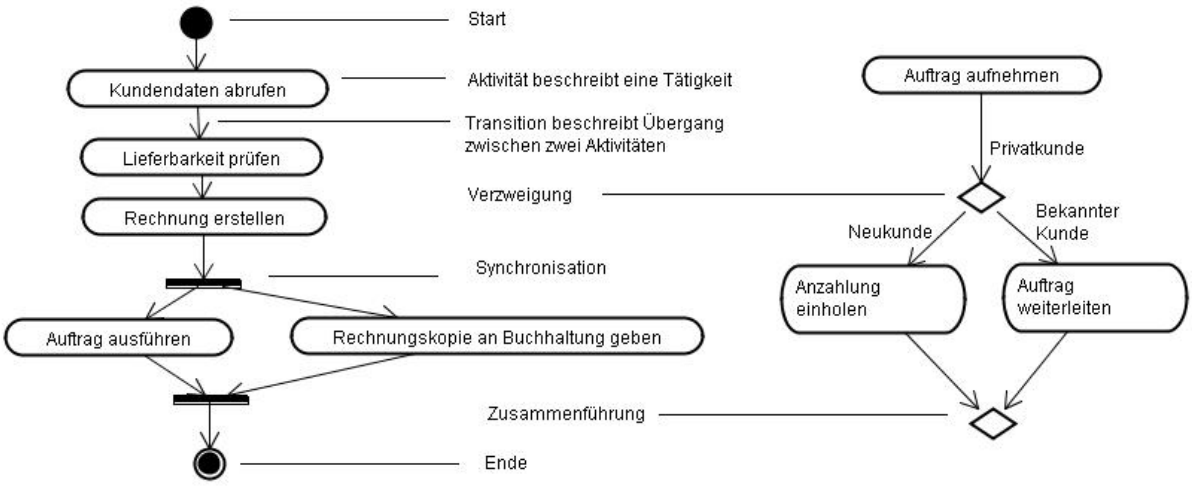
  bedAktiv := Bed einfAktiv

  Verzweigung := Szenario Szenario

  Szenario ist eine Sequenz von Verarbeitungsschritten (Hauptziel des Akteurs)

- Aktivitätsdiagramm

  Verhalten oder Ablauf eines UseCase



  Auch Weitergaben von Objekten zwischen Aktivitäten (als eckiges Kästchen)

  Abgrenzen der Aktivitäten durch Balken (in wessen Verantwortungsbereich Aktivität liegt)

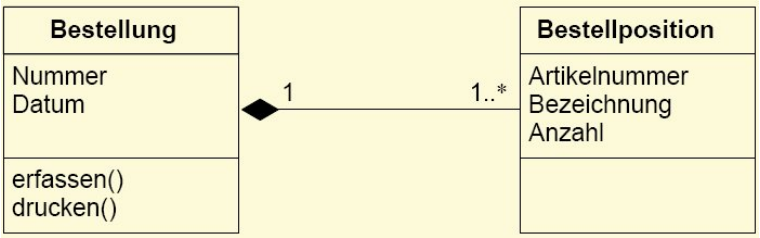
  Jeder Bahn werden Einheiten zu geordnet, die die Verantwortung wahrnehmen

- Paket

  fasst Modellelemente (z.B. Klassen) zusammen

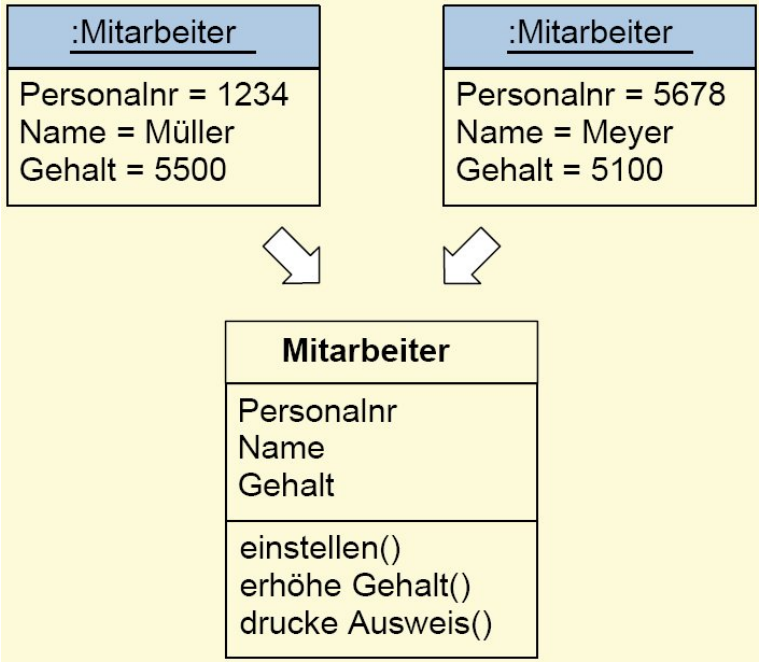
- CRC

Class Bestellung	
Responsibilities ■ verwaltet eine Bestellung ■ delegiert Aufgaben an Bestellpositionen	**Collaborations** ■ Bestellposition

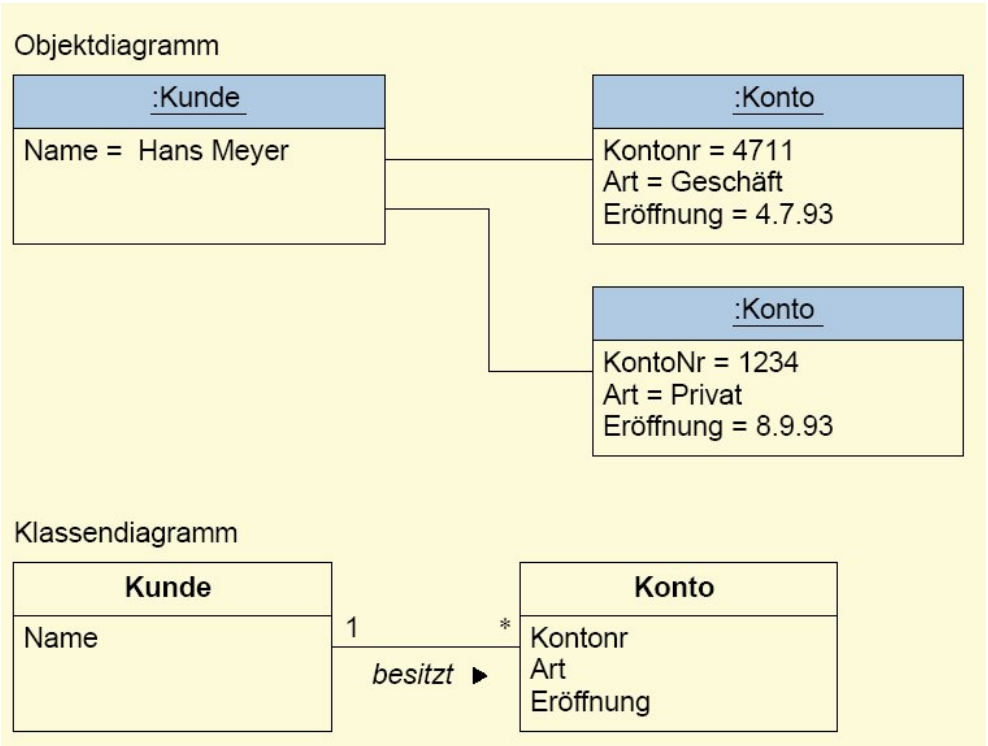


- Klasse

  Attribute - Operationen - Relationships



- Klassendiagramme

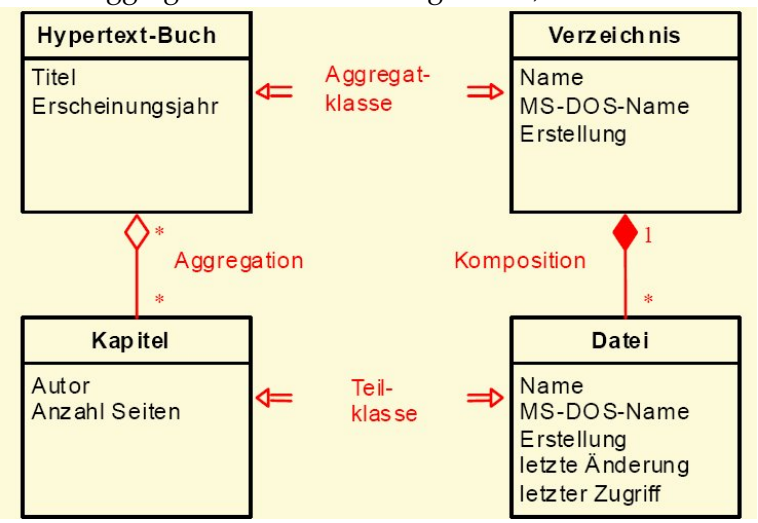


- Aggregation

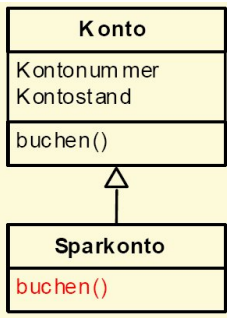
  ist Teil von bzw. besteht aus

- Komposition

  starke Aggregation: wird Ganzes gelöscht, werden automatisch seine Teile gelöscht

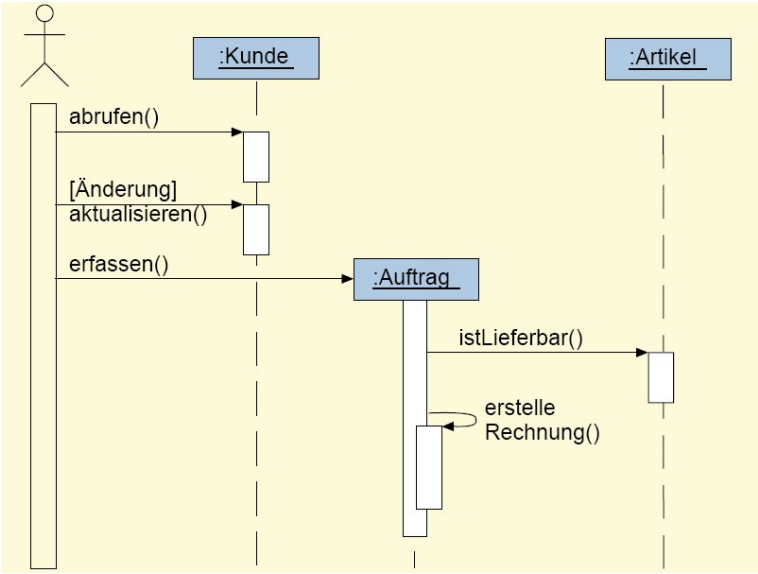


- Vererbung



- Sequenzdiagramme
  besitzt 2 Dimensionen
  - Vertikale: Zeit
  - Horizontale: Lebenslinie der Objekte (Aktionen, Austausch von Nachrichten)

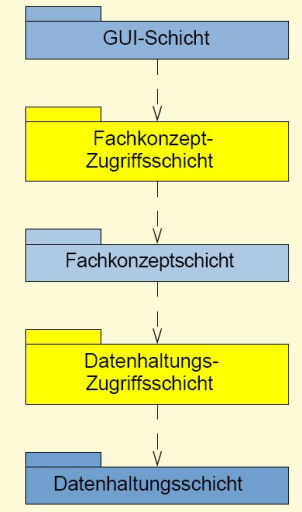
  Netzwerk von Objekten die untereinander Meldungen austauschen



2.3 OOE (Objektorientierter Entwurf)

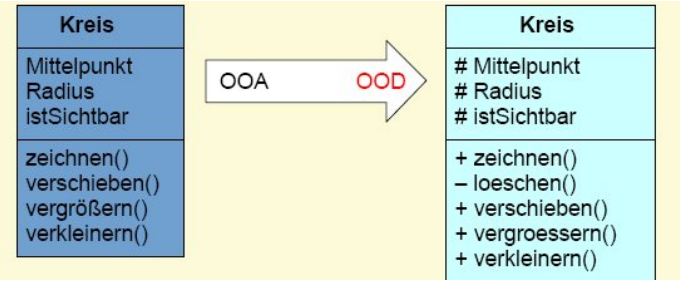
Entwurfsziel: Weitgehende Entkopplung von Fachkonzept, GUI und Datenhaltung

- 2-Schichten-Architektur
  Client-Server-Anwendung (Anwendungsschicht → Datenhaltungsschicht)
  Client: Anwendung, Dienst anfordern
  Server: warten, Ergebnis liefern, Dienst leisten
- 3-Schichten-Architektur
  (GUI-Schicht → Fachkonzeptschicht → Datenhaltungsschicht)
- Getrennte Entwicklung von Benutzungsoberfläche und Fachkonzept möglich
- MVC (Model / View / Controller)
  View: Präsentation der fachlichen Daten
  Controller: Reaktion auf Eingaben des Benutzers
  Model: Repräsentiert Fachkonzept
- Mehr-Schichten-Architektur

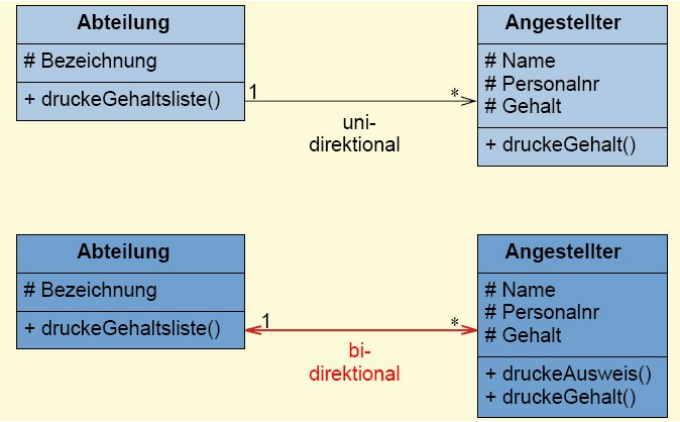


- Analyse → Entwurf (Klassendiagramme)
  - Klassen aus Analyse verfeinern
  - Verantwortlichkeiten verteilen
  - Attribute und Operationen festlegen
- Schnittstellen (Interfaces)
  Spezifiziert einen Ausschnitt aus dem Verhalten einer Klasse
  keine Attribute und abstrakte Operationen

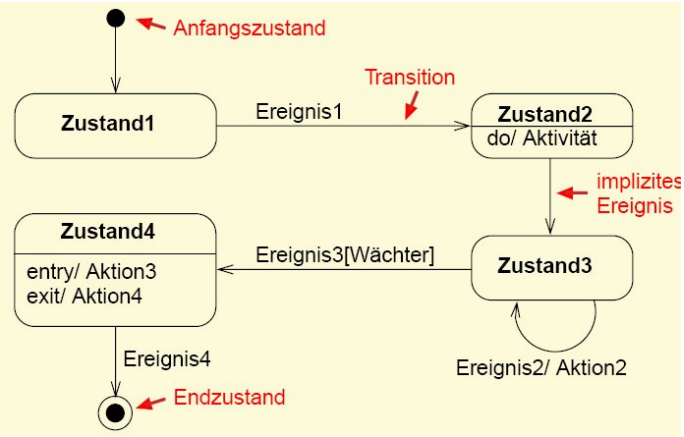
- Produkt der Entwurfsphase:
  - Abbild des späteren Programms
  - statisch: Klassen (inkl. Verantwortlk., Schnittstellen und Vererbung)
  - dynamisch: übersichtliche Beschreibung der komplexen Kommunikation zwischen Objekten
- Attribute und Operationeneigenschaften (+public, #protected, -private)



- Assoziationen



- Zustandsautomat



2.4 Analyse- und Entwurfsmuster

- Analysemuster

  beschreiben einfache wiederkehrende Situationen oder Strukturierungen

- Entwurfsmuster

  Bewährte generische Lösung für ein immer wiederkehrendes Entwurfsproblem, das in bestimmten Situationen auftritt

- Erzeugungsmuster

  - Fabrikmethode: (Factory)
    dynamische Kreation von Objekten (virtueller Konstruktor)
    Abstrakter Erzeuger, nur Schnittstelle, kein anwendungsspezifischer Code
    wenn Klasse ihr zu erzeugendes Objekt nicht kennen kann
    (Unterklassen legen fest, welches Objekt erzeugt wird)
  - Singleton: für Klasse existiert genau ein Objekt
    if(null == exemplar) exemplar = new Singleton(); return exemplar;

- Strukturmuster

  - Kompositum (Composite): Objekte zu komplexen Einheiten zusammenfassen
    Prinzip: rekursive Komposition (whole-part)
    andere Klassen können mit einheitlicher Schnittstelle auf Komponenten zugreifen
    (Erweiterbare Softwarestruktur)
  - Adapter: Passe Schnittstelle einer Klasse an eine andere an (Objekt oder Klasse)
    existierende Klassen sollen benutzt werden, passen aber nicht zusammen (Flexibilität)
    Mehrfachvererbung

- Verhaltensmuster
  - Schablonenmethode: Ausnutzen von Polymorphie (Vielgestaltigkeit)
    Methode, die nur andere Methoden verwenden; Invarianten festlegen
    invariante Teile eines Algorithmus genau einmal festlegen
    konkrete Ausführung wird Unterklassen überlassen
    Vermeidung von Duplikationen von Code
  - Beobachter (Observer): mehrere Objekte sind von anderem abhängig
    Synchronisation jeden Beobachters mit dem Zustand des Subject (ActionListener)
    Automatische Nachrichtenverteilung an alle Observer
    flexible, erweiterbare Software
    Push-Modell: Subject schickt Observer Änderungsinfo
    Pull-Modell: Observer müssen Änderungsinfo erfragen
  - Zustand (State): Verhalten abhängig von Zustand
    (mehrere Klassen nötig; Stack: EmtpyStack, StackInUse, FullStack)
    Zustandsspezifisches Verhalten wird in Klassen zusammengefasst (modellnah)
  - Strategie (Strategy): unterschiedl Algorithmen für gleiche Aufgabe
    Familie von Algorithmen
  - Iterator: Schnittstelle zum Zugriff auf Elemente (z.B. Vector)
- Frameworks: konkrete und abstrakte Klasse

  (Architektur der Anwendung, Zusammenarbeit der Objekte und Klassen, Kontrollfluss)

  spezifisch auf Anwendungsbereich

  Entwurfsmuster sind abstrakter als Frameworks

  typisches Framework enthält mehrere Entwurfsmuster
- Zusammenfassung: Muster (Patterns) ermöglich Wiederverwendung von gelösten Teilproblemen

  komplexe Zusammenhänge in Software besser nachvollziehbar

3 Hypertextsysteme – Internetprogrammierung

3.1 Dokumente modellieren mit XML und DTD

- Single Source Prinzip: write once - read everytime
- eXtensible Markup Language (XML)
  - Markierungssprache
  - textuelles Datenaustauschformat
  - keine Programmiersprache oder Protokoll
- Beispiel:

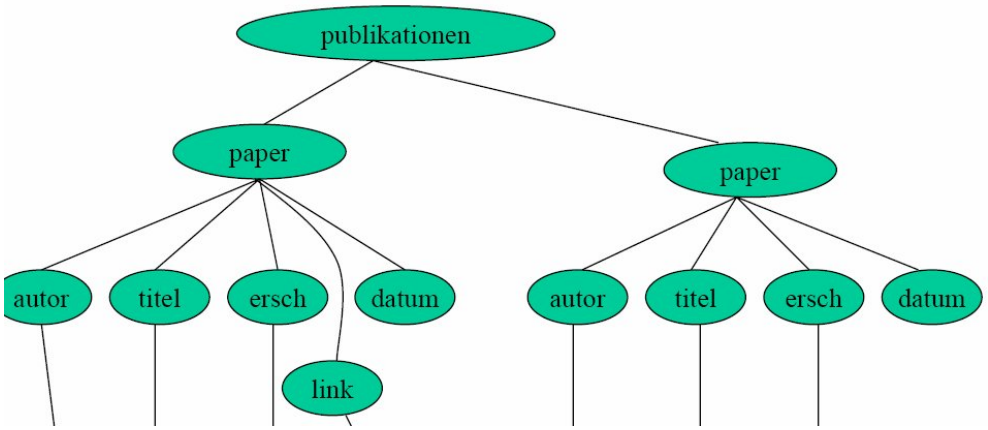
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE pub SYSTEM "wolff/pub.dtd" >
<?xml-stylesheettype= "text/xsl" href= "wolff/pub.xsl" ?>
<publikationen>
<paper typ= "report">
        <autor> H. Eichelberger, G. Tischler, J. WvG</autor>
        <titel> Comprehensive Graphical Description of the STL </titel>
        <ersch> Inst. f. Informatik, Würzburg TR 270 </ersch>
        <datum jahr= "2001" monat= "Feb" />
        <link> http://www/reports/stlAbstract.html</link>
</paper>
</publikationen>
```

- DTD Document Type Defintion

  legt Struktur eines XML-Dokuments fest(Entitys, Elemente und Attribute)

```
<!DOCTYPE publikationen [
<!ELEMENT publikationen(paper*)>
<!ELEMENT paper(autor,titel,ersch, link?, datum)>
<!ATTLIST paper typ(report|proc|journal) #REQUIRED>
<!ELEMENT autor(#PCDATA)>
<!ELEMENT titel(#PCDATA)>
<!ELEMENT ersch(#PCDATA)>
<!ELEMENT datum EMPTY>
<!ATTLIST datum jahr NMTOKEN 2001 monat CDATA #IMPLIED >
<!ELEMENT link (#PCDATA)>
]
```

- Baum



- Dokumentmodellierung
  - Analyse, Architektur, Entwurf
  - Element oder Attribut
  - Aufbau von Hierarchien (Baumdarstellung)
  - Inhalts- oder darstellungsbezogene Tags
- XML Syntax

  document:= prolog element misc*

  element:= <name1 attribut* (/> | > (text | element | &entityname;)* </name1>)

  attribut:= attributname= ”string”

  prolog:= <?xmlversion= ”1.0” (standalone= ”(yes|no)”)? ?>

  (<!DOCTYPE dtdname (SYSTEM ”url” | PUBLIC name url)([markupdecl*])?>)?

  misc:= (<!–beliebigerKommentar–>)

  |(<? targetname processing-instruction?>)

  markupdecl:= <!ELEMENT name contentspec>

  |<!ATTLIST elemname attdef*>

  |<!ENTITY name ”&(entname|char);”>

  contentspec:= EMPTY |ANY |mixed|children

  mixed:= (#PCDATA (| elemname)*)* | (#PCDATA)

  children:= (choice | seq)(? | * |+)?

  choice:= (cp(| cp)*)seq:= (cp(, cp)*)

  cp:= (elemname | choice | seq) (? | * | +)?
- Kooperation

  XML: Inhalt, Element

  DTD: Struktur, Elementdef

  stylesheet: Darstellung

- UML → XML

  Darstellung von UML Diag. als XML Dokument

  i) Klassen
     a) <class name = ”Vorlesung”>
        Inhalt
        </class>
     b) <Vorlesung> Inhalt </Vorlesung>

  ii) Attribute
     a) als Inhalt
        a) <attribut name = ”Titel” type = ”String”> wert </attribut>
        b) <Titel type = ”String”>wert</Titel>
        c) <String name = ”Titel”>wert</String>
     b) als Attribut (Aggregation)
        <Vorlesung Titel = ”Softwaretechnik” Zeit = ”Mon 10:00”/>
     c) global als Referenz (Komposition, als eigenes Element)
        a) <Titel class = ”Vorlesung” type = ”String” value = ”ST” />
        b) <attribut class = ”Vorlesung” type = ”String value = ”ST”/>
        c) <attribut name = ”Titel” idref = ”KV1”/>

3.2 XHTML und CSS

- XHTML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http...">
<html xmlns="http://www.w3.org/1999/xhtml">
        <head>
                <title>Hermann Rezept</title>
                <link rel="stylesheet" type="text/css" href="hermann1html.css">
                <style type="text/css" href="hermann1html.css">
                </style>
                <!–xhtml mit externem Stylefile –>
        </head>
        <body>
                <p>Guten Tag</p>
                <p>ich bin Hermann, euer neues Familienmitglied
                        <strong><span class="rot"> Metalllöffel</span></strong>
                        nur Plastik oder Holz
                </p>
                <div>
                        <span class="underline">Am 5.Tag</span>
                        habe ich Hunger. Bitte füttert mich mit:
                        <span class="inlineliste">
                        <ul style="text-indent:10cm">
                                <li>1 Tasse <span class="zutat">Milch</span></li>
                        </ul>
                        </span>
                </div>
        </body>
</html>
```

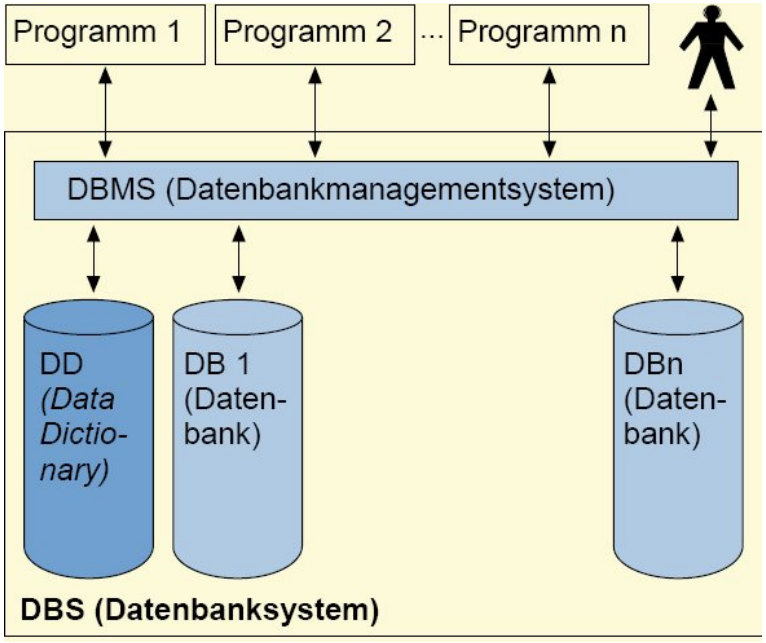
- CSS

```
* {font-size:large}
.rot{color: red}
.underline{text-decoration: underline}
.zutat{color: blue}
.inlineliste{display:inline}
```

4 Datenbanken und SQL

4.1 Datenbanksystem

- Verschiedene Programme in einer Organisation benötigen gemeinsame Daten
- DBMS: integrierte Verwaltung aller Daten in einer Organisation



- DBMS: Software zur Definition, Konstruktion und Manipulation von DB
- Relationales DBS
  relationales Datenmodell
- Objektorientiertes DBS
  homogene objektorientierte Entwicklung
  Objekte der Anwendung lassen sich dirket in der DB speichern
- relationale und XML DBS
  komplexe Daten als XML Dokumente

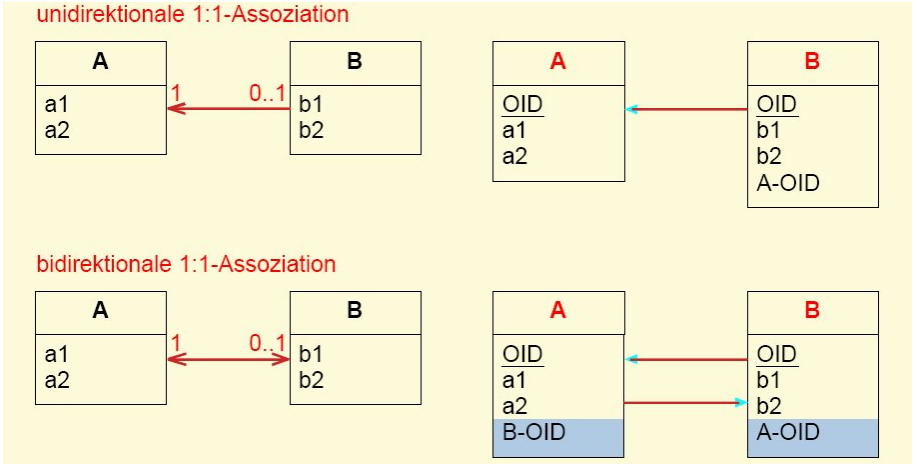
4.2 Relationale Datenbanken

- Relationale DB speichern Daten in Form von Tabellen
- Jede Zeile der Tabelle (Tupel) repräsentiert Objekt
  Tupel: Teilmenge des kart. Produkts
- Primärschlüssel (Identifizierung des Tupels), auch ID
- Schlüsselattribute müssen explizit verwaltet werden (haben immer Wert)
  Objektidentität: gehört explizit zu jedem Objekt
- Fremdschlüssel: muss Primärschlüssel in korrespondierender Tabelle sein
- Attribut: Name und Typ (optional: not null)
- Charakteristika
  - Datenintegration und Datenkonsistenz
  - Datenabstraktion
  - Datensicherheit
  - Datenschutz
  - Persistenz (von dauerhafter Beschaffenheit)

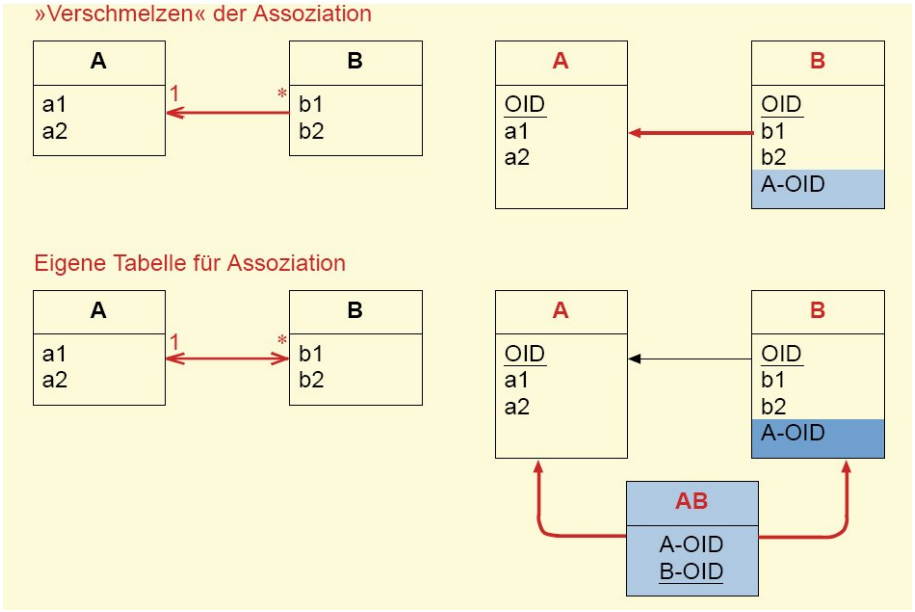
4.3 Datenmodellierung - OOA

- Datenbahängigkeiten: Menge von Attributen und Bedingungen
- Richtlinien
  - klare Semantik
  - Vermeide Redundanz, Update-Anomalien
  - Vermeide Nullwerte
  - ermögliche komplexe Anfagen
- Normalformen
  - Erste Normalform (Datenredundanz)
    keine Wiederholung von Werten und keine internen Datenstrukturen
  - Zweite Normalform
    Alle Nichtschlüssel-Attribute hängen vom gesamten Schlüssel ab
  - Dritte Normalform (Schlüsselredundanz)
    Alle Nichtschlüssel-Attribute hängen direkt von jedem Schlüssel ab
  - Boyce-Codd Normalform
    3NF und keine Abhängigkeit innerhalb Schlüsseln

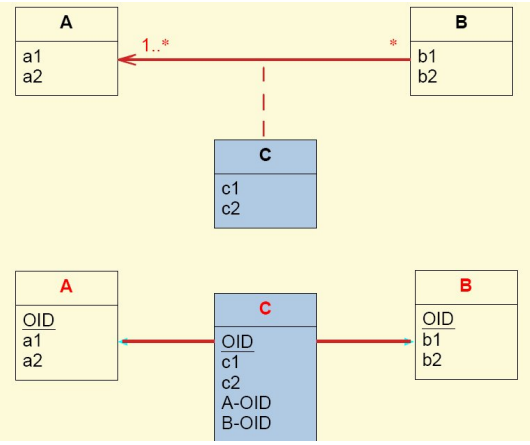
- ”soviel Normalisierung wie nötig”
- Schlüsselabhängigkeit: ein Teil der Schlüssel reicht aus um Tupel zu identifizieren
- OOA ⇒ relational (Abbildung einer Klasse auf eine Tabelle)
  - Erweiterung jeder Tabelle im OID-Attribut (Object Identifier)
  - Klassenattribut: speichere nur einmal für alle Objekte einer Klasse
  - 1:1 Assoziation:



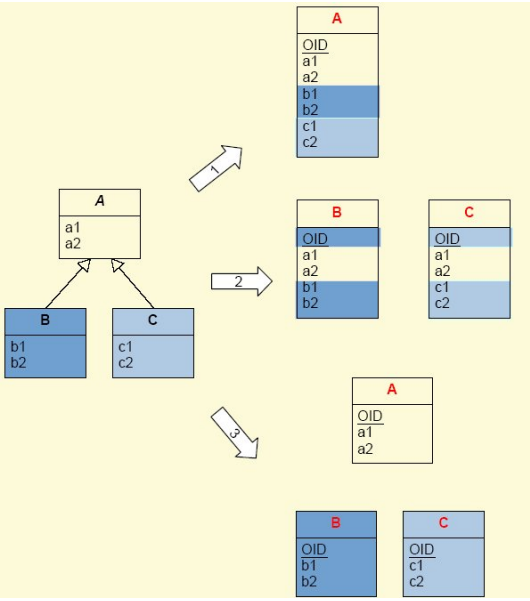
  - 1:m
    Verschmelzen der Assoziation mit Tabelle (weniger Tabellen)
    Abbilden der Assoziation auf separate Taelle



– m:m Assoziation
  immer auf separate Tabellen abbilden
  Primärschlüssel dieser Tabelle aus Schlüsseln der beteiligten



– Abbildung der Vererbungshierarchie



1. Eine Tabelle für gesamte Vererbungshierarchie
   (keine Joins, viele Nullwerte)
2. Eine Tabelle für jede konkrete Klasse
   Bei Modifizierung der Attribute: Aktualisieren aller betroffenen Tabellen
3. Eine Tabelle für jede Klasse (auch abstrakte)
   entspricht am besten objektorientierten Konzept(neue Attribute können einfach ergänzt werden)
   aber: viele Tabellen (viele Joins, langsamer Zugriff)

4.4 SQL

- Tabellen

  create database datenbank;

  use datenbank;

- Data Definition: create table, drop table

  create table Lieferant

  (Nummer number(5) not null,

  Firma char(30) not null);

- Data Manipulation
  - insert into Artikel values(...);
  - update Artikel set Preis = 4.95 where Nummer = 4711;
  - delete from Artikel where Nummer = 4711;

- Datenzugriff
  - Selektion (Ergebnis: Zeilen einer Tabelle)
    select * from Artikel;
    select * from Artikel where Preis $\geq$ 100
  - Projektion (Angabe von gewünschte Attributen)
    distinct vermeidet Duplikate von Datensätzen
    select distinct Bezeichnung, Preis from Artikel;
  - Natürlicher Verbund (Natural Join)
    select Lieferant.Nummer, Firma, Bezeichnung,Preis
    from Lieferant, Artikel where Artikel.L_Nummer = Lieferant.Nummer

- Index

  Verwendung: Steigerung der Performance
  - für Attribute, die häufig in where Klauseln von select-Befehlen
  - für Schlüsselattribute

  create unique index Artikelnummer on Artikel(Nummer);

- Komplexität bei Join:

  O(n^2)

  normaler Index: O(n log n)

  Hashindex: O(n)