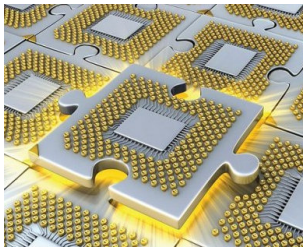
Objektorientierte und Funktionale Programmierung
SS 2014

2 Objektorientierte Analyse mit UML



Zu meiner Person - Alexander Holland

- Studium der Informatik an der Universität Dortmund
- Promotion an der Universität Siegen 2008
- Seit 2005 wiss. Mitarbeiter für Wissensbasierte Systeme und Wissensmanagement an der Universität Siegen
- Seit 2013 Professor für Wirtschaftsinformatik
- Forschung: Computational Intelligence, Graphische Modellbildung, Entscheidungstheorie, Entscheidungsunterstützungs-Systeme, Verteilte Systeme, Soft Computing Anwendungen, Relationship Discovery im Wissensmanagement, IT Management
- E-mail: alex@informatik.uni-siegen.de
- Web: http://www.uni-siegen.de/fb12/ws/mitarbeiter/
- Tel.: 0271/740-2276 Büro: H-A 8413
- Sprechstunde: Mittwoch, 16:00 - 17:00 Uhr

2 Objektorientierte Analyse mit UML ...

Lernziele

- Vertieftes Verständnis objektorientierter Modellierung
- Verständnis der Nutzung und des Aufbaus von UML Klassendiagrammen für die Analyse
- Grobes Wissen um die weiteren Möglichkeiten von UML

Literatur

- **[Ba05], LE 2, 3**
- [Oe05], Kap. 2.1-2.9
- Zur Vertiefung: [Oe05], Kap. 4.3, 4.4, 4.8.2

2 Objektorientierte Analyse mit UML ...

2.1 Einführung in UML

- UML = *Unified Modelling Language*
  - standardisierte (graphische) Sprache zur objektorientierten Modellierung von (Software-)Systemen
  - Entwicklung seit 1994, derzeit aktuelle Version UML 2.0
- UML definiert eine Vielzahl von Diagrammtypen
  - unterschiedliche Sichtweisen des modellierten Systems
    - statische vs. dynamische Aspekte
    - unterschiedlicher Abstraktionsgrad
- UML unterstützt sowohl OOA als auch OOD
- (In dieser Vorlesung: nur sehr kleiner Teil der UML behandelt)

2.1 Einführung in UML ...

- **Wie kam es zur Objektorientierung und zur UML?**
- **In welchem historischen Kontext stehen die UML und objektorientierte Entwicklungsmethoden?**

  - **Smalltalk**: erste umfassende und kommerziell verfügbare objektorientierte Entwicklungsumgebung (seit 1976), die – bis auf Interface-Konzept – alle Grundkonzepte der Objektorientierung enthält
  - **OMT** (Object Modeling Technique) von Rumbaugh seit Anfang der 90er
  - **UM** (Unified Method) von Rumbaugh und Booch in 1995
  - **UML** (Unified Modeling Language): Vereinheitlichung der grafischen Darstellung und Semantik der Modellierungselemente (Notation)
  - die **Amigos**: Rumbaugh, Booch und Ivar Jacobson (Use Cases)

2.1 Einführung in UML ...

- **UML 1.1:** Einreichung und Akzeptanz bei der OMG (Object Management Group) als standardisierte Methode in 1997
- **UML 2.0:** Einreichung und Akzeptanz bei der OMG als grundsätzliche Überarbeitung der UML (Spezifikation; Metamodell und Abdeckungsumfang) in 2003
- **UML 2.1:** Einreichung und Akzeptanz bei der OMG als weitere Überarbeitung und Erweiterung (Zustandsdiagramme) der UML in 2006

2.1 Einführung in UML ...

- **Wesentliche Unterschiede zwischen den strukturierten und den objektorientierten Methoden**
  - **Ganzheitliche Arbeitsgegenstände:** Klassenkonzept mit Einheiten aus Daten und Operationen statt Trennung von Daten und Operationen
  - **Bessere Abstraktionsmöglichkeiten:** Schwerpunkt der Modellierung stärker im Problembereich statt im Lösungsbereich
  - **Methodische Durchgängigkeit:** Ergebnisse einer Aktivität i lassen sich ohne weiteres in die Aktivität i + 1 übernehmen und umgekehrt; in allen Phasen der Softwareentwicklung wird mit denselben Konzepten gearbeitet (Klassen, Objekte, Beziehungen)
  - **Evolutionäre Entwicklung:** Übertragung des Prinzips der Evolution (schrittweise Weiterentwicklung) auf die Softwareentwicklung

2.1 Einführung in UML ...

- **Methodische Durchgängigkeit:**
  - **Anforderungserhebung, Analyse**

    Entwickler: „Was ist euch wichtig?“
    Anwender: „Der Kunde.“
    Entwickler: „Was ist denn ein Kunde, welche Merkmale sind für euch relevant?“
    Anwender: „Der Kunde hat einen Namen, eine Anschrift und eine Bonität, die wir überprüfen.“

  - **Design:**

Kunde
name anschrift bonitaet
bonitaetPruefen()

Kunde als Klasse in UML

  - **Implementierung**

```
class Kunde {
  String      name;
  Anschrift   anschrift;
  Bonitaet    bonitaet;
  public void bonitaetPruefen() {
  …
  } }
```

Kunde als Klasse in Java

2.1 Einführung in UML ...

➢ Vorteile und Nachteile objektorientierter Softwareentwicklung

- **Vorteile**
  - einfacher Zugang zum Anwendungsgebiet
  - bessere Kommunikationsbasis
  - kein Paradigmenwechsel in der Entwicklung
  - bessere Durchschaubarkeit durch Übereinstimmung der Strukturen
  - Möglichkeit von Strukturwiederverwendung
- **Nachteile**
  - die Bedeutung der Vererbung in der objektorientierten Analyse wird häufig überschätzt
  - Gefahr der Überspezifikation
  - Methoden erfordern meist Werkzeugunterstützung, aber die zugehörigen Werkzeuge sind nicht immer ausgereift

2.1 Einführung in UML ...

➢ Abgrenzung Analyse und Design

- **Was ist Analyse?**

  Mit (objekt-orientierter) **Analyse** werden alle Aktivitäten im Rahmen des Softwareentwicklungsprozesses bezeichnet, die der Ermittlung, Klärung und Beschreibung der Anforderungen an das System dienen (d.h. die Klärung, **was** das System leisten soll).

- **Wichtige Ergebnisse der Analyse:**

  Anwendungsfälle

  Fachklassenmodell

  Fachliches Glossar

2.1 Einführung in UML ...

➢ Abgrenzung Analyse und Design

➢ **Was ist Analyse <u>nicht</u>?**

Analyse beinhaltet keine Lösungsansätze

Analyse-Ergebnisse geben keine Antwort auf „Wie“-Fragen

Analyse ist in der Regel unabhängig von der Implementierungstechnik (Programmiersprache, Datenbank, usw.)

2.1 Einführung in UML ...

➢ Abgrenzung Analyse und Design

➢ **Was ist Design?**

Mit (objekt-orientiertem) **<u>Design</u>** werden alle Aktivitäten im Rahmen des Softwareentwicklungsprozesses bezeichnet, mit denen ein Modell logisch und physisch strukturiert wird. Sie dienen zur Beschreibung, **<u>wie</u>** das System die in der Analyse beschriebenen Anforderungen erfüllt.

➢ **Wichtige Ergebnisse des Designs:**

Design-Klassenmodell

Verhaltensmodelle

Implementierungsmodelle

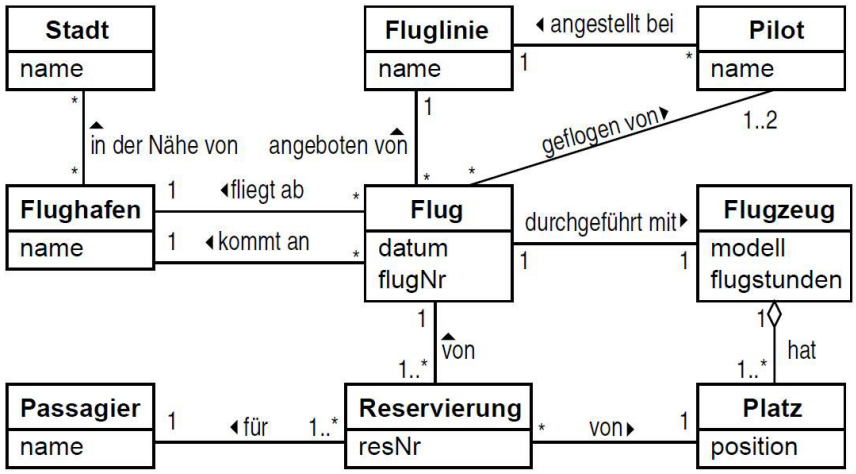
2.1 Einführung in UML ...

Einige Diagrammtypen der UML

- Für statisches Modell:
  - **Klassen- und Objektdiagramme**
    - Objekte, Klassen und ihre Beziehungen
  - Paket- bzw. Komponentendiagramme
    - beschreiben Modulstruktur des Software-Systems
- Für dynamisches Modell:
  - Anwendungsfalldiagramme
    - grobe Aufgaben des Systems und beteiligte Personen
  - Aktivitäts- und Zustandsdiagramme
    - Spezifikation von Abläufen (global bzw. in einem Objekt)
  - Kommunikations-, Timing- und **Sequenzdiagramme**
    - beschreiben die Kommunikation zwischen Objekten

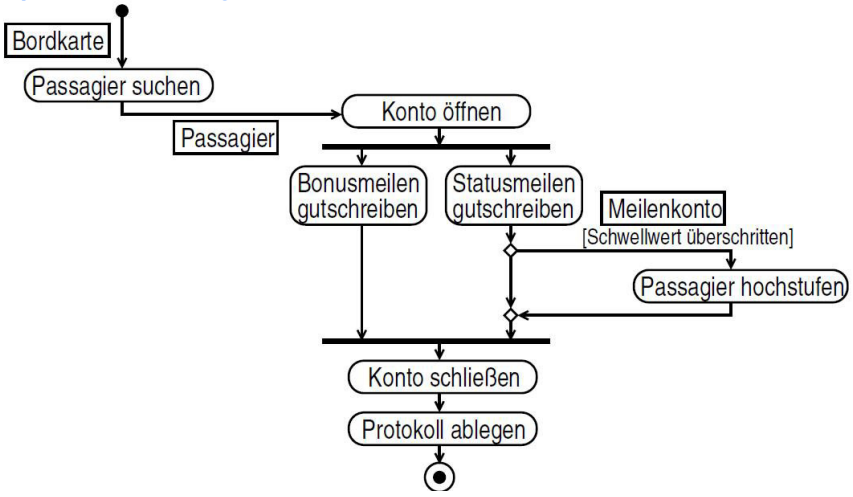
2.1 Einführung in UML ...

Beispiel: Klassendiagramm



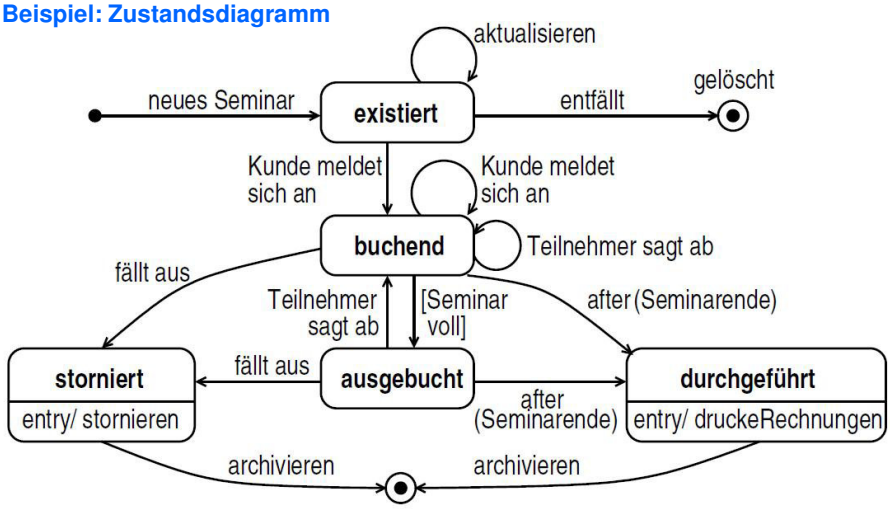
2.1 Einführung in UML ...

Beispiel: Aktivitätsdiagramm



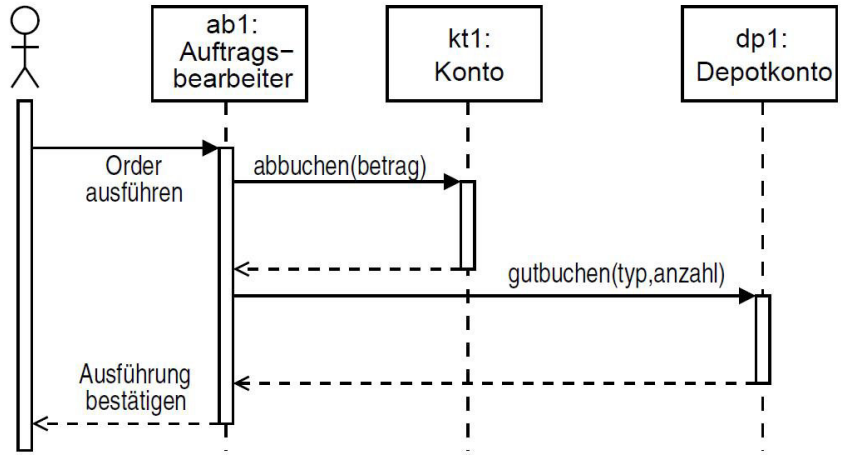
2.1 Einführung in UML ...

Beispiel: Zustandsdiagramm



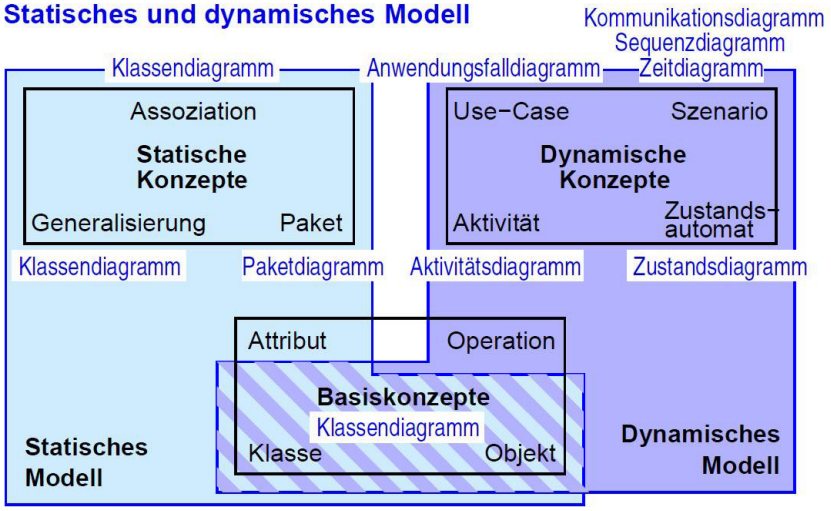
2.1 Einführung in UML ...

Beispiel: Sequenzdiagramm



2.1 Einführung in UML ...

Statisches und dynamisches Modell



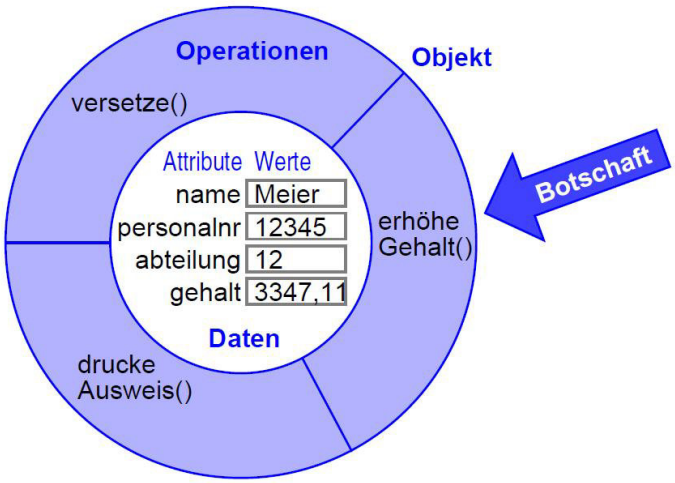
2 Objektorientierte Analyse mit UML ...

2.2 Objekte

- Ein Objekt wird beschrieben durch Zustand und Verhalten
- Der Zustand umfaßt
  - die **Attribute**, bzw. deren aktuelle **Attributwerte**
    - Attribute sind dem Objekt inhärent und unveränderlich,
    - nur die Attribut**werte** können sich verändern
  - die aktuellen **Beziehungen** zu anderen Objekten
- Das Verhalten wird beschrieben durch eine Menge von
- **Operationen (Methoden)**
  - die Änderung oder Abfrage des Zustands ist nur über Operationen möglich (Geheimnisprinzip!)
  - eine Operation wird aktiviert, indem dem Objekt eine **Botschaft** gesendet wird

2.2 Objekte ...

Objekt und Geheimnisprinzip



2.1 Einführung in UML ...

➢ Geheimnisprinzip

- Ein wesentliches Konzept von objektorientierter Programmierung ist, dass der innere Aufbau eines Objekts für andere Objekte zum großen Teil nicht zugänglich ist. Viele Teile eines Objekts können geheim gehalten werden, befinden sich sozusagen in einer geschützten Kapsel. Man nennt das Geheimnisprinzip auch **Kapselung**.
- In Java werden die geheimen, nach außen nicht direkt zugänglichen Teile mit **private** gekennzeichnet, in UML verwendet man ein Minuszeichen **-** .
- Wenn alle Teile eines Objekts geheim wären, könnte man dieses Objekt von außen nicht ansprechen, was ja keinen Sinn macht. Die öffentlichen Teile eines Objekt werden in Java mit **public** gekennzeichnet, in UML verwendet man das Pluszeichen **+**.

2.1 Einführung in UML ...

➢ Geheimnisprinzip

Vorteile:

1. Der Zustand eines Objektes kann nicht inkonsistent werden, ohne dass dieses Objekt ausdrücklich die Erlaubnis dazu erteilt.
2. Seiteneffekte vermindern sich, weil eine Änderung innerhalb der Kapsel keine Auswirkung auf die Schnittstellennutzer hat.

Nachteile:

1. Der Implementierungsaufwand ist größer.
2. Die Performance sinkt, weil mehr Operationsaufrufe und Prüfungen benötigt werden.

2.2 Objekte ...

Allgemeines Schema:

<Objektname> : <Klassenname>
<Attributname 1> = <Wert 1> <Attrubutname 2> = <Wert 2> ... <Attributname n> = <Wert n>

Beispiel:

meinLiebstesBuch : Buch
autor = "J. R. R. Tolkien“ titel = "The Lord of the Rings“ isbn = "0–261–10230–3"

Konvention:
Objekt– und Attributnamen klein geschrieben
Klassennamen groß geschrieben

Anmerkung: die Operationen werden hier nicht angegeben, da sie für alle Objekte einer Klasse identisch sind

2.2 Objekte ...

Darstellungsvarianten

- Objekt ohne Klasse
  - Klasse geht aus Zusammenhang hervor

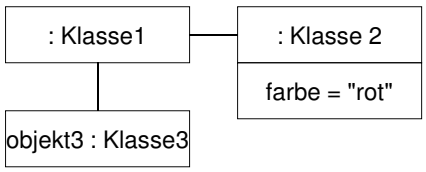
objekt

- Anonymes Objekt

: Klasse

Objektdiagramm mit Objektbeziehungen

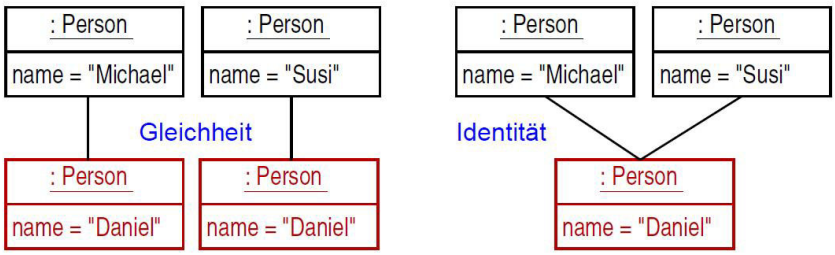
- Beziehungen werden durch Verbindungslinien zwischen Objekten dargestellt



2.2 Objekte ...

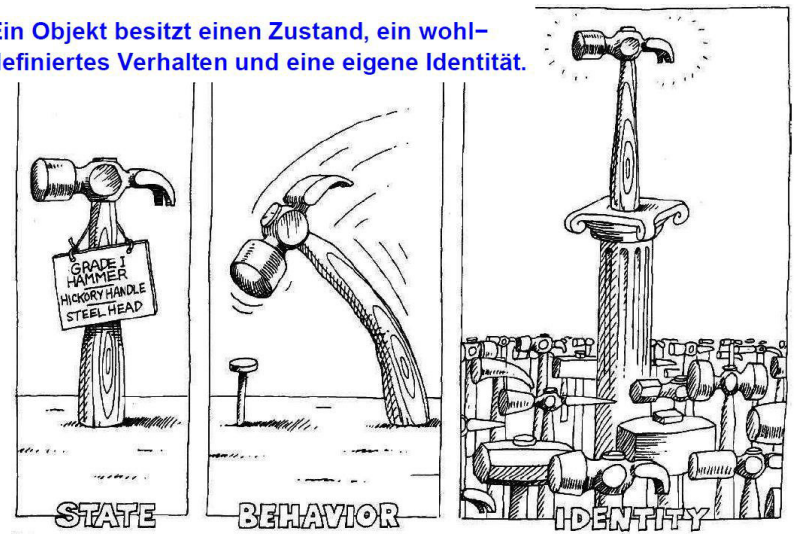
Identität und Gleichheit von Objekten

- Ein Objekt besitzt einen Zustand, ein wohldefiniertes Verhalten und eine Identität, die es von allen anderen Objekten unterscheidet
- Zwei Objekte sind gleich, wenn sie dieselben Attributwerte



2.2 Objekte ...

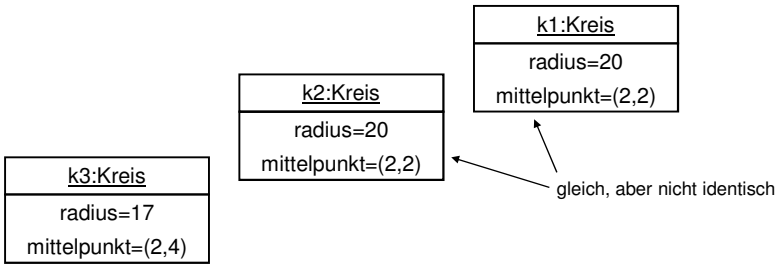
Ein Objekt besitzt einen Zustand, ein wohl-definiertes Verhalten und eine eigene Identität.



2.2 Objekte ...

- **Objektidentität:**

  Jedes Objekt ist per Definition, unabhängig von seinen konkreten Attributwerten, von allen anderen Objekten eindeutig zu unterscheiden.



  - Zur Laufzeit wahren Speicheradressen die Identität eines Objektes
  - In ODBMS werden oft künstlich erzeugte Identitätsnummern (sog. OID‘s verwendet

2 Objektorientierte Analyse mit UML ...

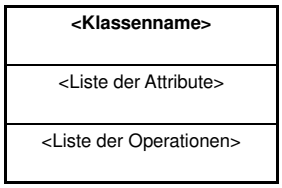
2.3 Klassen

- Eine Klasse definiert für eine Kollektion gleichartiger Objekte
  - deren Struktur, d.h. die Attribute (nicht die Werte!)
  - das Verhalten, d.h. die Operationen
  - die möglichen Beziehungen (**Assoziationen**) zu anderen Objekten
    - einschließlich der Generalisierungs-Beziehung
- Eine Klasse besitzt einen Mechanismus, um neue Objekte zu erzeugen
  - die Klasse ist der ”Bauplan“ für diese Objekte

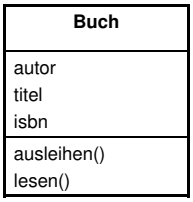
2.3 Klassen ...

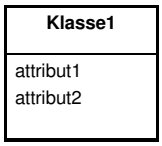
Darstellung von Klassen in UML

Allgemeines Schema:

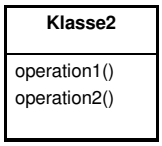


Beispiele:





Klasse ohne Operationen



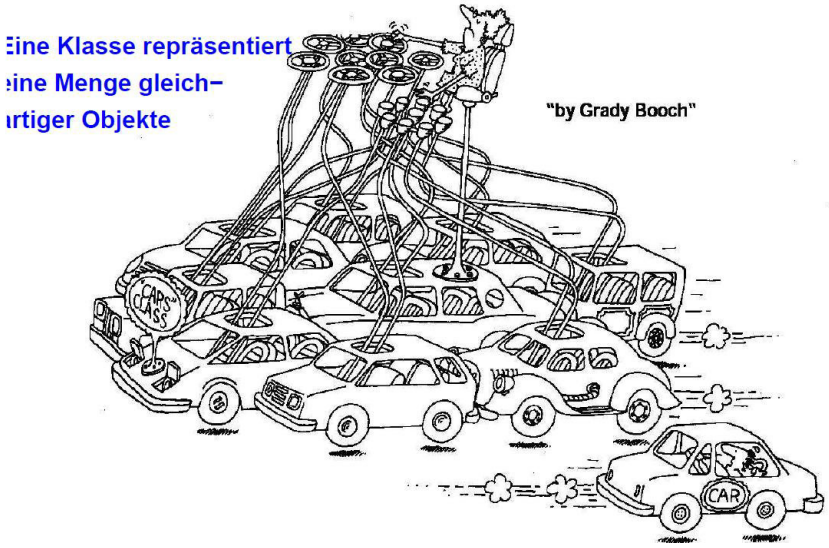
Klasse ohne Attribute

2.3 Klassen ...

Objekte und Klassen

- Jedes Objekt weiß, zu welcher Klasse es gehört
  - somit findet das Objekt die passende Operation, wenn es von einem anderen Objekt eine Botschaft erhält
- Umgekehrt kennt eine Klasse die zu ihr gehörenden Objekte **nicht**
- Bei der OO-**Analyse** nehmen wir aber zur Vereinfachung an, daß eine Klasse alle von ihr erzeugten Objekte kennt
  - d.h. jede Klasse führt über Objekterzeugung und -löschung Buch (**Objektverwaltung**)
    - ohne daß dies explizit modelliert wird
  - die Objektverwaltung muß bei Bedarf(!) in der Entwurfs- und Implementierungsphase realisiert werden

2.3 Klassen ...



2 Objektorientierte Analyse mit UML ...

2.4 Attribute

- Attribut: Datenelement, das in jedem Objekt der Klasse enthalten ist
- Zu einem Attribut kann ein Datentyp angegeben werden,
  - z.B.: Integer (ganze Zahl), String (Zeichenkette), ...
- bei OOA: Datentyp wird nicht angegeben, wenn er aus dem Kontext hervorgeht
- Ein Anfangswert legt fest, welchen Wert das Attribut bei der Erzeugung eines neuen Objekts zunächst erhält
- der Wert kann später beliebig geändert werden

Klasse
attribut 1
attribut2: Typ
attribut3= „Anfragswert“

2.4 Attribute ...

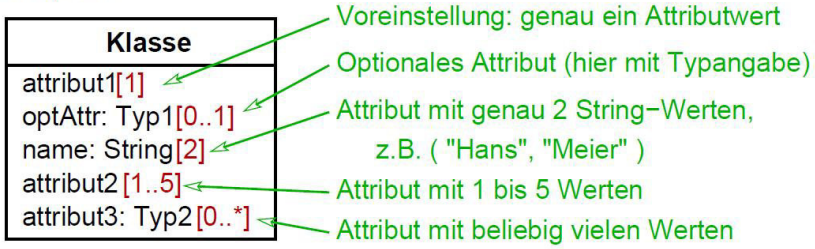
Multiplizitäten

- Attribute können mit einer Multiplizität versehen werden
- Die Multiplizität gibt an, aus wievielen Werte das Attribut bestehen
- kann: [Untergrenze .. Obergrenze]

mindestens so viele

höchstens so viele (* = beliebig viele)

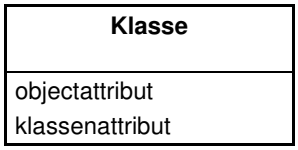
- Beispiele:



2.4 Attribute ...

Klassenattribute

- Klassenattribute sind Attribute, für die nur ein einziger Attributwert für alle Objekte der Klasse existiert:
  - sie werden daher der Klasse zugeordnet, nicht den Objekten
  - sie existieren auch, wenn es (noch) kein Objekt der Klasse gibt
  - sie stellen oft auch Eigenschaften der Klasse selbst dar
- Klassenattribute werden durch Unterstreichen gekennzeichnet:



2 Objektorientierte Analyse mit UML ...

2.5 Operationen

- Operation: ausführbare Tätigkeit, die von einem Objekt über eine Botschaft angefordert werden kann
  - alle Objekte einer Klasse haben dieselben Operationen
  - Operationen können direkt auf die Attributwerte eines jeden Objekts der Klasse zugreifen
  - (Synonyme: Services, Methoden, Funktionen, Prozeduren)

- Drei Arten von Operationen:
  - (Objekt-)Operationen
    - werden immer auf ein einzelnes (bereits existierendes) Objekt angewandt
  - Konstruktoroperationen
    - erzeugen ein neues Objekt u. initialisieren seine Attribute
  - Klassenoperationen

2.5 Operationen ...

Klassenoperationen

- Klassenoperationen sind der Klasse zugeordnet und werden nicht auf ein einzelnes Objekt angewendet

- Sie werden durch Unterstreichen kenntlich gemacht

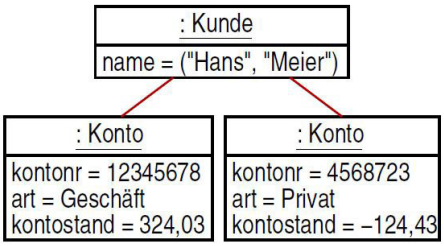
- In der OOA werden Klassenoperationen in zwei Fällen benutzt:
  - Manipulation von Klassenattributen ohne Beteiligung eines Objekts
    - z.B. erhöheStundenlohn()
  - Operation bezieht sich auf alle oder mehrere Objekte der Klasse
    - nutzt Objektverwaltung aus
    - z.B. druckeListe()

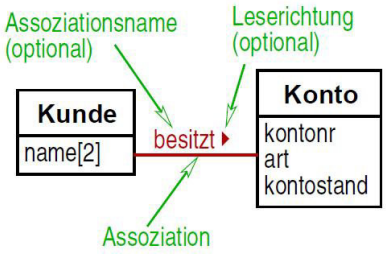
Aushilfe
name adresse studenzahl studenlohn
erhöheStudenlohn() Druckliste()

2 Objektorientierte Analyse mit UML ...

2.6 Assoziationen

- Zwischen Objekten können Objektbeziehungen bestehen
- Assoziationen beschreiben gleichartige Objektbeziehungen zwischen Klassen
  - Objektbeziehung ist Instanz einer Assoziation

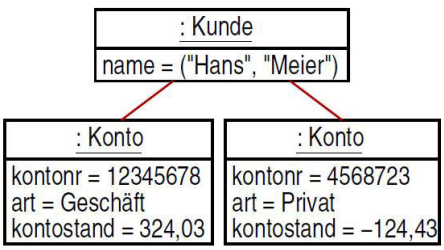


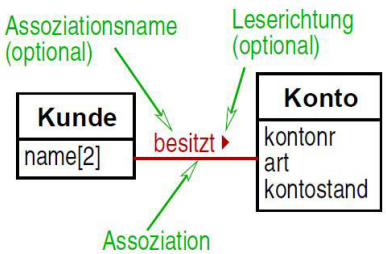


2 Objektorientierte Analyse mit UML ...

2.6 Assoziationen

- Zwischen Objekten können Objektbeziehungen bestehen
- Assoziationen beschreiben gleichartige Objektbeziehungen zwischen Klassen
  - Objektbeziehung ist Instanz einer Assoziation

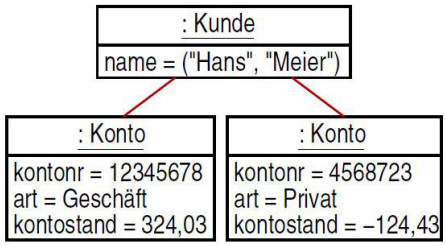


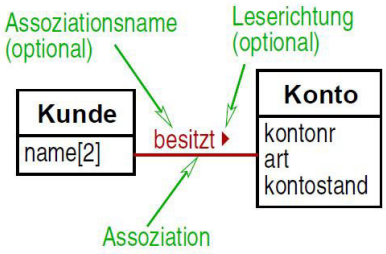


2 Objektorientierte Analyse mit UML ...

2.6 Assoziationen

- **Assoziationen sind Strukturbeziehungen**
  Die Assoziation ist eine **statische** Beziehung, d.h. sie drückt aus, welche Objekte zu jederzeit miteinander verbunden sind, damit Abläufe funktionieren können (sie kann nicht darstellen, wie eine Datenänderung entsteht.

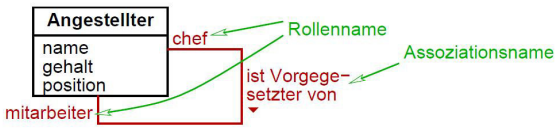




2.6 Assoziationen ...

Reflexive Assoziationen, Rollennamen

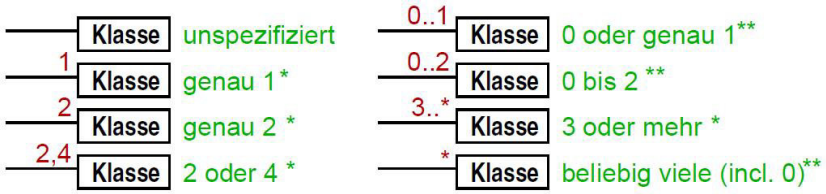
- An einer Objektbeziehung können auch zwei Objekte derselben Klasse beteiligt sein
  - führt zu **reflexiver Assoziation** zwischen Klassen
- Neben Assoziationsnamen auch Rollennamen möglich
  - Assoziationsname: Bedeutung der Assoziation
  - Rollenname: Bedeutung einer Klasse in der Assoziation
- Beispiel für eine reflexive Assoziation:



2.6 Assoziationen ...

Multiplizität von Assoziationen

- Assoziation sagt zunächst nur, **daß** ein Objekt andere Objekte kennen kann
- Multiplizität legt fest, **wieviele** Objekte ein Objekt kennen kann (oder muß)
- Die Multiplizität wird am Ende der Assoziations-Linie notiert:

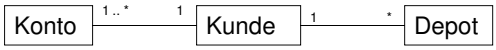


* **Muß–Assoziation:** Objekt muß in Beziehung zu anderen stehen
** **Kann–Assoziation:** Objekt kann, muß aber nicht in Beziehung stehen

2.6 Assoziationen ...

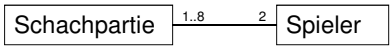
Beispiele zur Multiplizität

- Beispiel: Bank



  - ein Kunde muß mindestens ein Konto besitzen
  - ein Konto gehört zu genau einem Kunden
    - wenn der Kunde gelöscht wird, muß auch das Konto gelöscht werden
  - ein Kunde kann beliebig viele Depots besitzen

- Beispiel: Simultan–Schach

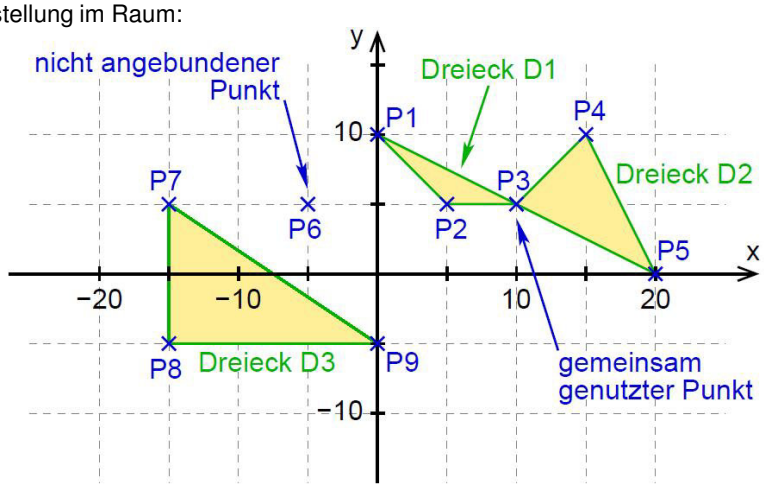


  - jede Schachpartie wird von zwei Spielern gespielt
  - ein Spieler spielt 1 bis 8 Partien gleichzeitig

2.6 Assoziationen ...

Beispiel: Punkte und Dreiecke

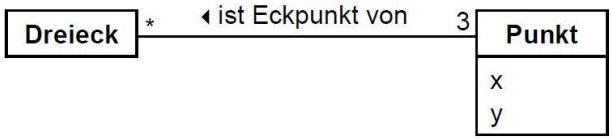
- Darstellung im Raum:



2.6 Assoziationen ...

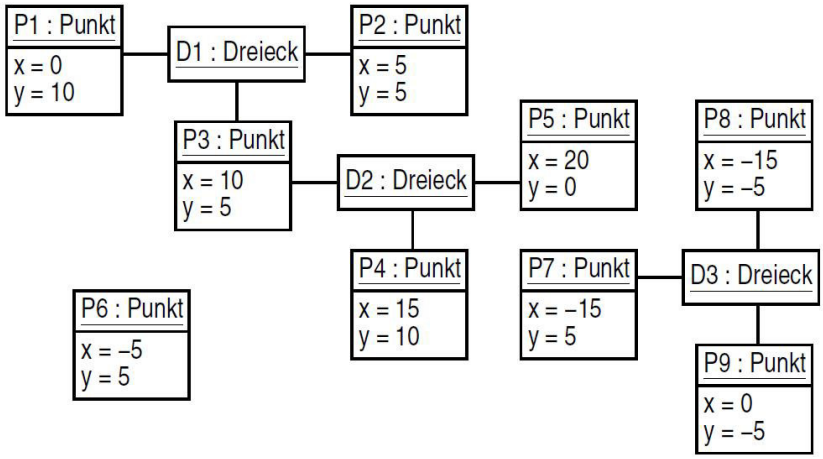
Beispiel: Punkte und Dreiecke ...

- Klassendiagramm
  - jedes Dreieck steht mit 3 Punkten in Verbindung
  - ein Punkt ist Teil von beliebig vielen Dreiecken



2.6 Assoziationen ...

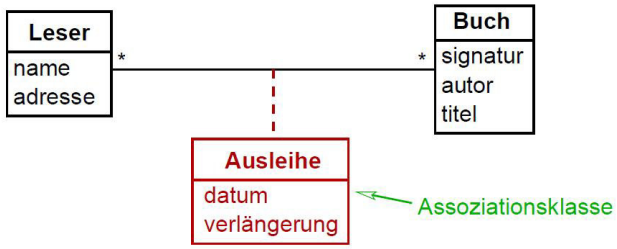
Beispiel: Punkte und Dreiecke ...



2.6 Assoziationen ...

Assoziationsklassen

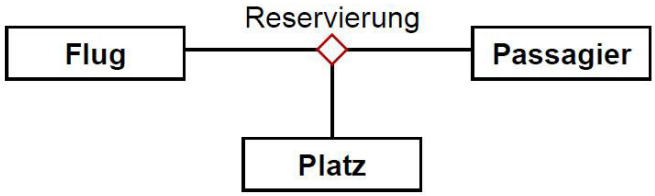
- Manchmal hat auch eine Assoziation Eigenschaften und Verhalten
- Dann: Assoziation wird explizit als Klasse modelliert
- Beispiel:



2.6 Assoziationen ...

Mehrstellige (n-äre) Assoziationen

- Assoziationen sind auch zwischen mehr als zwei Klassen möglich
- Darstellung am Beispiel einer ternären (dreistelligen) Assoziation:



- eine Reservierung ist eine Beziehung zwischen Passagier, Flug und Platz

2.6 Assoziationen ...

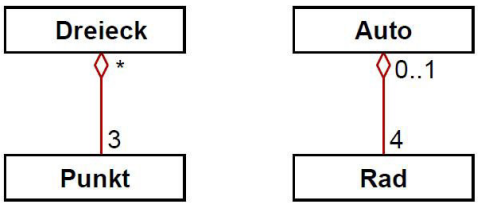
Aggregation und Komposition

- Häufige Abstraktion im täglichen Leben: Teile/Ganzes-Beziehung
  - "besteht aus" bzw."ist Teil von"
  - z.B.:"Ein Auto besteht aus einer Karosserie, 4 Rädern, ..."
- **Aggregation**: Teile existieren selbständig und können (gleichzeitig) zu mehreren Aggregat-Objekten gehören
- **Komposition**: starke Form der Aggregation
  - Teil-Objekt gehört zu genau einem Komposit-Objekt
    - es kann nicht Teil verschiedener Komposit-Objekte sein
    - es kann nicht ohne sein Komposit-Objekt existieren
  - Beim Erzeugen (Löschen) des Komposit-Objekts werden auch seine Teil-Objekte erzeugt (gelöscht)

2.6 Assoziationen ...

Darstellung von Aggregation und Komposition

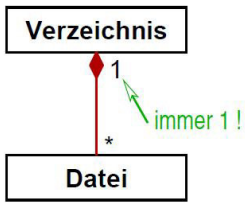
Aggregation



Ein Dreieck besteht immer aus 3 Punkten.
Ein Punkt ist Teil von beliebig vielen (incl. 0) Dreiecken.

Ein Auto besteht (u.a.) aus 4 Rädern.
Ein Rad ist Teil von höchstens einem Auto (es gibt auch Räder ohne Auto).

Komposition



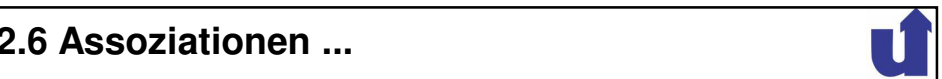
Ein Verzeichnis besteht aus beliebig vielen Dateien
Dateien stehen immer in einem Verzeichnis. Wird dieses gelöscht, so auch alle enthaltenen Dateien.

2.6 Assoziationen ...

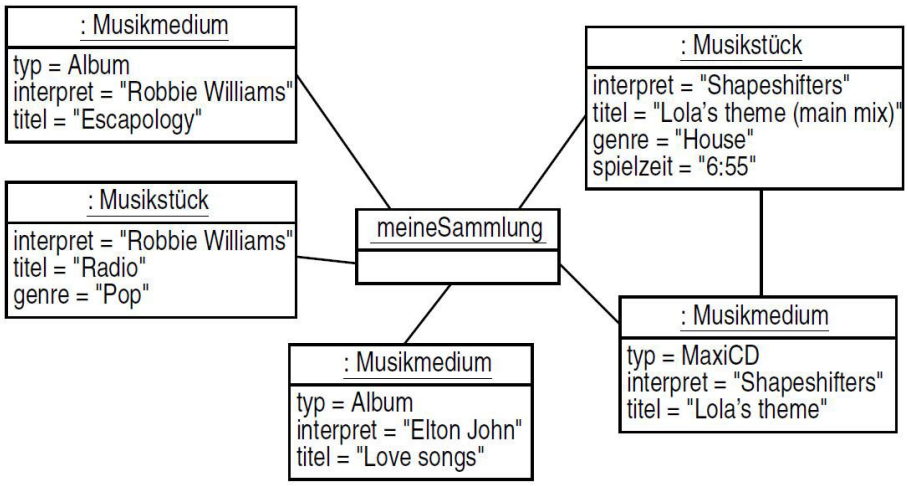
Beispiel: Verwaltung einer Musiksammlung

- Aufgabe: modelliere ein System, das Musikstücke und Musikmedien verwalten kann
- Die Sammlung enthält (u.a.):
  - *Robby Williams: Escapology* (Album)
  - *Elton John: Love songs* (Album)
  - *Shapeshifters: Lola's theme* (Maxi-CD)
  - *Robby Williams: Radio* (Musikstück)
  - *Shapeshifters: Lola's theme (main mix)* (Musikstück)
- Erstelle
  - ein Objektdiagramm
  - und ein Klassendiagramm

2.6 Assoziationen ...



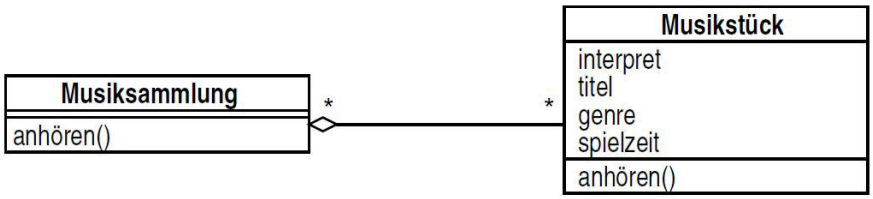
Musiksammlung: Objektdiagramm



2.6 Assoziationen ...

Musiksammlung: Klassendiagramm (erster Schritt)

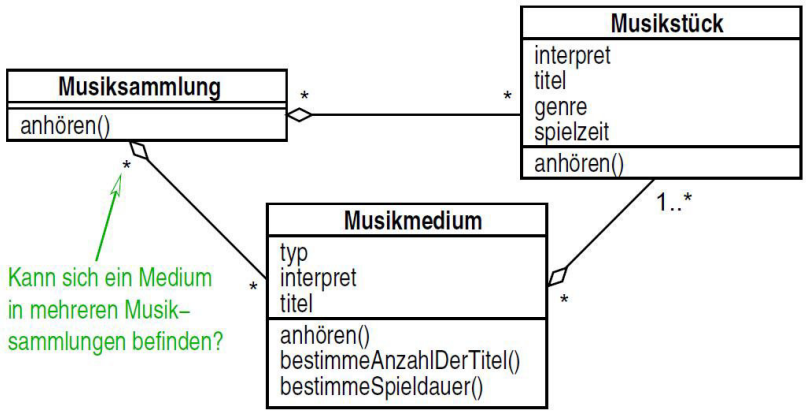
- Wir betrachten zunächst nur die Musikstücke:



2.6 Assoziationen ...

Musiksammlung: Klassendiagramm (zweiter Schritt)

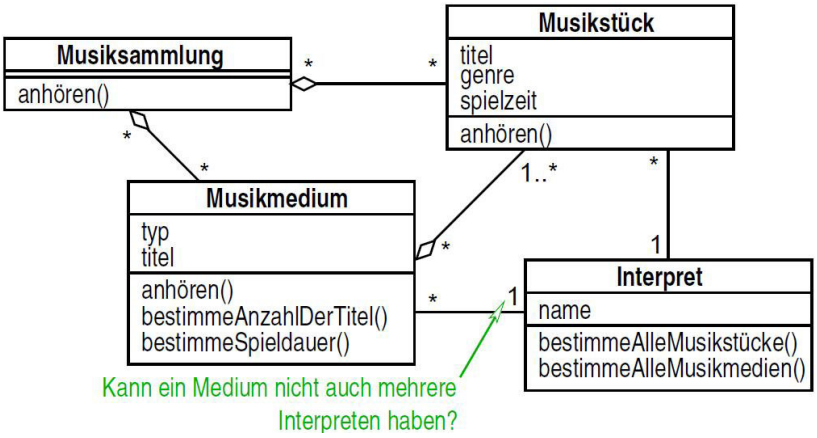
- Einordnung der Musikmedien:



2.6 Assoziationen ...

Musiksammlung: Klassendiagramm (dritter Schritt)

- Interpret kann(!) als eigene Klasse modelliert werden:



2 Objektorientierte Analyse mit UML ...

2.7 Generalisierung

- **Generalisierung**: Beziehung zwischen einer allgemeineren Klasse (**Basisklasse**, **Oberklasse**) und einer spezialisierteren Klasse (**Unterklasse**)
  - die spezialisierte Klasse ist konsistent mit der Basisklasse, enthält aber zusätzliche Attribute, Operationen und / oder Assoziationen
  - ein Objekt der Unterklasse kann überall da verwendet werden, wo ein Objekt der Oberklasse erlaubt ist
- Nicht nur: Zusammenfassung gemeinsamer Eigenschaften und Verhaltensweisen, sondern immer auch: Generalisierung im Wortsinn
  - jedes Objekt der Unterklasse **ist ein** Objekt der Oberklasse
- Generalisierung führt zu einer **Klassenhierarchie**

2.7 Generalisierung ...

Beispiel: Angestellte, Studenten und (stud.) Hilfskräfte

Modellierung als unabhängige Klassen:

Angestellter
personalnr name adresse geburtsdatum gehalt bankverbindung
druckeAdresse() überweiseGehalt()

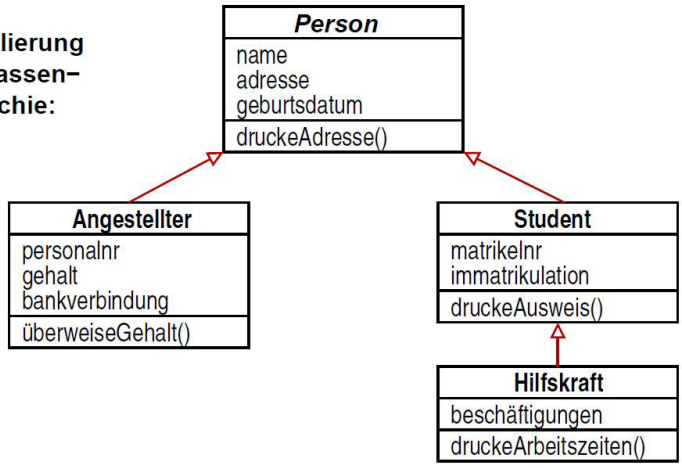
Student
matrikelnr name adresse geburtsdatum immatrikulation
druckeAdresse() druckeAusweis()

Hilfskraft
matrikelnr name adresse geburtsdatum immatrikulation beschäftigungen
druckeAdresse() druckeAusweis() druckeArbeitszeiten()

2.7 Generalisierung ...

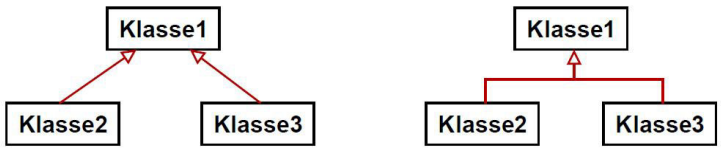
Beispiel: Angestellte, Studenten und (stud.) Hilfskräfte ...

Modellierung als Klassen-hierarchie:

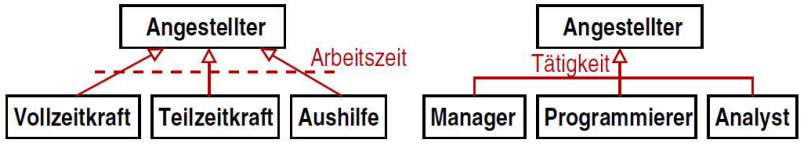


2.7 Generalisierung ...

Beispiel: Angestellte, Studenten und (stud.) Hilfskräfte



- Ein zusätzlicher **Diskriminator** (Generalisierungsmenge) kann das Kriterium angeben, nach dem klassifiziert wird:



2.7 Generalisierung ...

Abstrakte Klassen

- Im vorherigen Beispiel wurden im Modell nur Personen betrachtet, die entweder Angestellter, Student oder Hilfskraft sind
- Die neue Basisklasse ”Person“ wird daher als **abstrakte Klasse** modelliert
  - von einer abstrakten Klasse können keine Instanzen (Objekte) erzeugt werden
- Darstellungen in UML: *Klasse* *Klasse* *{abstract}*

Klassenname in Kursivchift — Für handschriftliche Diagramme

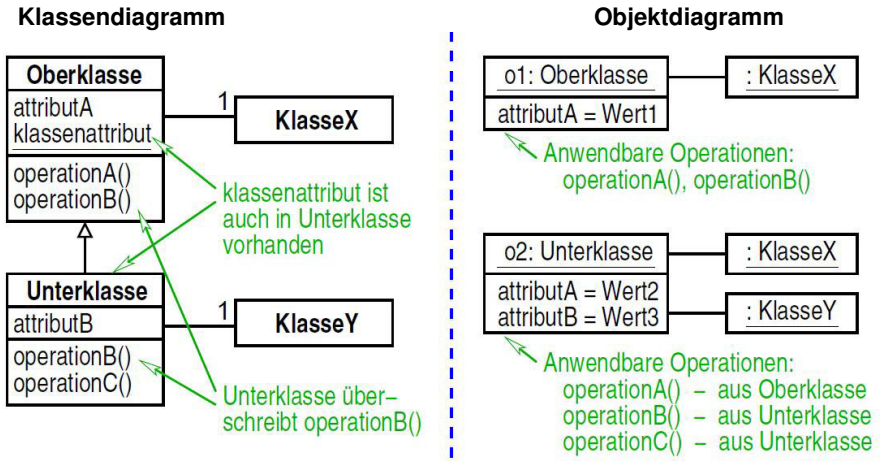
2.7 Generalisierung ...

Vererbung

- Eine Unterklasse übernimmt (**erbt**) von ihren Oberklassen
  - alle Attribute (auch Klassenattribute)
    - ggf. auch den Anfangswert
  - alle Operationen (auch Klassenoperationen)
    - d.h. alle Operationen einer Oberklasse können auch auf ein Objekt der Unterklasse angewendet werden
  - alle Assoziationen
- Die Unterklasse kann zusätzliche Attribute, Operationen und Assoziationen hinzufügen, aber ererbte nicht löschen
- Die Unterklasse kann das Verhalten neu definieren, indem sie Operationen der Oberklasse **überschreibt**
  - d.h. eine Operation gleichen Namens neu definiert

2.7 Generalisierung ...

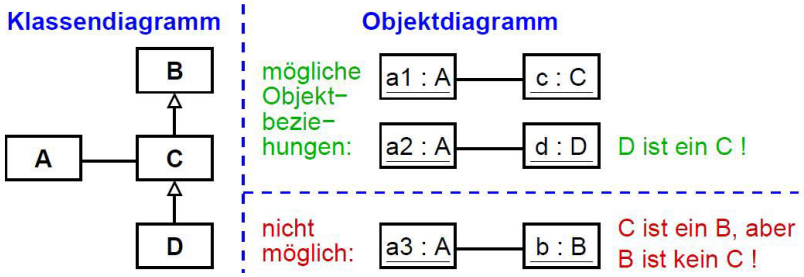
Beispiel zur Vererbung



2.7 Generalisierung ...

Assoziationen und Generalisierung

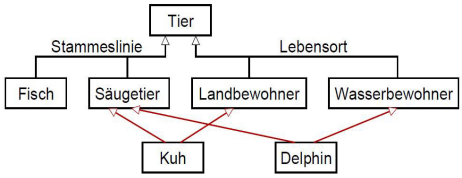
- Assoziationsbeziehungen werden ebenfalls vererbt
  - das bedeutet: wenn es eine Assoziation zwischen A und B gibt, kann ein Objekt von A auch mit einem Objekt einer **Unterklasse** von B in Beziehung stehen
- Beispiel:



2.7 Generalisierung ...

Mehrfachvererbung

- Eine Klasse kann auch von mehreren direkten Basisklassen erben:



- Konzept wird nicht von allen Programmiersprachen unterstützt
  - Probleme, wenn z.B. Oberklassen verschiedene, aber gleichnamige Attribute / Operationen besitzen

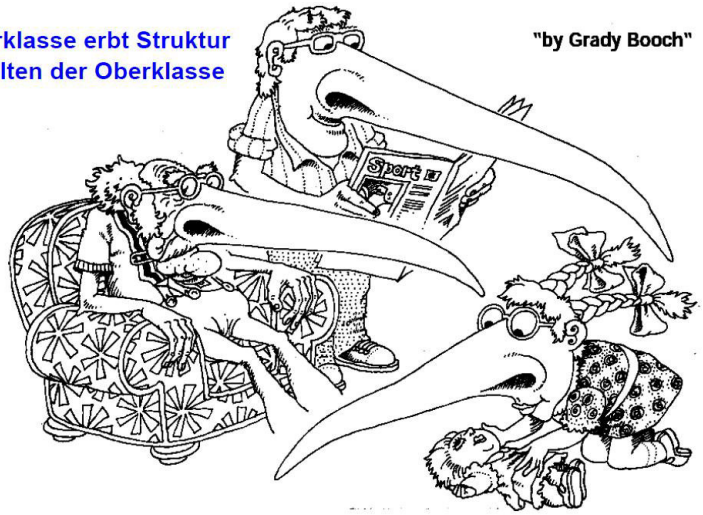
2.7 Generalisierung ...

Diskussion: Vorteile und Probleme

- Bessere Strukturierung des Modell-Universums
- Aufbauend auf vorhandenen Klassen können ähnliche Klassen mit wenig Aufwand erstellt werden
- Einfache Änderbarkeit: Änderung von Attributen / Operationen der Basisklasse wirkt sich automatisch auf die Unterklassen aus
  - dies kann aber auch unerwünscht sein
- Klassen sind schwieriger zu verstehen / zu verwenden
  - auch alle Oberklassen müssen verstanden werden
- Gefahr, überflüssige Klassenhierarchien zu bilden
- Fazit: Generalisierung mit Bedacht verwenden!

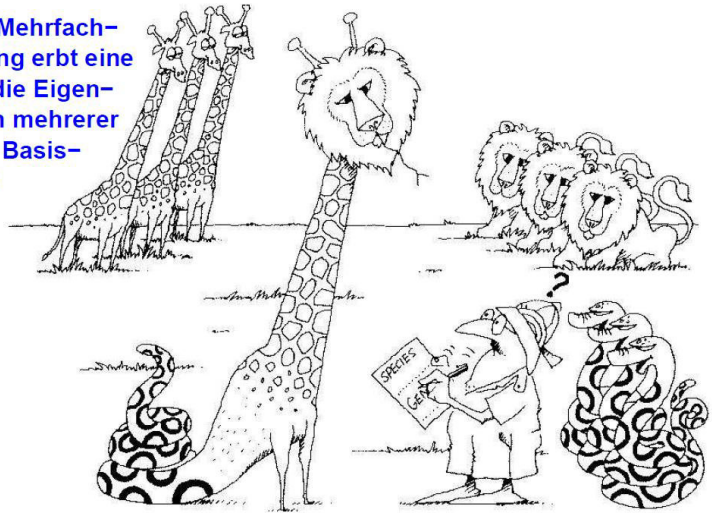
2.7 Generalisierung ...

Eine Unterklasse erbt Struktur und Verhalten der Oberklasse



2.7 Generalisierung ...

Bei der Mehrfach-vererbung erbt eine Klasse die Eigen-schaften mehrerer direkter Basis-klassen



2 Objektorientierte Analyse mit UML ...

2.8 Methoden zur Vorgehensweise

- Wie kommt man von der Problemstellung zum Klassendiagramm?
- Im folgenden zwei einfache Ansätze ([BK03], Kap 12.1):
  - Verb/Substantiv-Methode
    - Bestimmung von Klassen, Attributen und Methoden aus einer textuellen Problembeschreibung
    - für große (reale) Projekte nicht ausreichend, da oft
      - keine vollständige Problembeschreibung vorliegt
      - oder diese sogar widersprüchlich ist
  - CRC-Karten
    - zum Herausarbeiten der Interaktion zwischen Klassen auf der Basis von Anwendungsfällen
- Vertiefung: Softwaretechnik I

2.8 Methoden zur Vorgehensweise ...

2.8.1 Verb/Substantiv-Methode

Beispiel für textuelle Aufgabenstellung: übungsanmeldung

Es soll ein Programm zur Verwaltung von Studenten und Übungen erstellt werden. Eine Übung besteht aus maximal 10 Übungsgruppen, zu denen der Raum und die Uhrzeit gespeichert ist. Jeder Raum hat eine Raumnummer und eine bestimmte Anzahl von Plätzen. Für jeden Studenten wird Name, Matrikelnummer und Email–Adresse erfaßt. Ein Student kann für eine der Gruppen angemeldet sein. In einer Gruppe ist die Zahl der angemeldeten Studenten nur durch die Zahl der Plätze limitiert.

2.8.1 Verb/Substantiv-Methode ...

Heuristiken:

- Jedes Substantiv ist ein Klassen**kandidat**
- Ein Substantiv mit einem einfachen Wert (d.h. kein zusammengesetztes Objekt) ist ein Attribut**kandidat**

Es soll ein Programm zur Verwaltung von Studenten und Übungen erstellt werden. Eine Übung besteht aus maximal 10 Übungsgruppen, zu denen der Raum und die Uhrzeit gespeichert ist. Jeder Raum hat eine Raumnummer und eine bestimmte Anzahl von Plätzen. Für jeden Studenten wird Name, Matrikelnummer und Email–Adresse erfaßt. Ein Student kann für eine der Gruppen angemeldet sein. In einer Gruppe ist die Zahl der angemeldeten sein. In einer Gruppe ist die Zahl der angemeldeten Studenten nur durch die Zahl der Plätze limitiert.

2.8.1 Verb/Substantiv-Methode ...

- **Von Klassen-/Attributkandidaten zu Klassen und Attributen**
  - Streiche Substantive, die zum Beschreibungstext, aber nicht zum Problem gehören (-----)
  - Streiche doppelte Kandidaten (-----)
  - Betrachte immer die Singular-Form (z.B. Student statt Studenten)

Es soll ein ~~Programm~~ zur ~~Verwaltung~~ von Studenten und Übungen erstellt werden. Eine ~~Übung~~ besteht aus maximal 10 Übungsgruppen, zu denen der ~~Raum~~ und die Uhrzeit gespeichert ist. Jeder Raum hat eine Raumnummer und eine bestimmte Anzahl von Plätzen. Für jeden ~~Studenten~~ wird Name, Matrikelnummer und Email–Adresse erfaßt. Ein ~~Student~~ kann für eine der ~~Gruppen~~ angemeldet sein. In einer ~~Gruppe~~ ist die Zahl der angemeldeten sein. In einer Gruppe ist die Zahl der angemeldeten Studenten nur durch die ~~Zahl der Plätze~~ limitiert.

2.8.1 Verb/Substantiv-Methode ...

- Es soll ein Programm zur Verwaltung von Studenten und Übungen erstellt werden.

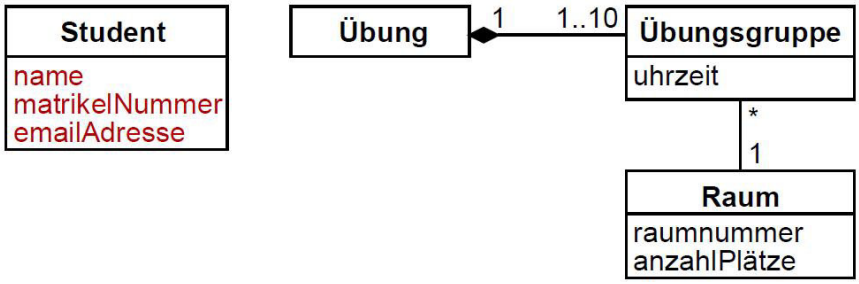
2.8.1 Verb/Substantiv-Methode ...

- Eine Übung besteht aus maximal 10 Übungsgruppen , zu denen der Raum und die Uhrzeit gespeichert ist. Jeder Raum hat eine Raumnummer und eine bestimmte Anzahl von Plätzen.

Anmerkung: Statt Raum als eigene Klasse zu modellieren, hätte man die Raumnummer und die Anzahl der Plätze auch als Attribute der Übungsgruppe wählen können (sinnvoll, wenn das Objekt "Raum" nie anderweitig benötigt wird)

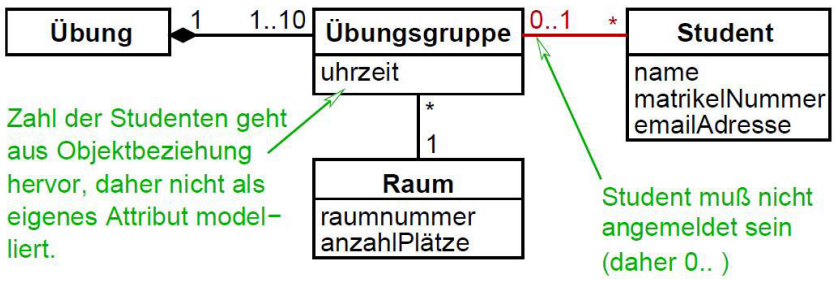
2.8.1 Verb/Substantiv-Methode ...

- Für jeden Studenten wird Name , Matrikelnummer und Email-Adresse erfaßt.



2.8.1 Verb/Substantiv-Methode ...

- Ein Student kann für eine der Gruppen angemeldet sein. In einer Gruppe ist die Zahl der angemeldeten Studenten nur durch die Zahl der Plätze limitiert.



2.8.1 Verb/Substantiv-Methode ...

Weitere Heuristik:

- Verben geben Hinweise auf Methoden

Es soll ein Programm zur Verwaltung von Studenten und Übungen erstellt werden. Eine Übung besteht aus maximal 10 Übungsgruppen, zu denen der Raum und die Uhrzeit gespeichert ist. Jeder Raum hat eine Raumnummer und eine bestimmte Anzahl von Plätzen. Für jeden Studenten wird Name, Matrikelnummer und Email–Adresse erfaßt. Ein Student kann für eine der Gruppen angemeldet sein. In einer Gruppe ist die Zahl der angemeldeten sein. In einer Gruppe ist die Zahl der angemeldeten Studenten nur durch die Zahl der Plätze limitiert.

- Hier nur wenig Hinweise im Text:
  - Erfassung eines Studenten (= Objekterzeugung)
  - Anmeldung bei einer Gruppe

2.8 Methoden zur Vorgehensweise ...

2.8.2 CRC-Karten

- CRC = Class / Responsibilities / Collaborators
- Idee: Anlegen von Karteikarten mit folgendem Aufbau:

Klassenname	Partnerklassen
Zuständigkeiten für welche Aufgaben ist diese Klasse zuständig? (Hinweise auf Attribute und Methoden)	mit welchen anderen Klassen kooperiert diese Klasse? (Hinweise auf Assoziationen)

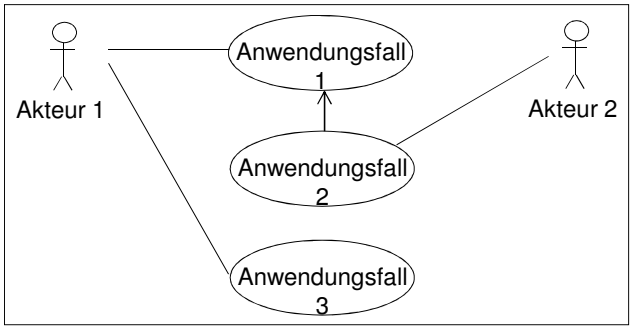
2.8.2 CRC-Karten ...

Methode: Durchspielen von Anwendungsfällen

- Zunächst werden (zusammen mit dem Auftraggeber) typische **Anwendungsfälle** (***Use Cases***) definiert
  - Modellierung z.B. über UML *Use-Case*-Diagramme
- Die Anwendungsfälle werden dann nacheinander durchgespielt
  - auf den CRC- Karten werden dabei neue Zuständigkeiten und Partnerklassen festgehalten
  - im Lauf der Zeit ergibt sich so ein vollständiges Bild
- Wichtig:
  - Untersuchung möglichst aller typischen Anwendungsfälle
  - Festhalten aller Details

2.8.2 CRC-Karten ...

Use Case - Beispiel



Ein Use Case (Anwendungsfalldiagramm) enthält eine Menge von Anwendungsfällen, die durch einzelne Ellipsen dargestellt werden, und eine Menge von Akteuren, die die daran beteiligt sind. Die Anwendungsfälle sind durch Linien mit den beteiligten Akteuren verbunden. Ein Rahmen um die Anwendungsfälle sind symbolisiert die Systemgrenzen.

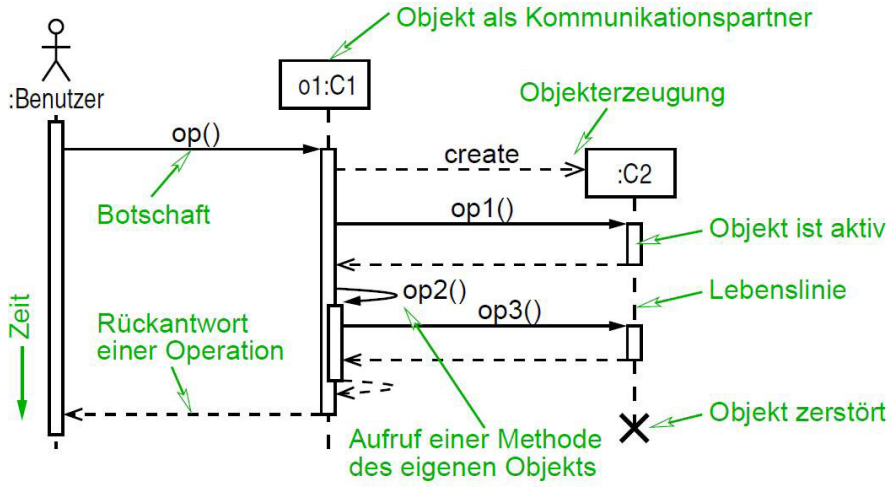
2 Objektorientierte Analyse mit UML ...

2.9 Dynamisches Modell: Sequenzdiagramm

- UML Sequenzdiagramme stellen den zeitlichen Verlauf von Interaktionen zwischen Objekten dar:
  - Botschaften
  - Objekterzeugung und -löschung
- Zusätzlich kann dargestellt werden, wann Objekte aktiv sind, d.h. Operationen bearbeiten
- Ein Sequenzdiagramm dient zur Darstellung genau eines Szenarios (d.h. Beispiel-Ablaufs)
  - nicht zur Modellierung aller möglichen Abläufe

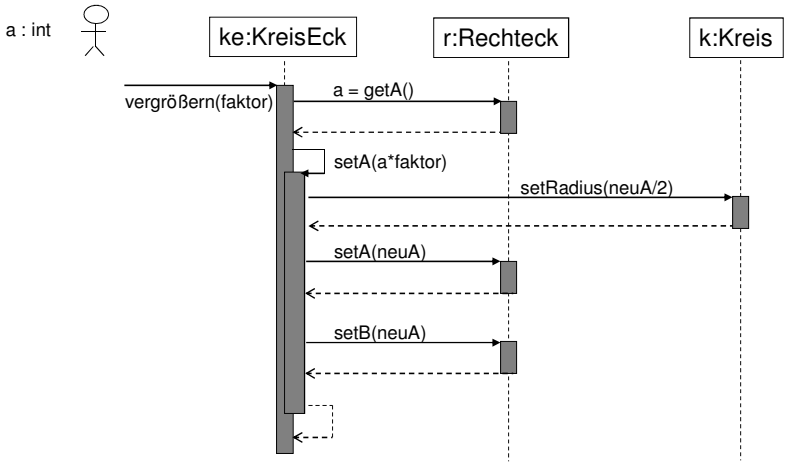
2.9 Dynamisches Modell: Sequenzdiagramm ...

Notationselemente in Sequenzdiagrammen



2.9 Dynamisches Modell: Sequenzdiagramm ...

Beispiel Sequenzdiagramm



2 Objektorientierte Analyse mit UML ...

Beispiel Sequenzdiagramm

- Das Objekt *KreisEck* sendet an das Rechteck die Nachricht *getA()*. Das Rechteck antwortet mit dem Wert der aktuellen Kantenlänge. Die Antwort wird von *KreisEck* temporär in der lokalen Variablen *a* festgehalten.
- Anschließend sendet das *KreisEck* die Nachricht *setA(a*faktor)* an sich selbst.
- Innerhalb der Operation *setA(a)* sendet das Objekt *KreisEck* als erstes an den Kreis die Nachricht *setRadius(neuA / 2)*.
- Anschließend wird an das Rechteck die Nachricht *setA(a)* gesendet.
- Ebenso die Nachricht *setB(a)*.

2 Objektorientierte Analyse mit UML ...

2.10 Zusammenfassung

- UML: graphische Sprache zur objektorientierten Modellierung
  - statisches Modell: u.a. Klassendiagramme
  - dynamisches Modell: u.a. Sequenzdiagramme
- Objektorientierte Konzepte in der Analyse
  - Objekte: Attribute, Beziehungen, Operationen, Identität
  - Klassen: Abstraktion gleichartiger Objekte
    - Attribute, Multiplizitäten, Klassenattribute
    - Operationen, Klassenoperationen
    - Assoziationen, Multiplizitäten, Assoziationsklassen, mehrstellige Assoziationen, Aggregation und Komposition
    - Generalisierung, abstrakte Klassen, Vererbung, Mehrfachvererbung

2.10 Zusammenfassung ...

- Vorgehensweisen zur Erstellung von objektorientierten Modellen
  - Verb/Substantiv-Methode
  - CRC-Karten
- UML Sequenzdiagramme
  - zeitlicher Ablauf der Interaktion zwischen Objekten