



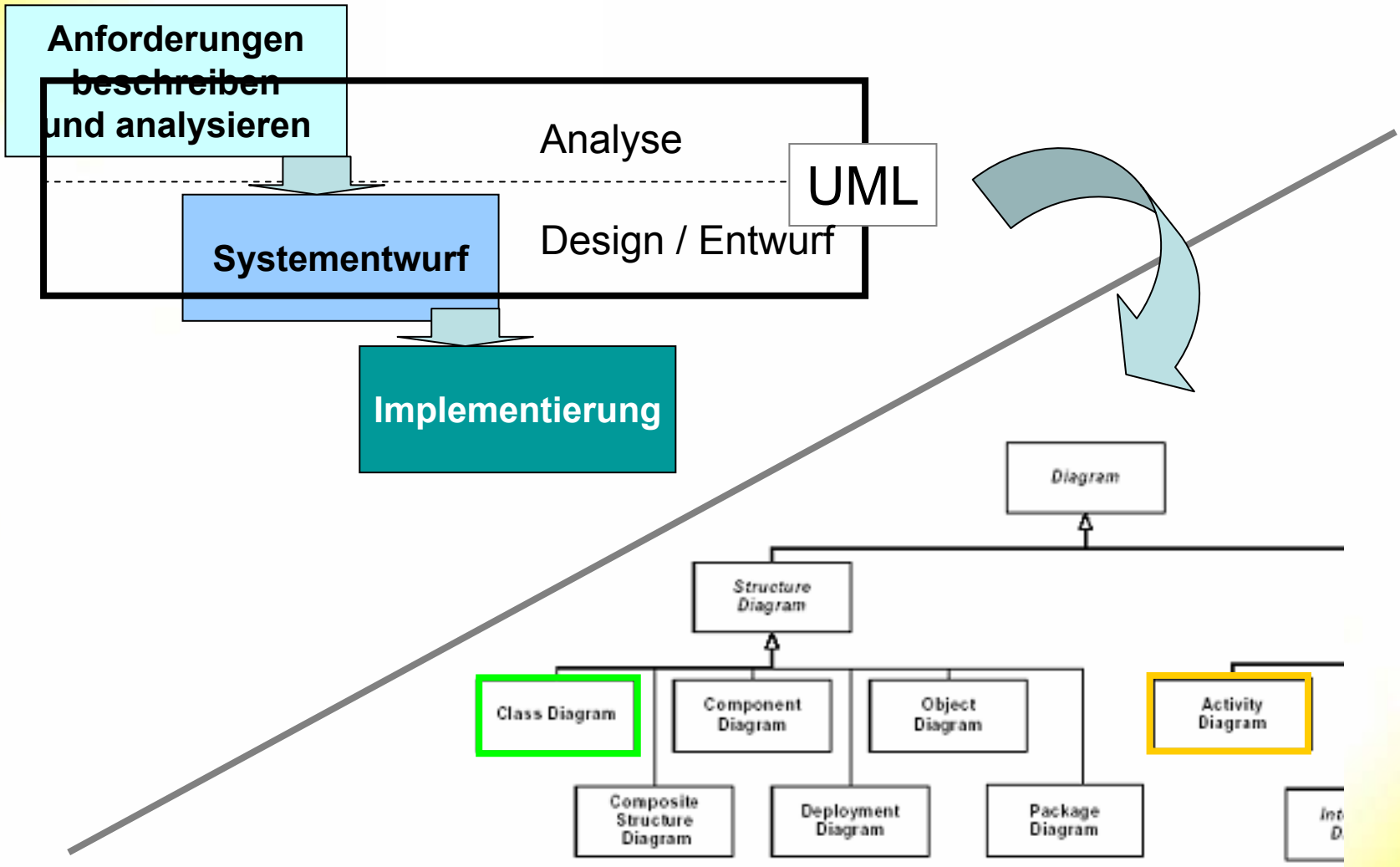
Informatik I

Prof. Dr. Christian Pape

Kapitel 5 Objekt-orientierter Entwurf von Software



Inhalt





Objekt-Orientierung / Definition

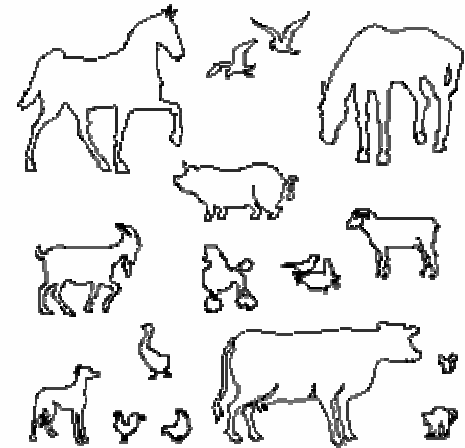
- (Java) Programme bestehen aus *Klassen*
- Klasse beschreibt *Menge von Objekten*
- Objekte sind *Abstraktionen* realer Objekte (physische oder ideelle)
 - Physisch (Materie):
Die Person „Johann W. Goethe“
 - Ideell (Information):
Die Zahl „7“
Temperatur



Objekt-Orientierung / Physische Objekte



Menge aller Personen



Menge aller Tiere



Menge aller Bundesländer (das Gebiet)



Objekt-Orientierung / Ideelle Objekte



Menge aller Bundesländer
(rechtstaatliche Einheit)

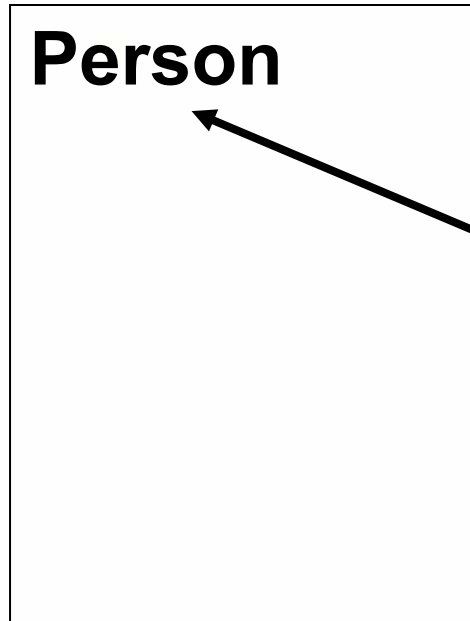
Menger aller deutschen Substantive (Wörter)

Ops	Osterei
Lampe	Apfel
Maus	Indianer
Nilpferd	Igel



Objekt-Orientierung / Klasse

UML



Notation

← Rechteck

← Namen der Klasse fett

Konventionen Klassen

Verwende Substantiv als Name für Klassen

Verwende Einzahl für Klassename

Schreibe ersten Buchstaben gross

Schreibe ersten Buchstaben eines Teilworts gross

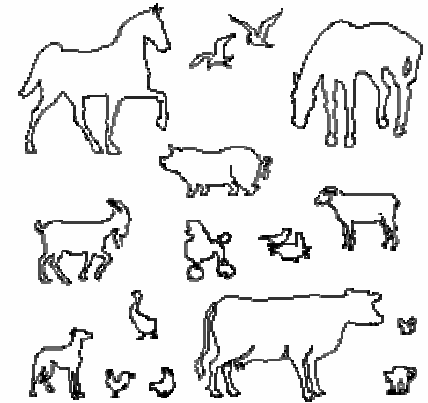


Objekt-Orientierung / Beispiele



Person

Tier



Bundesland

Wort



Opa

Osterei

Lampe

Apfel

Maus

Indianer

Nilpferd

Igel

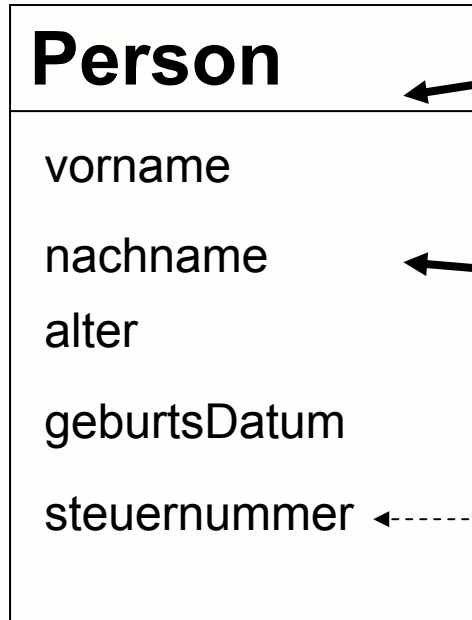


Objekt-Orientierung / Definition (Forts.)

- Klassen definiert *Eigenschaften* von Objekten
 - Eigenschaften werden durch *Attribute* angegeben (Felder, *fields*)
- Auswahl/Identifikation der Eigenschaften ist Teil der Abstraktion
 - Nur für die Problemlösung relevanten Teile werden modelliert

Objekt-Orientierung / Attribute

UML



Notation

← Attribute durch Balken abgetrennt

← Attribute untereinander

← steuernummer
Auswahl hängt von Anforderungen und Problem ab

Konventionen Attribute

Verwende Substantiv als Name für Attribute

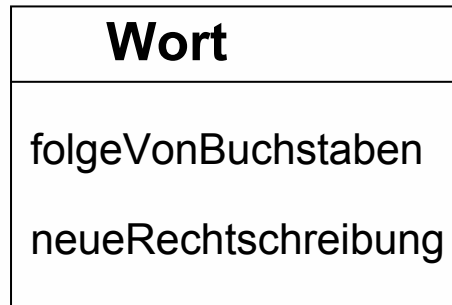
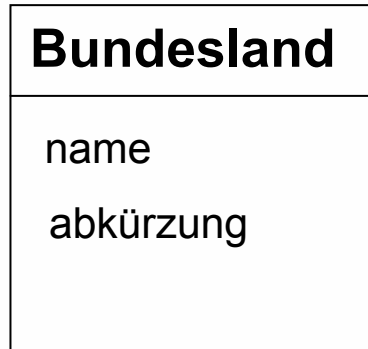
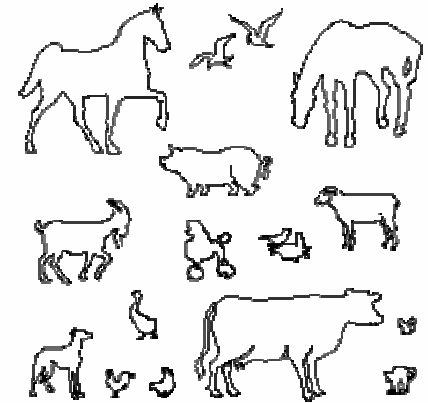
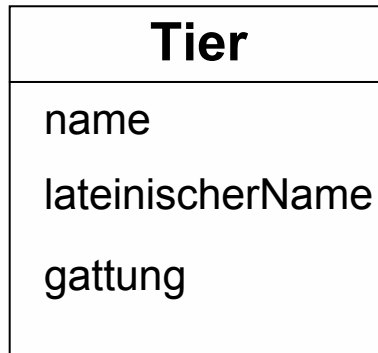
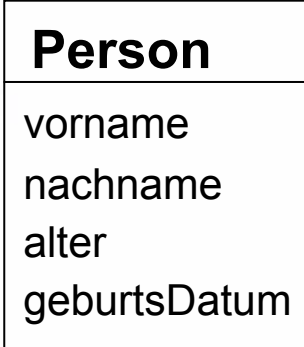
Verwende ggf vorangestelltes Adjektiv

Schreibe ersten Buchstaben klein

Schreibe ersten Buchstaben eines Teilworts gross



Objekt-Orientierung / Beispiele



Ops	Osterei
Lampe	Apfel
Maus	Indianer
Nilferd	Igel

(UML) Klassendiagramm



Objekt-Orientierung / Definition (Forts.)

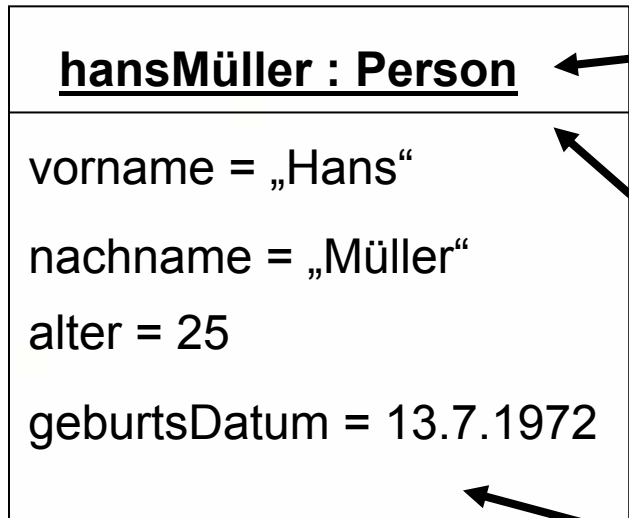
- Ein Objekt ist eine beispielhafte Ausprägung einer Klasse
 - „Hans Müller, 25 Jahre, geb. 13.7.1972“
- Objekt ist *Instanz* einer Klasse
 - Fehlübersetzung von engl. *instance*
 - Eigentlich: Beispiel, Fall
- UML Objekt Diagramme
 - beschreiben Objekte zu einem bestimmten Zeitpunkt
 - „Schnappschuss“

Person
vorname
nachname
alter
geburtsDatum



Objekt-Orientierung / Objekt-Diagramm

UML



Notation

Objektname, optional
Klassenname mit „:“
getrennt

fett, unterstrichen

Attribute und deren
Werte durch
Balken abgetrennt

Attribut = Wert des
Attributes

Aufzählung Attribute
und Werte kann
unvollständig sein

Konventionen Objekte

Schreibe Objektamen
klein

Schreibe ersten
Buchstaben eines
Teilworts gross

Gib immer Klasse des
Objekts mit an



Objekt-Orientierung / Beispiel Objektdiagramm



hansMüller : Person

vorname = „Hans“
nachname = „Müller“
alter = 25
geburtsDatum = 13.7.1972

meier : Person

vorname = „Frida“
nachname = „meier“



badenWürttemberg : Bundesland

name = „Baden-Württemberg“
abkürzung = „BaWü“

rheinlandPfalz

name = „Rheinland-Pfalz“



Objekt-Orientierung / Definition (Forts.)

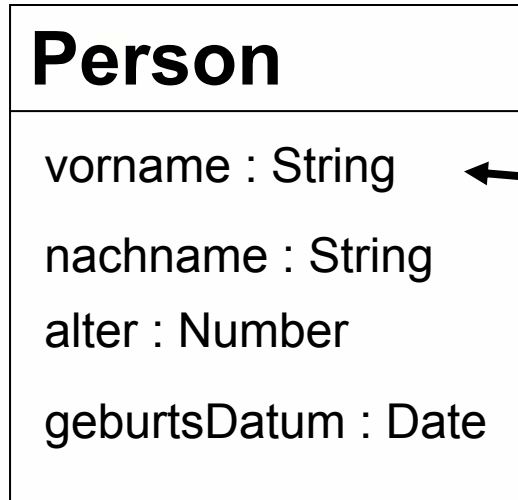
- Typen definieren Wertebereich von Attributen
- Klassen sind Typen
- Beispiele für Typen
 - Natürliche Zahlen von 0..255 (Byte), Zeichenketten beliebiger Länge
 - Person, Tier (falls als Klasse definiert)
- UML kennt „elementaren“ Typen
 - Typen in UML sind immer Klassen
 - Oder Typen einer gewählten Programmiersprache
- Java (Details später)
 - **String** für Zeichenketten wie „Frida Meier“
 - **Character** für ein einzelnes Zeichen
 - **Date** für ein Datum wie „13.7.1972“
 - **Number** für eine Zahl wie -123123
 - **Boolean** für einen Wahrheitswert „true“ oder „false“



Objekt-Orientierung / UML Klassen

UML

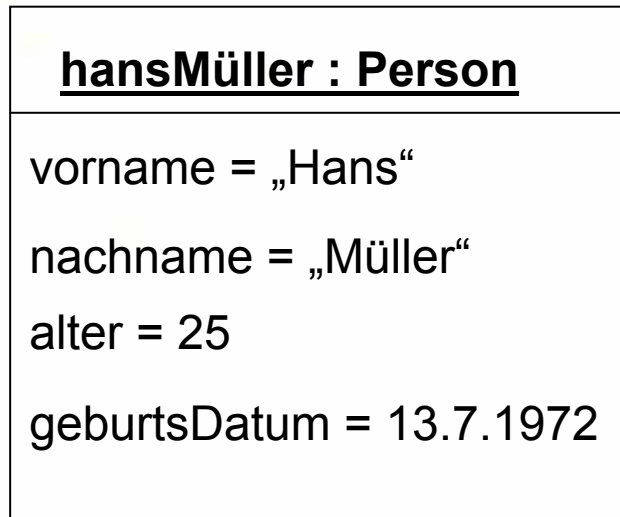
Notation



Optionaler
Typ hinter
Attribut durch
„:“ separiert

Objekt-Orientierung / UML

Objektediagramm



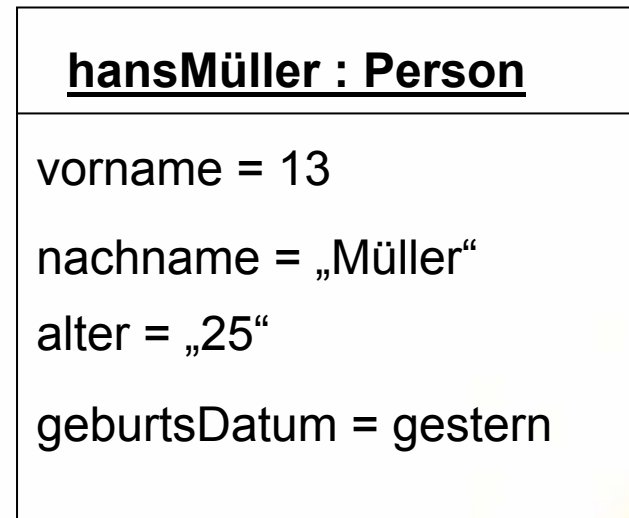
Gültiges Objekt

(Werte entsprechen Typen der Attribute)



Kein gültiges Objekt

(Werte entsprechen nicht immer den Typen der Attribute)



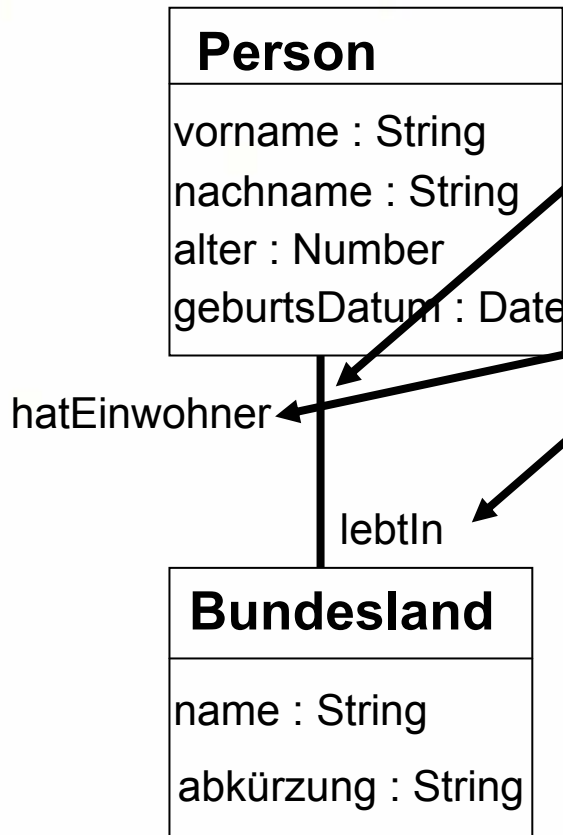


Objekt-Orientierung / Beziehungen

- Reale und ideelle Objekte besitzen *Beziehungen* zueinander
 - Person *lebt in* einem Bundesland
 - Wort *besteht aus* einer Folge von Buchstaben
- Beziehungen sind auch Attribute
 - lebtIn : Bundesland
- Gesonderte Notation in UML
 - Assoziation

Objekt-Orientierung / Assoziationen

UML



Notation

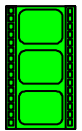
Beziehung zwischen
Zwei Klassen durch
Linie

Opt. Rollennamen
am jeweiligen Ende

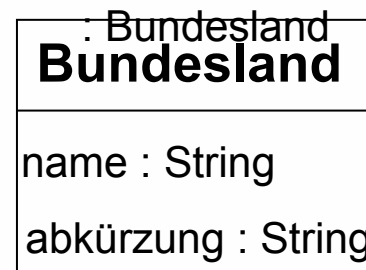
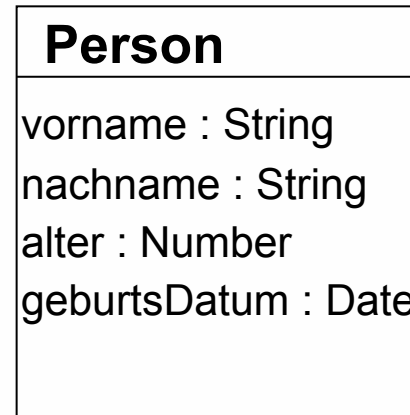
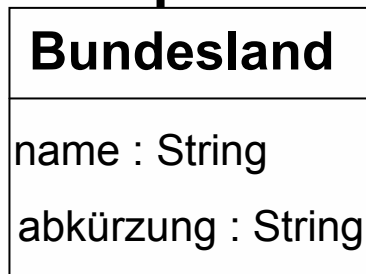
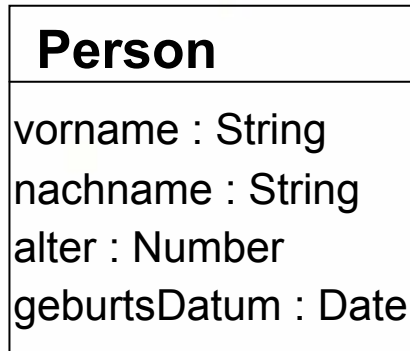
Konventionen Assoziation

Schreibweise
wie Attribute

Verwende ggf
vorangestellte Verben,
Adverbien



Objekt-Orientierung / Assoziationen



Assoziation
kann
als Attribut
modelliert
werden

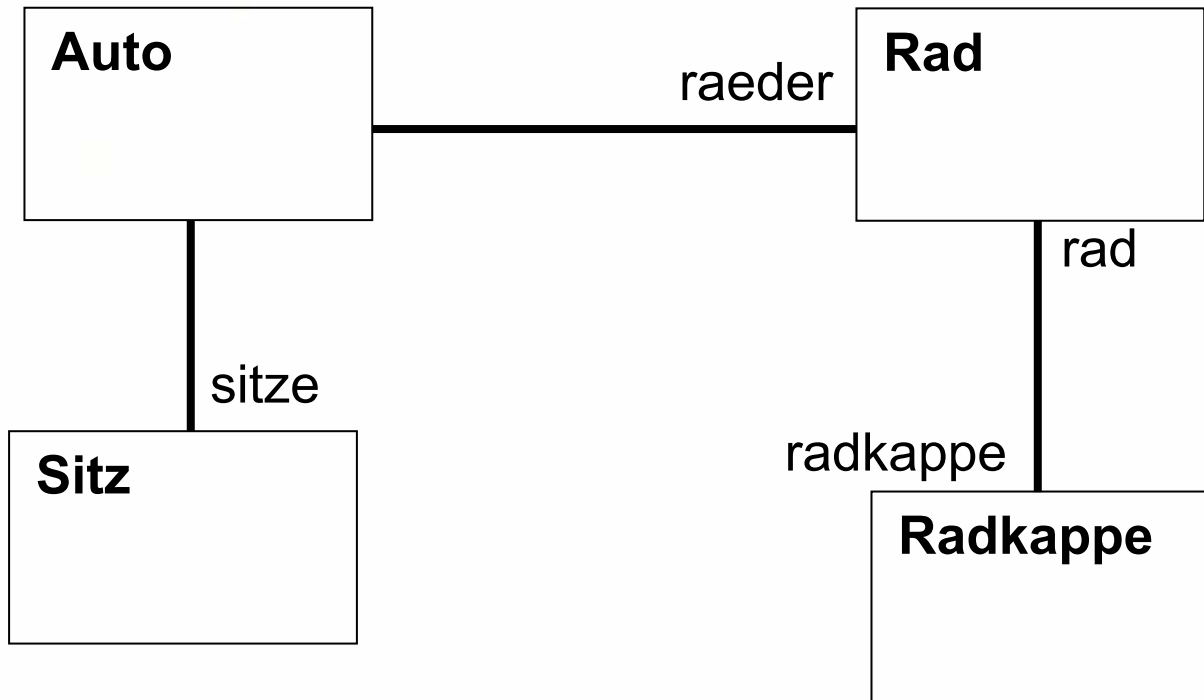


Objekt-Orientierung / Assoziationen

- Assoziationen meist eine Beziehung „hat“
 - Person hat Adresse
 - Baum hat Blätter
 - Auto hat Räder
- Rollenname bei „hat“ Beziehung
 - Name der Klasse als Rollenname, ggf. Mehrzahl
 - „adresse“, „blaetter“, „raeder“



Bespiel / Modellierung Auto



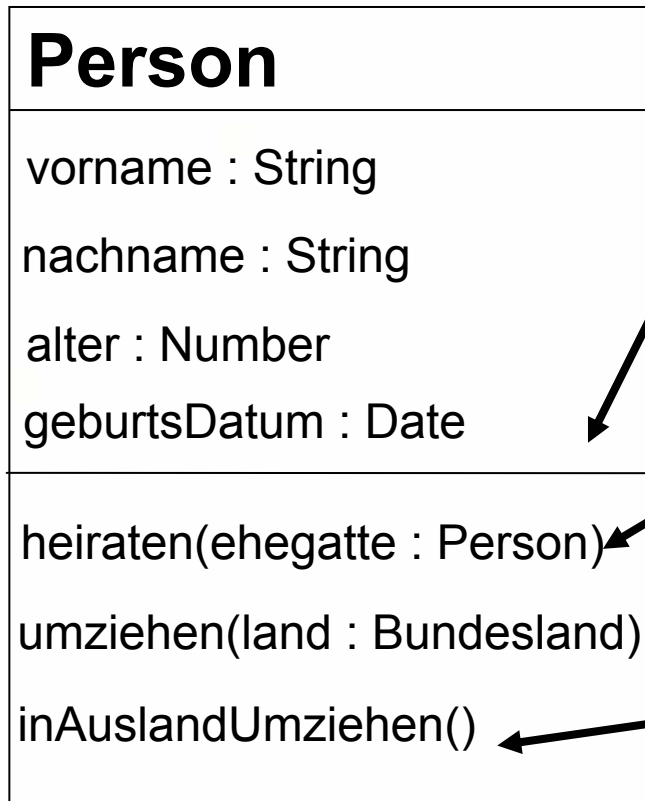


Objekt-Orientierung / Methoden

- Bisher
 - Objekte nur statisch (Datenobjekte)
- *Methoden* (auch Operation, Prozedur, Funktion)
 - Methoden Definieren *Verhalten* von Objekten
 - Sie können Eigenschaftswerte von Objekten ändern
- Beispiele
 - Andere Person heiraten
 - Namensänderung
 - In ein anderes Bundesland umziehen
 - assoziiertes Objekt ändert sich
- Methoden können Parameter besitzen
- Beispiel
 - Den (zukünftigen) *Ehegatten* heiraten
 - *Bundesland* in das umgezogen wird

Objekt-Orientierung / UML Methoden

UML



Notation

Methoden durch
Balken abgetrennt

Optionale Parameter
innerhalb Klammer,
Name und Typ
durch „:“ separiert

Methodenname
gefolgt von ()

Konventionen Methoden

Schreibeweise
wie Attribute

Verwende Verben
für Methodennamen

Schreibe Parameter
klein



Objekt-Orientierung / Methoden

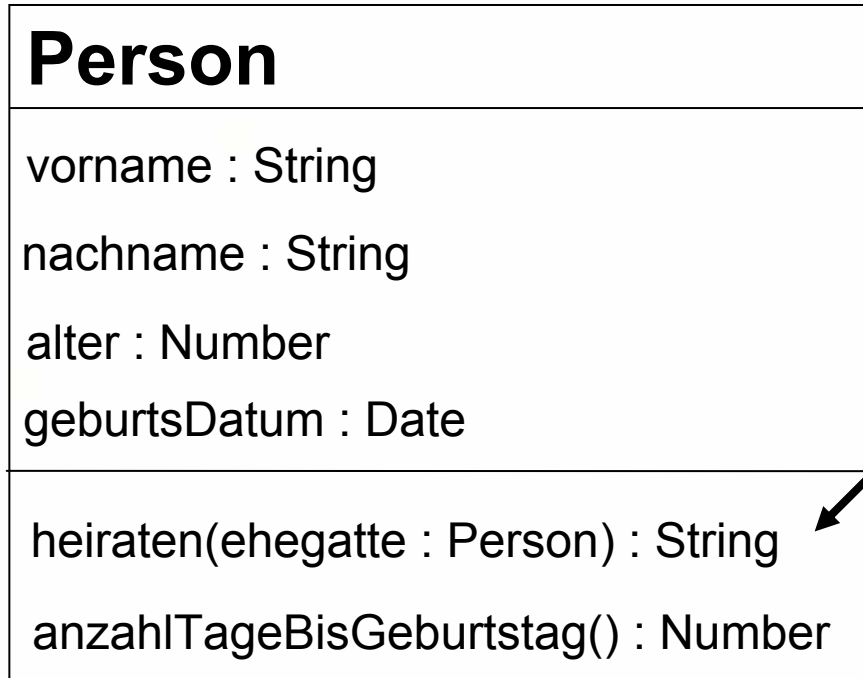
- Methoden können Ergebnis zurückliefern
- Beispiel
 - Verbleibende Anzahl Tage bis zum nächsten Geburtstag
 - *Neuen Namen* bei Heirat
- Mit UML
 - Methoden werden nur definiert (modelliert), aber nicht deren Verhalten implementiert
 - Methoden zum Ändern des Namens, Berechnen der Anzahl Tage: Ändern, Berechnen mit Programmiersprache



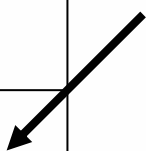
Objekt-Orientierung / UML Methoden

UML

Notation



Opt. Rückgabetypp hinter
Methodenname mit „:“ separiert



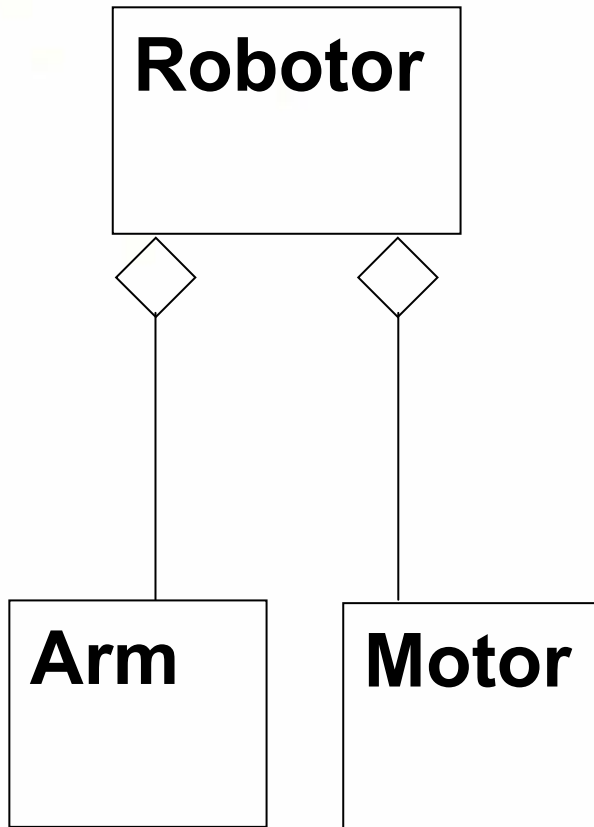


Komposition / Aggregation

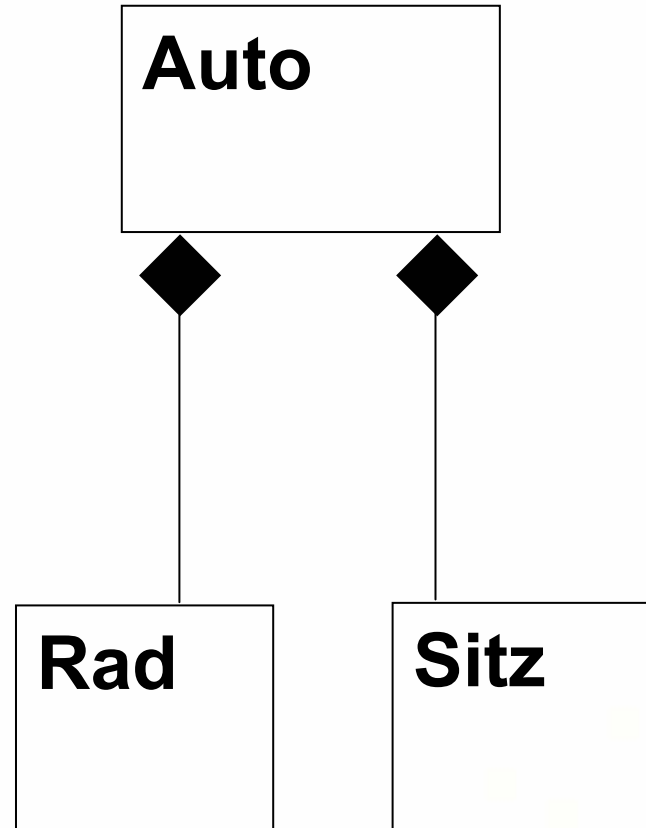
- Software für automatisierte Autofabrik
 - Autos werden von Roboter aus Einzelteilen zusammengesetzt
 - Autos hat Rädern, Sitzen, ...
 - Roboter hat Robotorarm, Motor, ...
- Aggregation
 - Teile hängen lose zusammen
- Komposition
 - Teile sind fest verbunden
- Modellierung hängt vom Anwendungszweck ab
- Auto: Aggregation
- Roboter: Komposition



Komposition



Aggregation





Inhalt

- **Klassen / UML**
 - Eigenschaften
 - Assoziationen
 - Methoden
- **Vererbung**
- **Multiplizitäten bei Assoziationen**

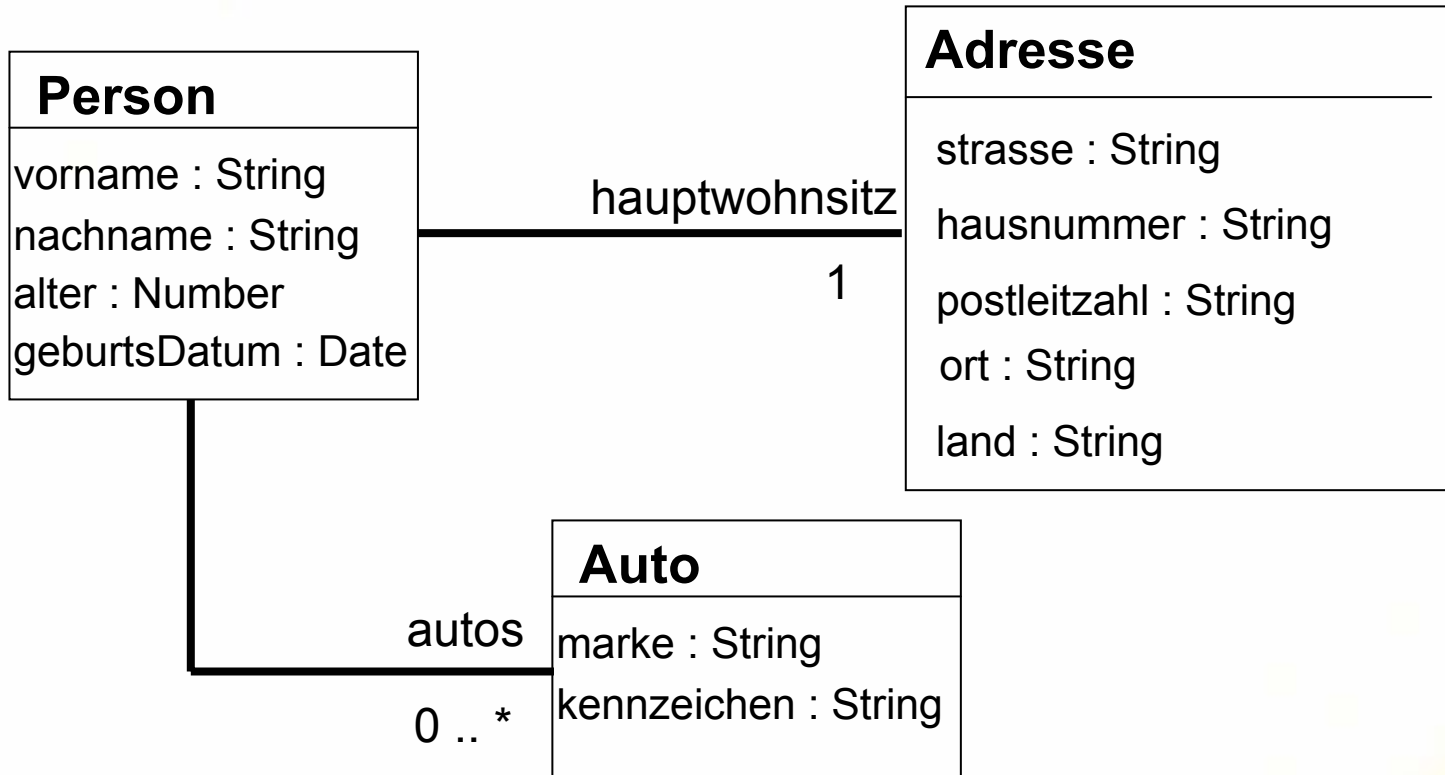


Multiplizitäten

- Beziehungen nicht nur 1-1
 - Person kann mehrere Adressen, Autos, Kinder, usw. besitzen
 - Student besucht mehrere Vorlesungen pro Studiengang
- UML erlaubt Angabe von Anzahl Objekten einer Beziehung
 - Person hat genau eine Adresse
 - Person kann beliebig viele Autos (auch keines) besitzen
- Angabe einer Kardinalität schränkt Implementierungsmöglichkeiten ein

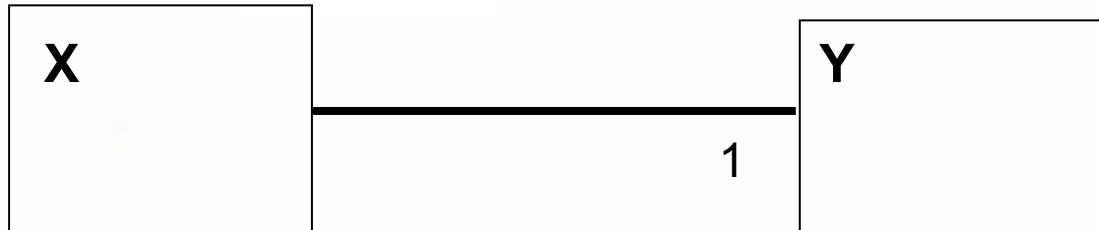


Multiplizitäten





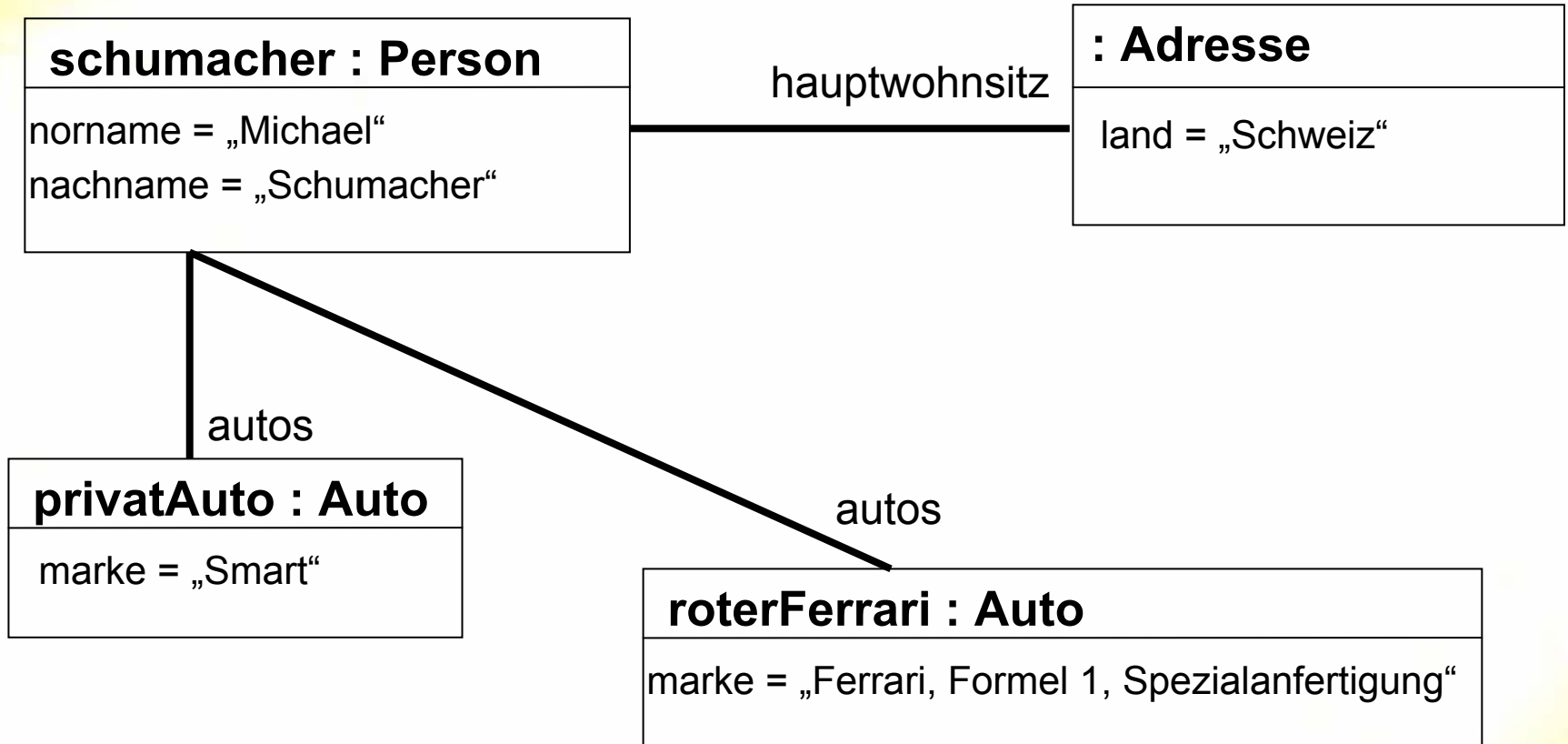
Multiplizitäten



Kardinalität	Bedeutung
1	X hat zu genau einem Y eine Beziehung
0..1	X hat zu höchstens einem Y eine Beziehung
0..*, *	X hat zu beliebig vielen Y eine Beziehung
1..*	X hat zu einem oder mehreren Y eine Beziehung
2..7	X hat zu zwei bis sieben Y eine Beziehung



Multiplizitäten





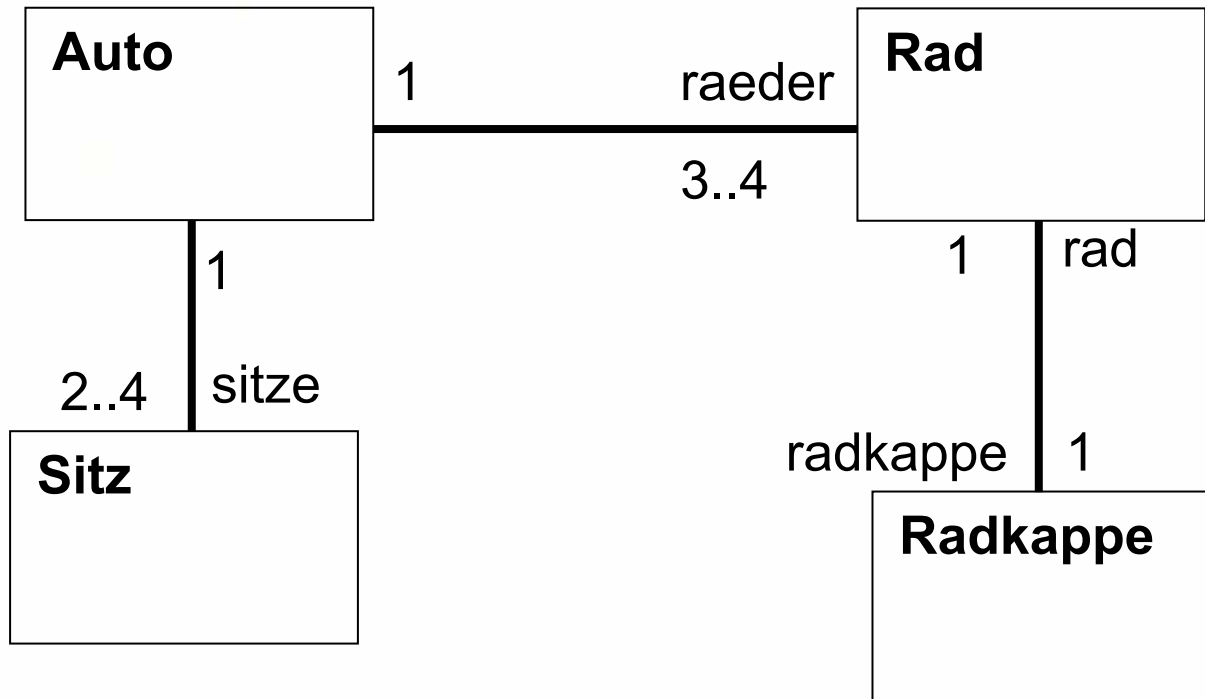
Multiplizitäten

schumacher : Person
vorname = „Michael“
nachname = „Schumacher“

- Objekt „schumacher“ ohne Adresse nicht erlaubt!
 - Implementierung des Entwurf muss sicherstellen, dass nur Personen mit einer Adresse erzeugt werden können
 - Ansonsten wäre Implementierung nicht korrekt: Logischer Programmierfehler



Beispiel / Modellierung Auto





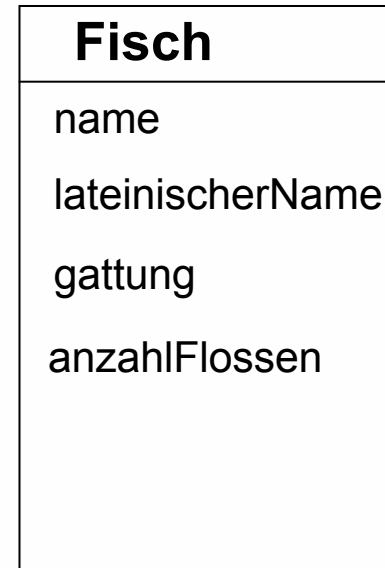
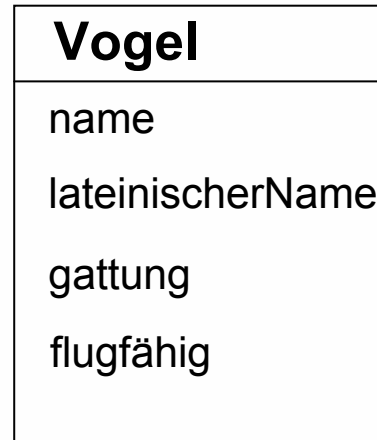
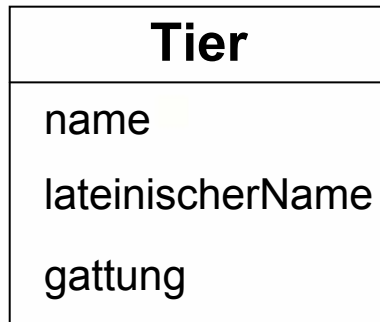
Objekt-Orientierung / Definition (Forts.)

- Vererbung
 - Übernahme von Eigenschaften und Methoden vorhandener Objekte
- Meist geeignet bei Beziehungen von der Form „X ist ein Y“
 - Vogel ist ein Tier
 - Alle Eigenschaften eines Tiers sind auch Eigenschaften eines Vogels
 - Fisch ist ein Tier
 - Blauwahl ist ein Säugetier
- Nur wählen, wenn zusätzliche Eigenschaften oder Verhalten hinzukommen



Objekt-Orientierung / Beispiel

- Modellierung von Tieren, Fischen, Vögeln



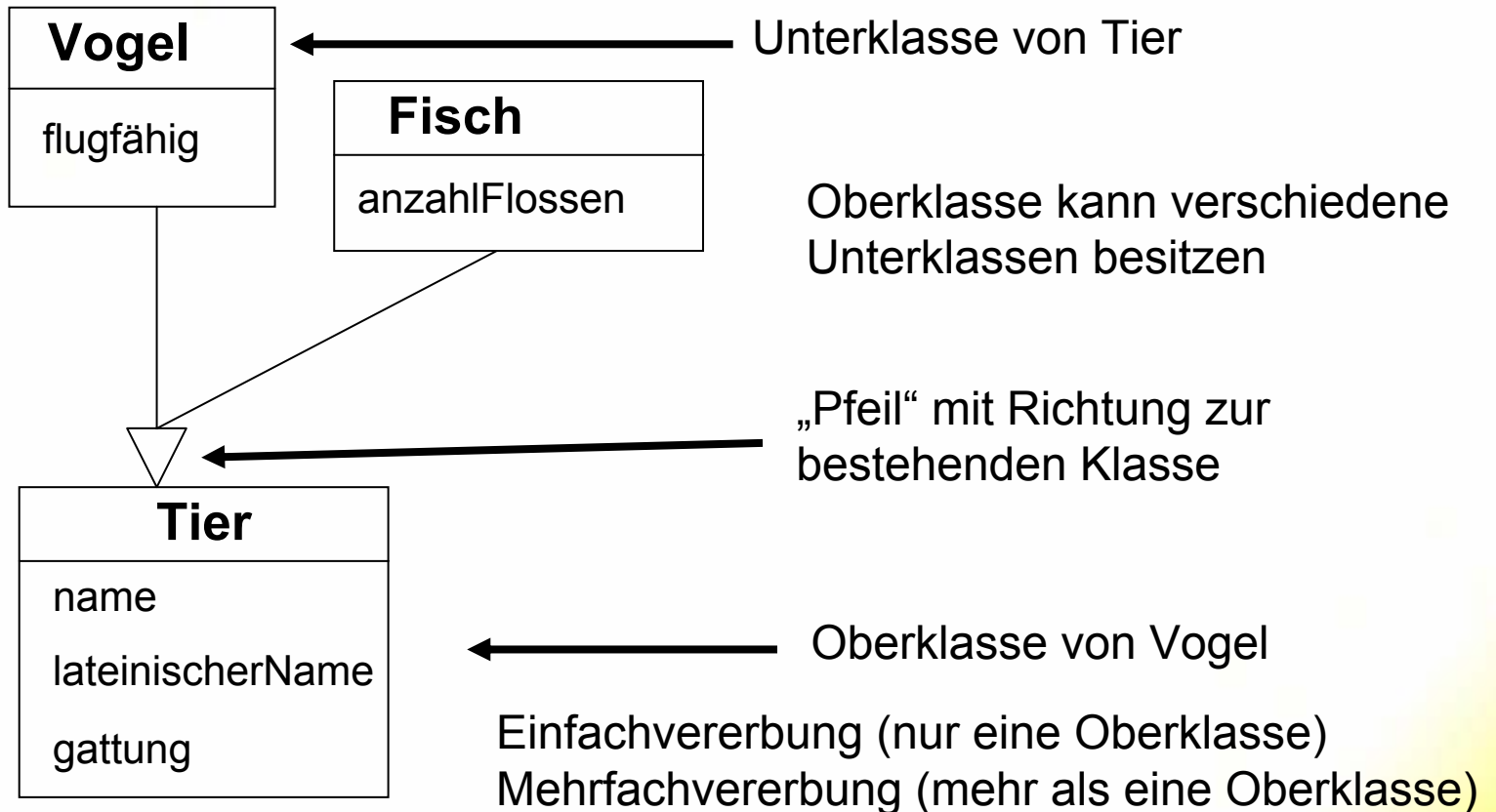
- Unnötige dreifache Modellierung
- Vererbung vermeidet Redundanz



Objekt-Orientierung / UML Vererbung

UML

Notation





Objekt-Orientierung / Objekt-Diagramm

simba : Tier



name = „simba“
lateinischerName = „leo“
gattung = „Panthera “

nemo : Fisch



name = „nemo“
lateinischerName
= „Amphiprion ocellaris “
gattung = „Clownfisch“
anzahlFlossen = 6 ½

duffy : Vogel

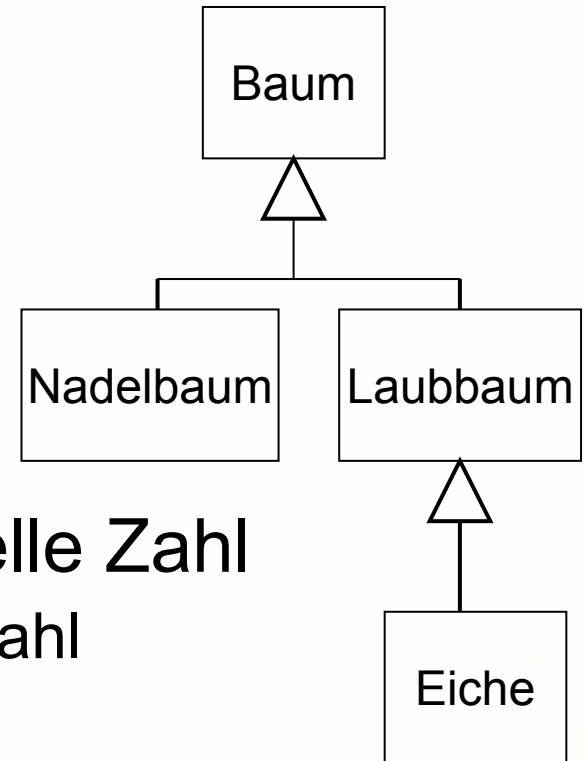


name = „Duffy Duck“
lateinischerName = „Anas platyrhynchos“
gattung = „Anas“
flugfähig = nein (?)



Vererbung / Beispiele

- Ein Nadelbaum ist ein Baum
 - Ein Laubbaum ist ein Baum
 - Eine Eiche ist ein Laubbaum
- Ein Diamant ist ein Edelstein
 - Ein Rubin ist ein Edelstein
- Eine natürliche Zahl ist eine reelle Zahl
 - Eine rationale Zahl ist ein reelle Zahl
- Ein Auto ist ein Kraftfahrzeug
 - Ein Sattelschlepper ist ein Kraftfahrzeug





Zusammenfassung

- **Objekt-Orientierung**
 - Beschreibung der Eigenschaften und Verhalten von Objekten mit Klassen
 - Klassen sind Abstraktion realer oder ideeller Objekte
 - Eigenschaften und Methoden bestehender Klassen (und deren Objekte) können an neue Klassen (und deren Objekte) vererbt werden
- **Modellierung von Klassen/Objekte mit der UML**
- **Weiteres wichtiges OO-Merkmale folgt später**
 - Geheimnisprinzip („Information hiding“)
 - Polymorphie nicht in Informatik I
- **Probleme in Informatik I klein**
 - Oft nur eine Klasse, um Problem zu lösen



UML Werkzeuge

- Marktführer
 - Rational (jetzt bei IBM)
 - Together (jetzt bei Borland), im Softwaretechnik-Labor
- MS Visio
 - reines Zeichenprogramm, unterstützt UML
 - Enterprise Version mit Codeerzeugung
- Diverse andere Anbieter...
- Poseidon for UML
 - Spin-off eines universitären Projektes
 - Abgespeckte freie „Community“ Version
 - <http://gentleware.com>

Beispiel Hochschulsoftware

■ „Anforderungen“

- Hochschule will Stundepläne, Vorlesungen, Studenten, Dozenten, usw. verwalten
 - Stundenpläne sollen ausgedruckt werden können
 - Studenten sollen sich für einen Studiengang immatrikulieren oder exmatrikulieren können
 - Matrikelnummer soll für einen Student vergeben werden
 - Vorlesungen sollen Räume und Zeiten zugeordnet werden können
 - Auskunft über SWS für Semester eines Studierenden
 - Die Wohnadressen aller Studenten sollen verwaltet werden können
 - uvm.

■ Gesucht ist ein erster Entwurf (in UML)

Anforderungen

OO Analyse
/ Design

Entwurf



Anforderungen (1/2)

- Ablauf (Aktivitätsdiagramm erstellen)
 - Bevor sich ein Student an einer Hochschule einschreibt, muss er sich für einen Studienplatz bewerben
 - Die Hochschule entscheidet, dann ob, der Student zugelassen wird oder nicht
 - Im Fall einer Zulassung kann sich der Student einschreiben (oder auch nicht)
 - Bei einer Einschreibung muss der Student in jedem Fall die Gebühren und den Semesterbeitrag zahlen
 - Wenn der Betrag nicht eingezahlt wurde (darüber wacht die Hochschule), dann darf der Student nicht studieren
 - Erst wenn der Betrag eingezahlt worden ist, kann der Student studieren



Beispiel

- Entwurfs- und Analyse „Verfahren“ für Klassendiagramme
 - Anforderungen auf Substantive untersuchen
 - Potentielle Klassen oder Eigenschaften
 - Anforderungen auf Beziehungen zwischen Objekte untersuchen
 - Potentielle Assoziationen
 - Anforderungen auf Verben untersuchen
 - Potentielle Methoden eines Objekts



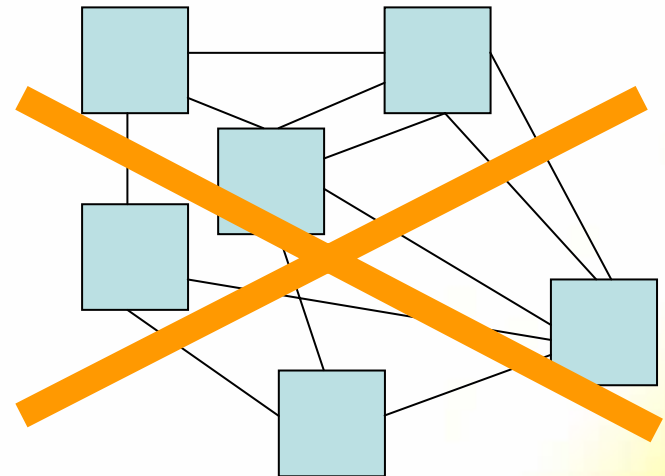
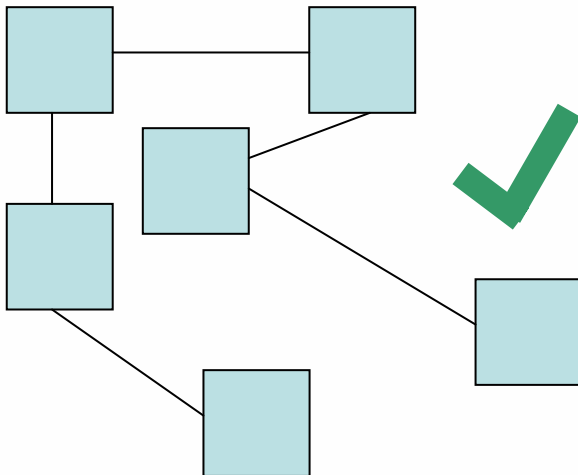
Anforderungen (2/2)

- Klassendiagramm erstellen:
 - Eine Hochschule hat Studenten und Dozenten
 - Studenten haben einen Vor-/Nachname, ein Matrikelnummer und ein Geburtsjahr
 - Die Wohnadressen mit Strasse, Ort und Postleitzahl aller Studenten sollen vorhanden sein
 - Die Matrikelnummer soll von der Hochschule für einen Student vergeben werden
 - Studenten sollen sich für einen Studiengang an der Hochschule immatrikulieren oder exmatrikulieren können



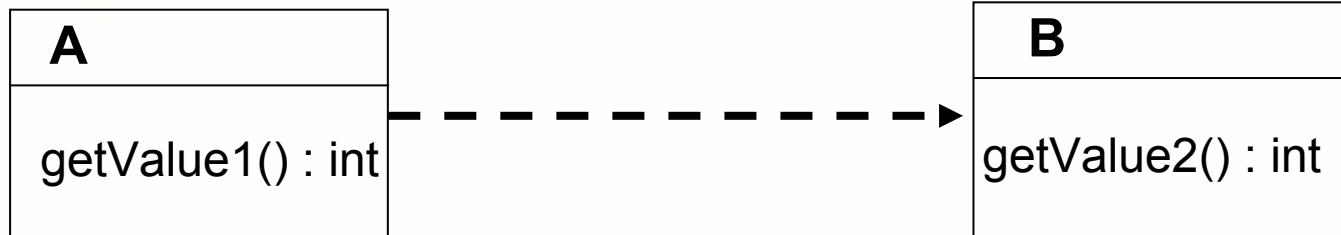
Abhängigkeiten

- **Guter Entwurf:**
 - **Wenig Abhängigkeiten zwischen Klassen**
 - Wenig Assoziationen
 - **Kleine Klassen**
 - Wenig Eigenschaften und Methoden
 - Wenig Programmzeilen



Abhängigkeiten

- Nicht nur Beziehungen sind Abhängigkeitend B
- A verwendet Klasse B (z.B. lokale Variable)
- A ist abhängig von B (lässt sich nur vom Compiler übersetzen wenn B vorhanden ist)
- Wenn B sich ändert, dann muss potentiell A geändert werden



```

class A {

    int getValue1() {
        B b = new B();

        return b.getValue2() + 12;
    }

}
  
```

```

class B {

    int getValue2() {
        return 7;
    }

}
  
```