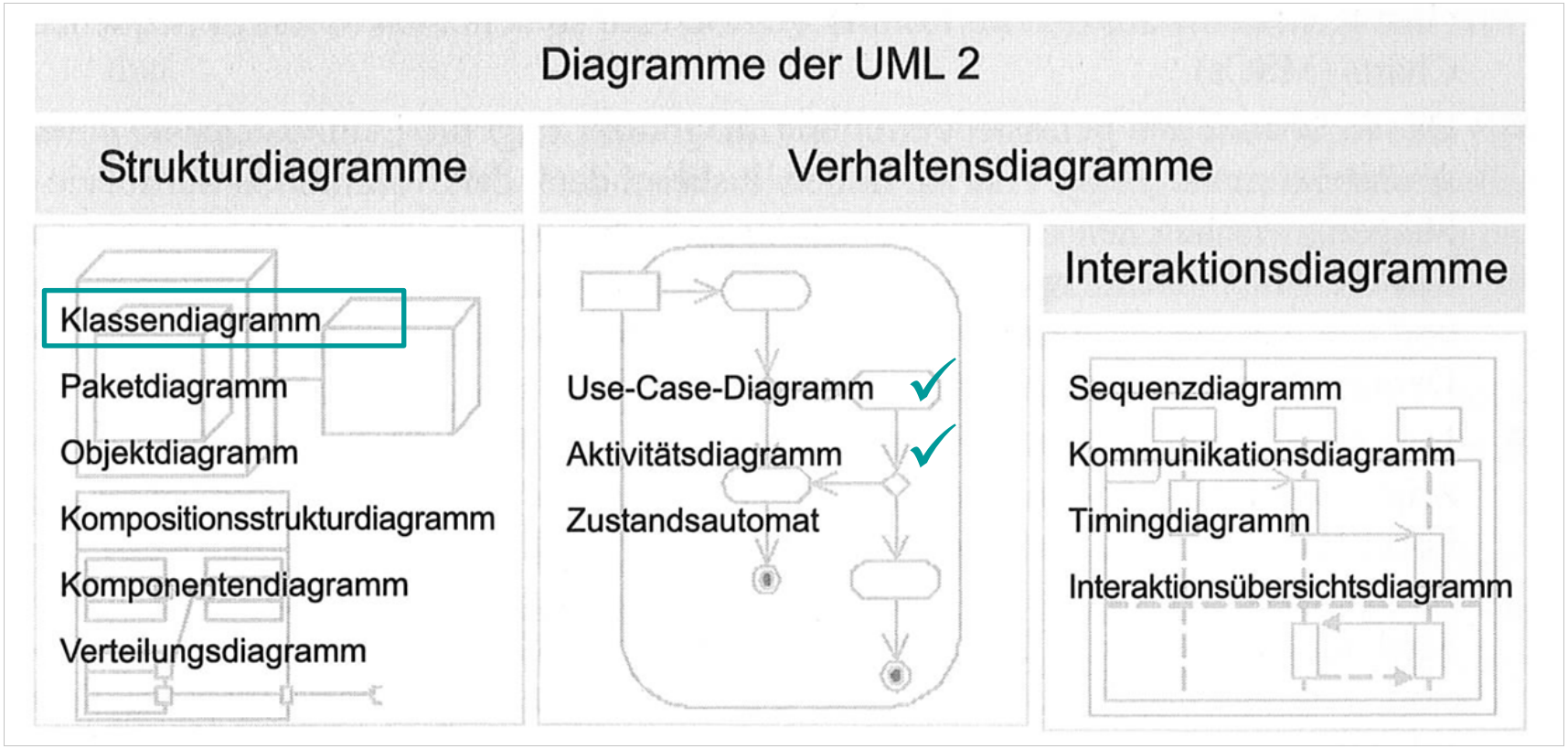
Softwaretechnik

Modellierung mithilfe von Klassendiagrammen

Prof. Dr. Bodo Kraft

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Übersicht UML-Diagramme

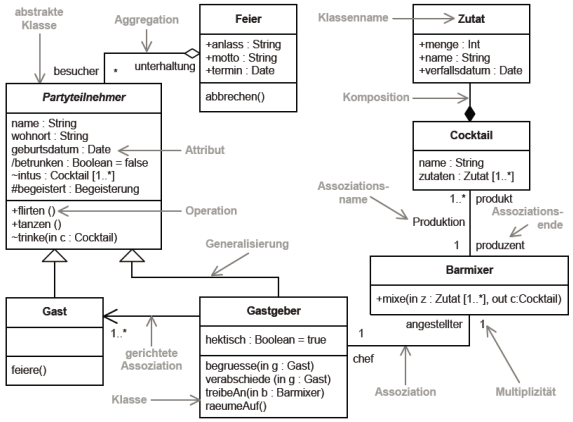


Quelle: UML 2 glasklar, Chris Rupp

Motivation
Klassendiagramme

- Klassendiagramme geben die Möglichkeit die Struktur des zu entwerfenden Systems darzustellen.
- Ein Klassendiagramm zeigt wesentliche statische Eigenschaften des Systems sowie deren Beziehungen zueinander.
- Ein Klassendiagramm gibt Ihnen die Antwort auf die Frage:

„Wie sind die Daten und das Verhalten meines Systems im Detail strukturiert?“

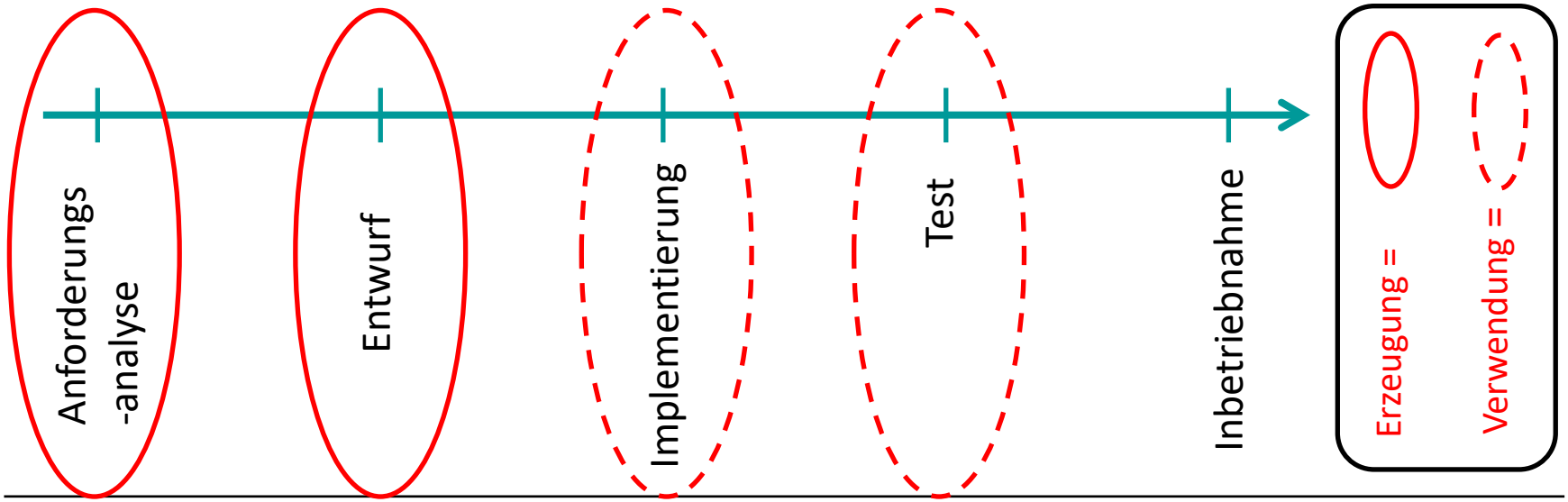


[nach Rupp, UML2 glasklar]

Zeitliche Einordnung in SW-Lifecycle
Klassendiagramme

In welchen Phasen werden Klassendiagramme verwendet?

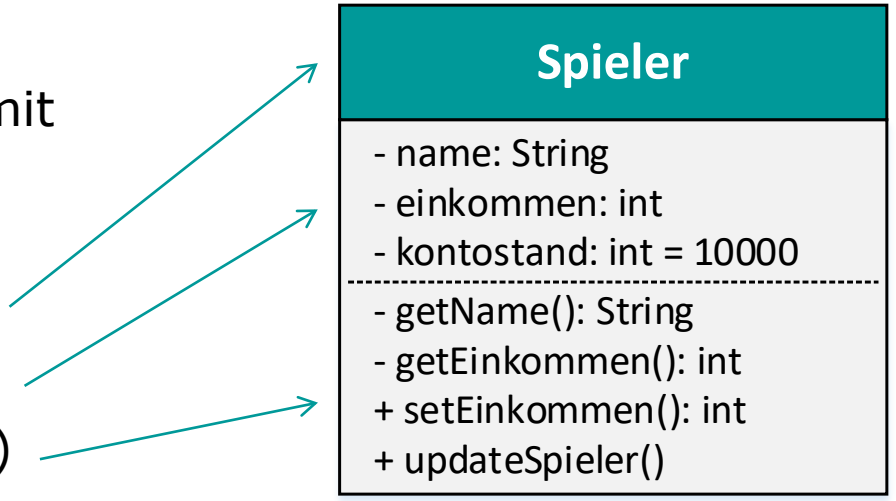
- Das Klassenmodell wird während der Anforderungsanalyse und der Entwurfsphase entwickelt
- Die modellierten Klassen werden in der Implementierungsphase umgesetzt
- Klassenstrukturen werden in der Testphase verifiziert werden



Allgemeine Darstellung von Klassen
Grundlagen (I)

Eine Klasse

- wird dargestellt als Rechteck mit durchgezogenen Linien
- Ist aufgeteilt in drei Bereiche:
  - Klassenname (Kopf)
  - Attribute (Rumpf, Teil1)
  - Operationen (Rumpf, Teil2)



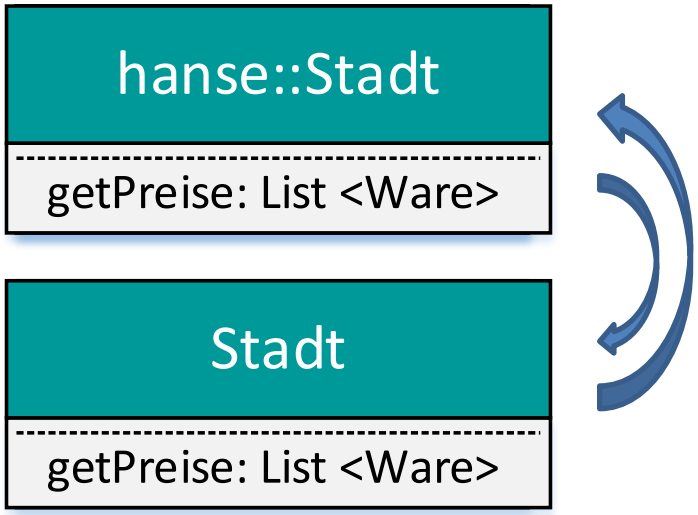
Der Klassenname

- Groß geschrieben
- Fettdruck
- Horizontal mittig zentriert

Attribute und Operationen

- Optional
- Angabe nur wo es sinnvoll ist
- bspw. Weglassen bei: getter/setter - Methoden

Paketnamen & Abstrakte Klassen
Grundlagen (II)

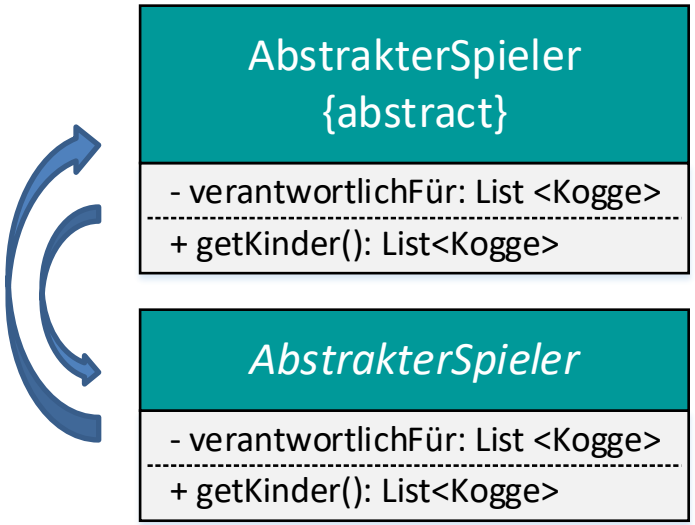


Der Paketname einer Klasse

- kann explizit angegeben werden.
- Wird Klassenname vorangestellt (mit Scope-Operator)

Alternativ Darstellung über Paket-diagramme möglich.

- Klassen können als abstrakt markiert werden.
- Klassen werden kenntlich gemacht
  - *durch Keyword:* ***{abstract}*** *oder*
  - *durch* ***kursive*** **Schreibweise**



Stereotypen im Klassennamen
Grundlagen (III)

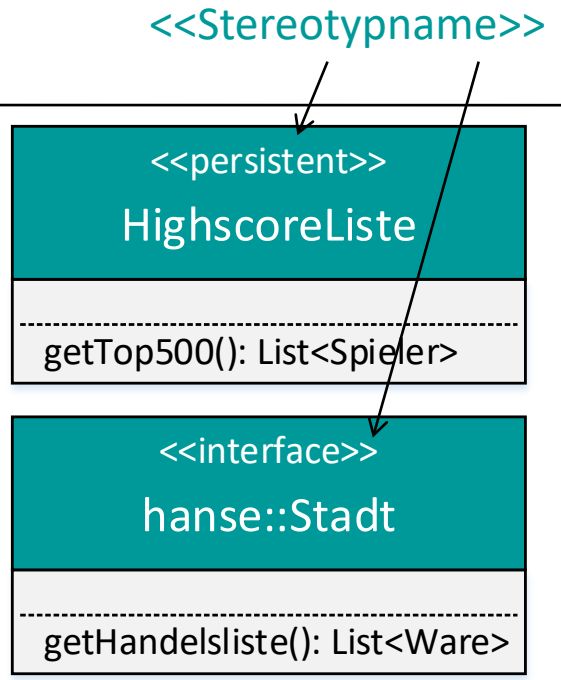
Eine Klasse kann zusätzlich über Stereotypen gekennzeichnet werden.

Stereotypen schränken den Kontext ein, in dem eine Klasse verwendet werden soll/muss.

In UML erfolgt die Darstellung mit Hilfe von **Guillemets** (franz. Anführungs-zeichen).

Stereotypen werden dem Klassennamen vorangestellt.

Schnittstellen(Interfaces in Java) haben kein eigenes Symbol. Sie werden als Stereotyp **<<interface>>** dargestellt.



Beispiele für Stereotypen:

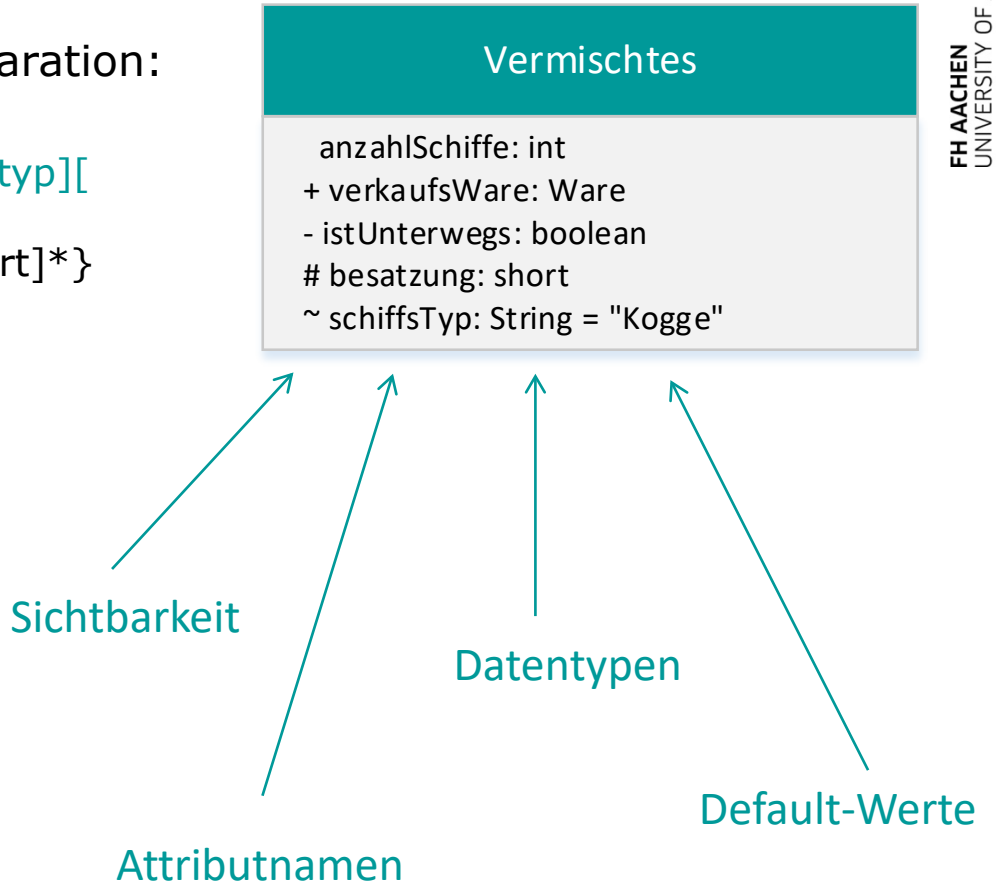
<<interface>>
<<auxiliary>>
<<utility>>
<<persistent>>
<<service>>
<<executable>>
<<encrypted>>
...

Sichtbarkeit, Name, Datentyp
Attributsyntax im Detail (I)

Allgemeine Syntax der Attributdeklaration:

```
[Sichtbarkeit] [/] attributname [: Datentyp][
    [Multiplizität]][= Vorgabewert]
    [{eigenschaftswert [, eigenschaftswert]*}
]
```

Sichtbarkeit	Symbol
public	+
private	-
protected	#
package	~
Abgeleitetes Attribut	/
(default, ohne Symbol)	Nicht explizit angegeben, Für uns: package based



Multiplizitäten, Eigenschaftswerte
Attributsyntax im Detail (II)

Allgemeine Syntax der Attributdeklaration:

[Sichtbarkeit] [/] Attributname [: Datentyp][
    [Multiplizität][= Vorgabewert]
    [{Eigenschaftswert [, Eigenschaftswert]*}]
]

Klassenattribute (**static**) werden unterstrichen!
Attributnamen schreibt man i.A. klein.

Vermischtes

auftragsZiele: String[0..*]{ordered}
bekannteHäfen: String[*]
glückszahl=3.1415 {readonly}
/alter: int {
    datum.heute- datum.bootstaufe}
- seriennummer: String{id}
berufserfahrungKapitän: String[] =
    { "neu", "erfahren", "seebär"}
    { ordered, unique}

Multiplizität	Symbol
Optional, höchst. 1 Wert	0..1
Zwingend, genau 1 Wert	1..1 oder 1
Optional, beliebig viele	0..* oder *
Mind.1, beliebig viele	1..*
Fixiert, mind. n, höchst. m	n..m

Eigenschaftswert	Symbol
Der Wert darf nicht verändert werden	readonly
Attribut kann nach Erzeugung und Initialisierung nicht mehr geändert werden	frozen oder immutable
Inhalte des Attributes in (un-)geordneter Reihenfolge. (default = unordered)	(un)ordered
Inhalte des Attributes treten duplikatfrei auf	unique
Das Attribut macht das Objekt eindeutig. Bei mehreren Ids macht die Kombinationen aller ID-Attribute das Objekt eindeutig.	Id

Eigenschaftswerte
Operationssyntax im Detail (I)

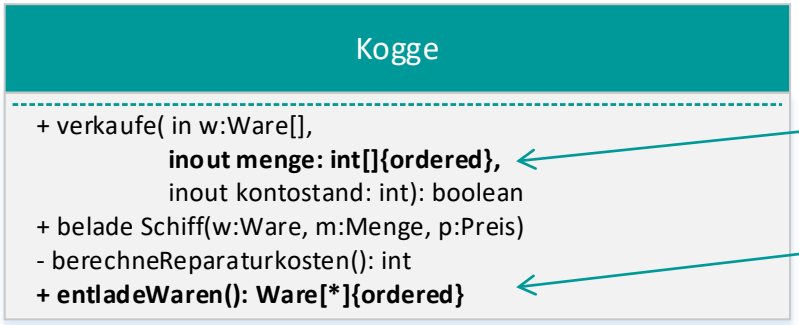
Allgemeine Syntax der Operationsdeklaration:

<Operationsname> ::=
[Sichtbarkeit] Operationsname ([Parameterliste])
[: [Rückgabetyp] [[Multiplizität]] {Eigenschaftswert [, Eigenschaftswert]*}]

<Parameterliste> ::= <Parameter> [, <Parameter>]*

<Parameter> ::=
[Übergaberichtung] Parametername: Typ [[Multiplizität]]
[= Vorgabewert] [{Eigenschaftswert[, Eigenschaftswert]*}]

Symbol	Eigenschaftswert
(anwenderdefinierte Vor, Nach, oder – Methodenbedingungen möglich)	
query	Kennzeichnet Operation als ausschließlich lesend. Keinerlei Daten werden verändert.
ordered	Rückgabewerte der Operation sind geordnet.
unique	Rückgabewerte müssen duplikatfrei sein
redefines <op-Name>	(geerbte) Operation gleichen Namens wird überschrieben



Reihenfolge wichtig für Zuordnung:
Menge→Ware

Reihenfolge wichtig beim Entladen

Übergaberichtung, Sonstiges
Operationssyntax im Detail (I)

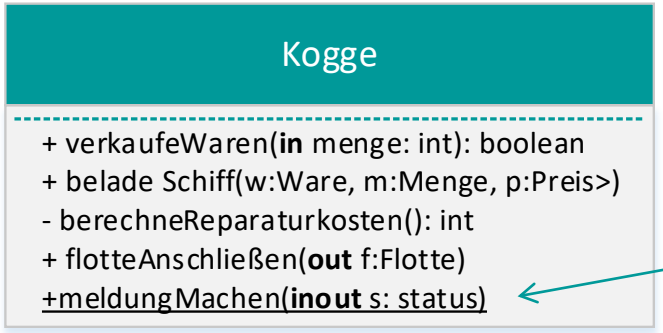
Allgemeine Syntax der Operationsdeklaration:

<Operationsname> ::=
[Sichtbarkeit] operationsname ([Parameterliste])
[: [Rückgabetyp] [[Multiplizität]] {eigenschaftswert [, eigenschaftswert]*}]

<Parameterliste> ::= <Parameter> [, <Parameter>]*

<Parameter> ::=
[Übergaberichtung] parametername: Typ [[Multiplizität]]
[= Vorgabewert] [{eigenschaftswert[, eigenschaftswert]*}]

Übergaberichtung	Symbol
Parameter wird nur ausgelesen	in
Parameter wird schreibend verwendet, ohne Inhalt vorher zu verarbeiten	out
Lesen, Verarbeiten und Neu schreiben	inout
Parameter als „Rückgabewert“	return



Operationsnamen schreibt man am Anfang klein.

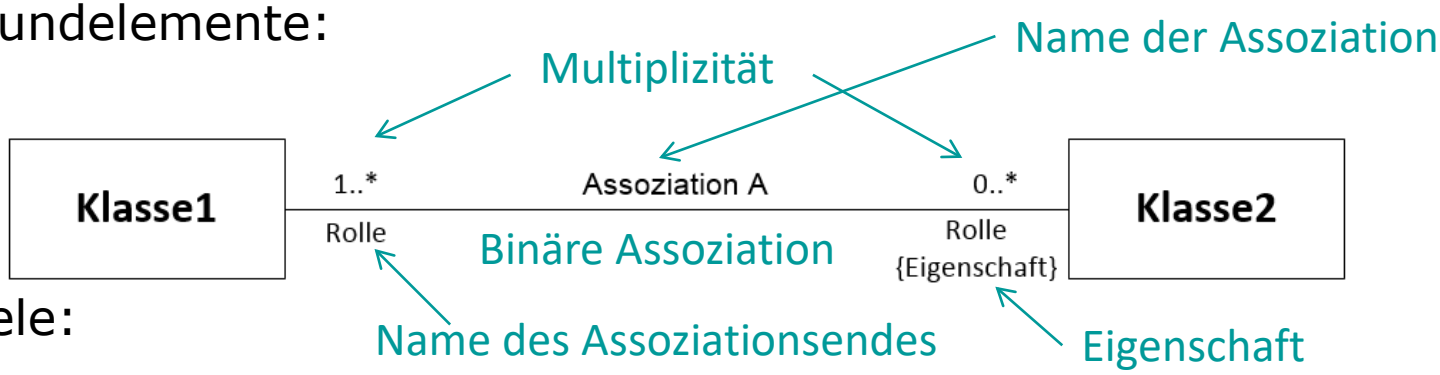
Abstrakte Operationen werden kursiv oder mit dem Eigenschaftswert {abstract} dargestellt.

Klassenoperationen (static) werden unterstrichen.

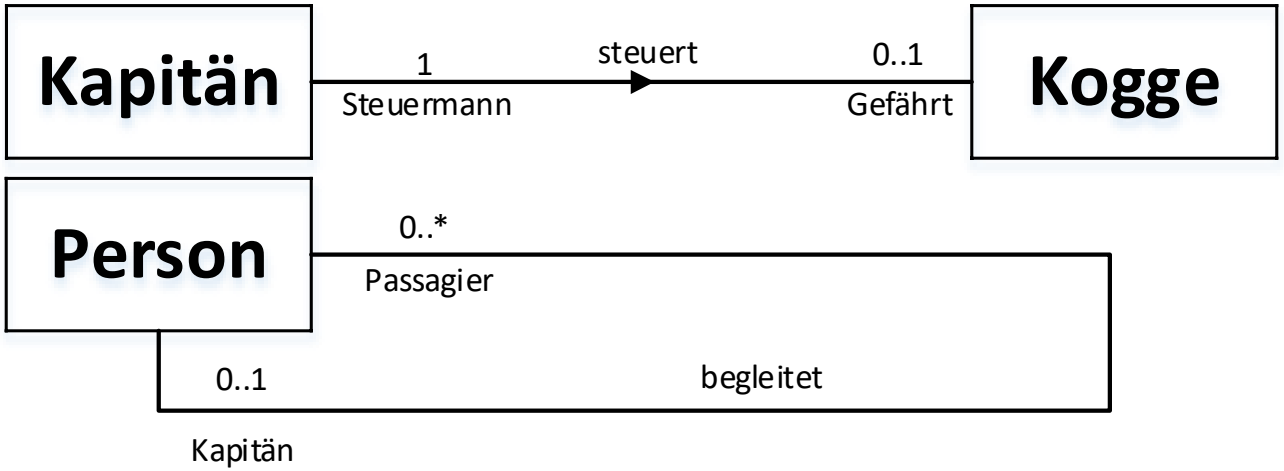
Assoziationen - Allgemein
Beziehungen zwischen Klassen

- Assoziationen verbinden Klassen
- Sie stellen Beziehungen zwischen Klassen dar.

Die Grundelemente:



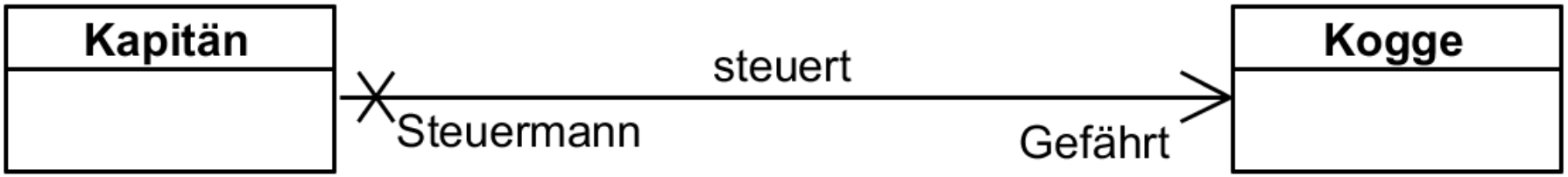
Beispiele:



Assoziationen – Navigierbarkeit
Beziehungen zwischen Klassen

Die Navigierbarkeit von Assoziationen wird an den Pfeilenden festgelegt.

- Keine Angabe: unspezifizierte Navigationsrichtung (d.h. möglich)
- Offene Pfeilspitze: navigierbare Richtung
- Kreuz: keine navigierbare Richtung



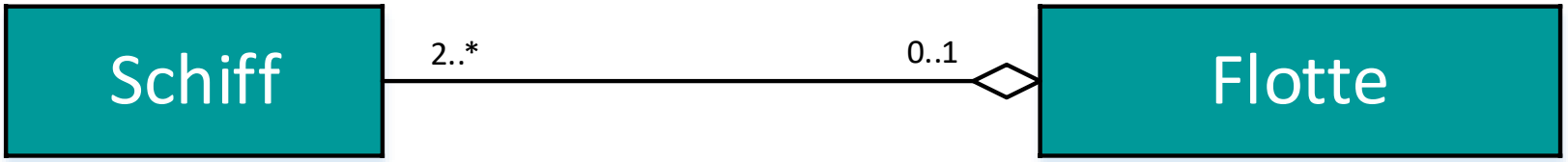
Aggregation und Komposition
Beziehungen zwischen Klassen

Zwei Spezialfälle der Assoziation bilden eine Teil/Ganzes-Beziehung

- Aggregation:

  Ein Objekt(Aggregat) besteht aus mehreren Einzelobjekten.

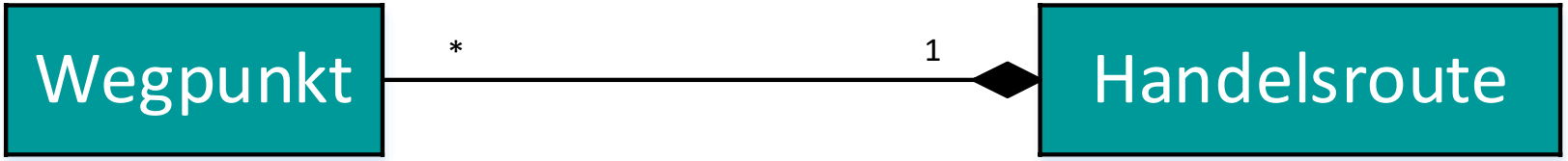
  Die Lebensdauer der Einzelobjekte kann länger sein als das Aggregat.



- Komposition (oder Aggregationskomposition):

  Das *Teilobjekt* ist von der Existenz des *Ganzen* abhängig.

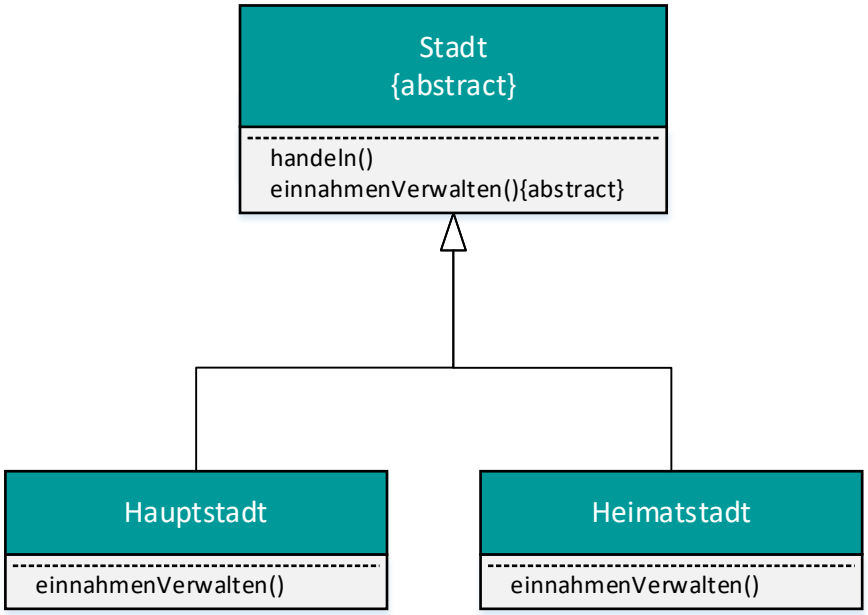
  Es kann nicht ohne existieren.



Generalisierung
Abhängigkeitsbeziehungen zwischen Klassen

Abstrakte Klassen werden verwendet

- um gemeinsame Eigenschaften/Funktionalitäten auszulagern
- Redundanzfreiheit in der Modellierung

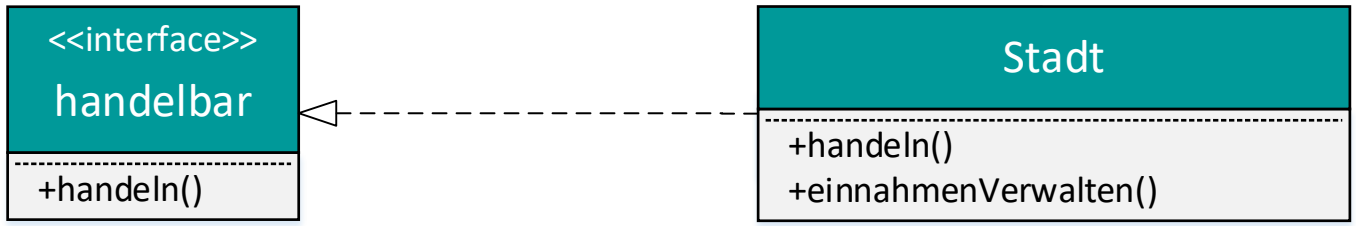


- Darstellung über Generalisierungsbeziehung:
  - durchgezogene Linie
  - weiße geschlossene Pfeilspitze

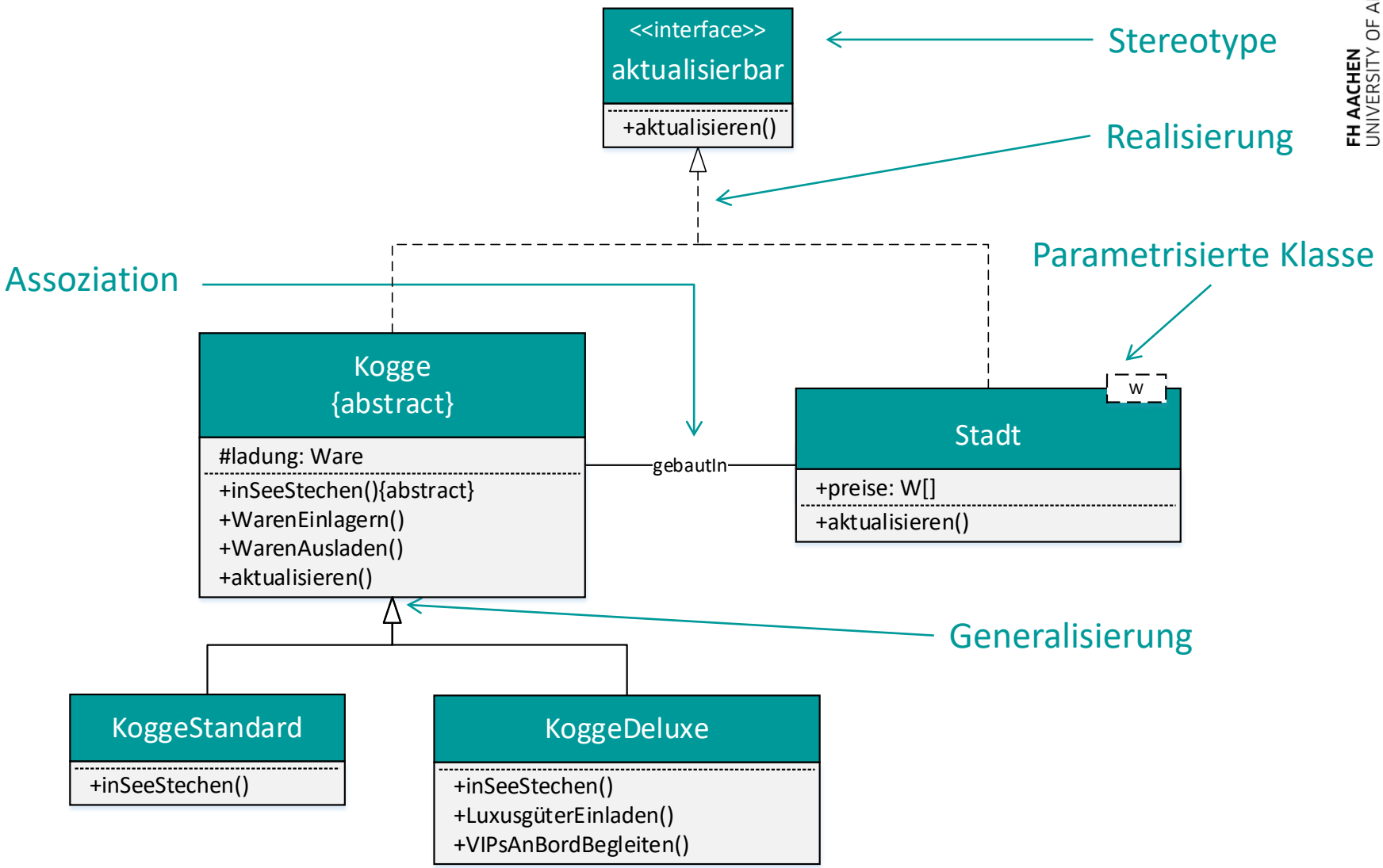
Realisierung
Abhängigkeitsbeziehungen zwischen Klassen

- Schnittstellen (Interfaces) sind ähnlich zu abstrakten Klassen, aber
  - Sind nicht instanziierbar
  - Keine privaten Eigenschaften
  - Methoden/Operationen werden nicht implementiert

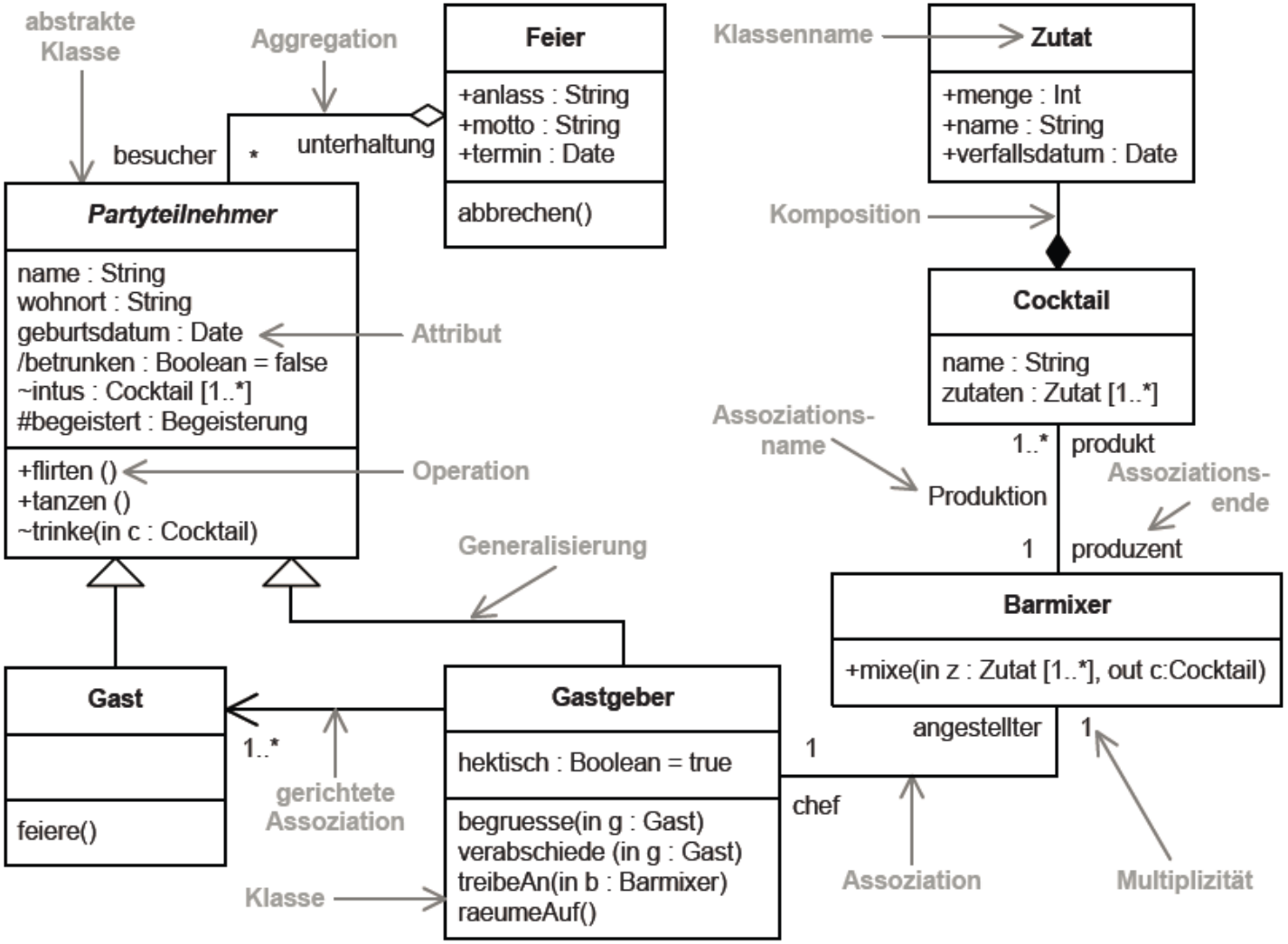
- Klassen realisieren Schnittstellen, indem sie die vorgegebenen Methoden implementieren
  - In UML Darstellung über die Realisierungsbeziehung:
    - Assoziation als
      - gestrichelte Linie
      - Mit geschlossener Pfeilspitze



Assoziationen auf einen Blick
Beziehungen zwischen Klassen



Beispiel: Party



Quelle: UML 2 glasklar

Literaturangaben:

- [RS] C. Rupp, SOPHIST GROUP, Requirements- Engineering und – Management, Hanser Fachbuchverlag, 2004
- [OW] B. Oestereich, C. Weiss, C. Schröder, T. Weilkiens, A. Lenhard, Objektorientierte Geschäftsprozessmodellierung mit der UML, dpunkt.Verlag, 2003

Vielen Dank!