Technische Universität Berlin

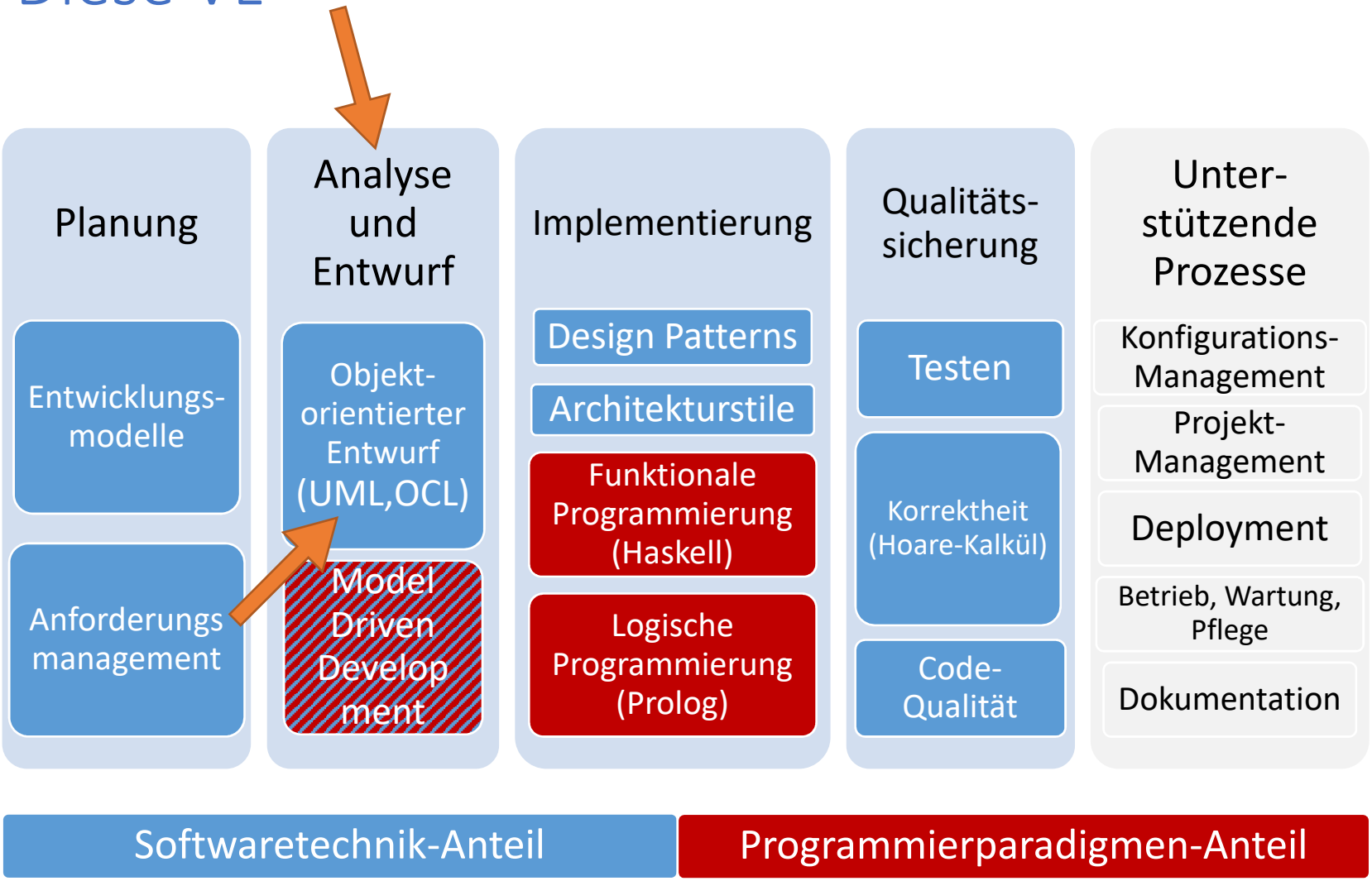
Softwaretechnik und Programmierparadigmen

07 Analyse und Entwurf - Struktur

Prof. Dr. Sabine Glesner
Software and Embedded Systems Engineering
Technische Universität Berlin

SESE

Diese VL


Planung
Entwicklungs-modelle
Anforderungs management
Analyse und Entwurf
Objekt-orientierter Entwurf (UML,OCL)
Model Driven Develop ment
Implementierung
Design Patterns
Architekturstile
Funktionale Programmierung (Haskell)
Logische Programmierung (Prolog)
Qualitäts-sicherung
Testen
Korrektheit (Hoare-Kalkül)
Code-Qualität
Unter-stützende Prozesse
Konfigurations-Management
Projekt-Management
Deployment
Betrieb, Wartung, Pflege
Dokumentation
Softwaretechnik-Anteil
Programmierparadigmen-Anteil

Inhalt

Analyse und Entwurf - Struktur

- Grundlagen
- Klassendiagramm
- Objektdiagramm

Inhalt

Analyse und Entwurf - Struktur

- Grundlagen
- Klassendiagramm
- Objektdiagramm

Von den Anforderungen zum Entwurf

Requirements Engineering (letzte VL)

- Herleitung der Anforderungen mithilfe von **Verhaltensmodellen**
- Stärken und Schwächen der Anforderungen aufzeigen und überarbeiten

Entwurf (diese VL)

- Beschreibung des Systems für die Entwickler mithilfe von **Struktur und Verhaltensmodellen**
- **Vorbereitung** der Implementierungsphase

Das Systemmodell ist eine **Abstraktion** und lässt bewusst Details aus!

- Vereinfachung und Konzentration auf hervorstechende Merkmale

„Essentially, all models are **wrong**, but some are **useful**."
George E. P. Box

Analyse und Entwurf

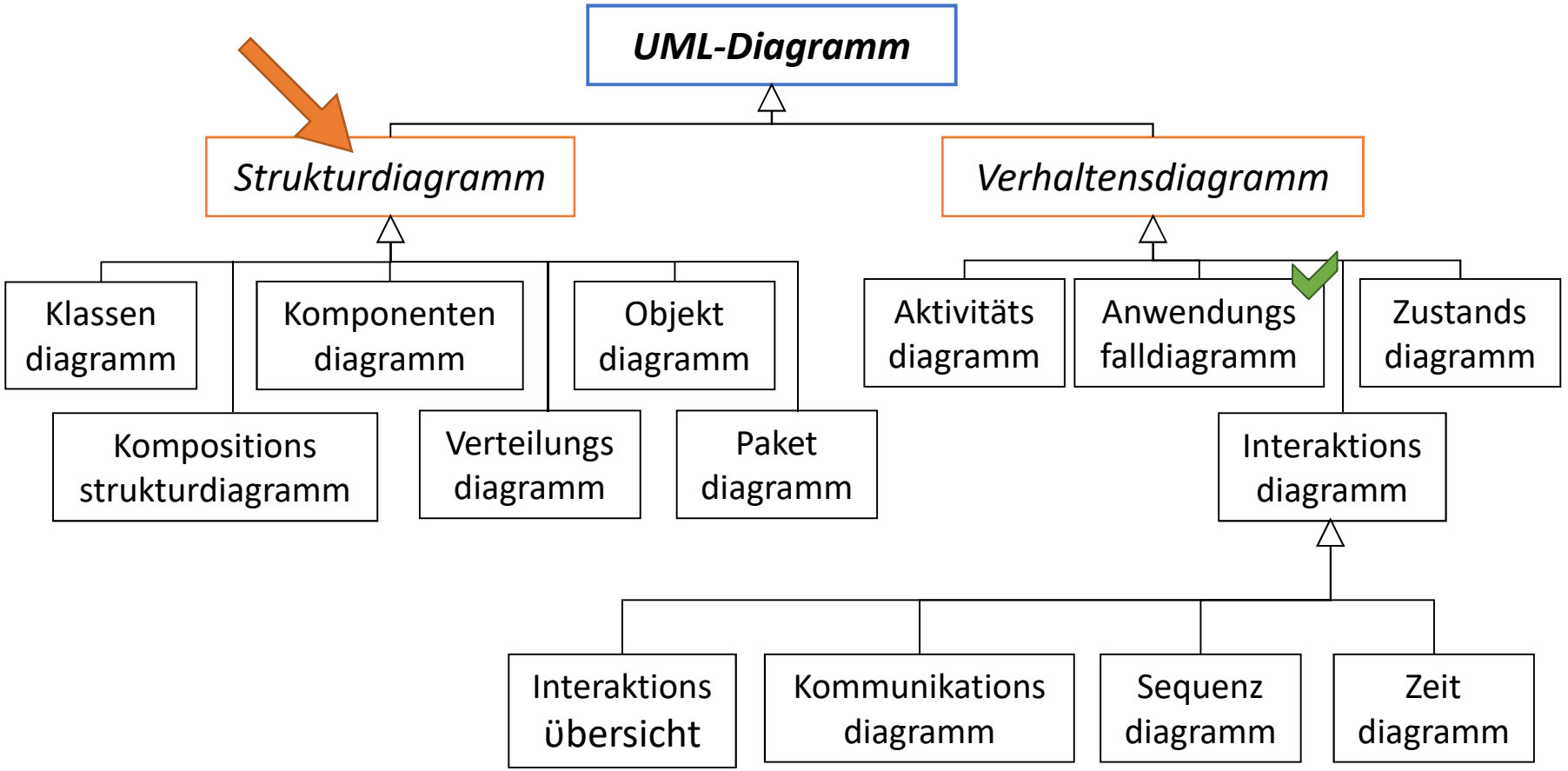
Struktur

- Entwurf der **Organisation des Systems**
- Beschreibung von **Zuständen**
- Bilden die Komponenten und Beziehungen zwischen den Komponenten ab
- Diese VL

Verhalten

- Modelle des dynamischen **Verhaltens des Systems**
- Beschreibung von **Zustandsübergängen**
- Zeigen was passiert / passieren soll, wenn das System auf Reize reagiert
- Nächste VL

UML Diagrammübersicht



Übersicht Strukturdiagramme

Paketdiagramme (Package Diagrams)

- Partitionierung des Modells in Pakete
- Aggregations- und Gen/Spec-Beziehungen zwischen Paketen
- Importbeziehungen:
  - Import einzelner Elemente mit <<access>>
  - Import ganzer Pakete mit <<import>>
- Darstellung der hierarchischen Struktur des Systems

Komponentendiagramme (Component Diagrams)

- Komponenten: ausführbare Klassen, kapseln internen Aufbau, stellen Verhalten über Schnittstellen und Ports zur Verfügung
- Komponentendiagramme stellen die Komponenten und deren Interaktion dar (Ports und Schnittstellen)
- Darstellung der funktionalen Struktur (Software Architecture)

Übersicht Strukturdiagramme

Kompositionsstrukturdiagramme (Composite Structure)

- Konfiguration von miteinander verbundenen Laufzeitelementen
- z.B. Zusammenarbeit von Klassen oder Objekten zur Erfüllung einer bestimmten Aufgabe

Verteilungsdiagramme (Deployment Diagrams)

- Beschreibung der physikalischen Struktur (Topologie) von verteilten Systemen
- Modellierung aller im realen System tatsächlich vorhandenen Hardware- und Software-Knoten und deren Verbindungen (Kommunikationspfade)
- Physikalische Struktur setzt sich zusammen aus
  - Artefakten (physikalische Informationseinheiten, z.B. Dateien)
  - HW- und SW-Knoten (Ausführungseinheiten, z.B. Geräte)
  - Kommunikationspfaden (physikalische Verbindungen)

Strukturdiagramme dieser VL

Klassendiagramm

- Zeigt die **Objektklassen** im System und deren **mögliche** Beziehungen
- Beschreibt **alle möglichen** Zustände des Systems
- Kann für **Entwurf** und **Dokumentation** verwendet werden
- **Statisch** (Compile-Zeit)

Objektdiagramm

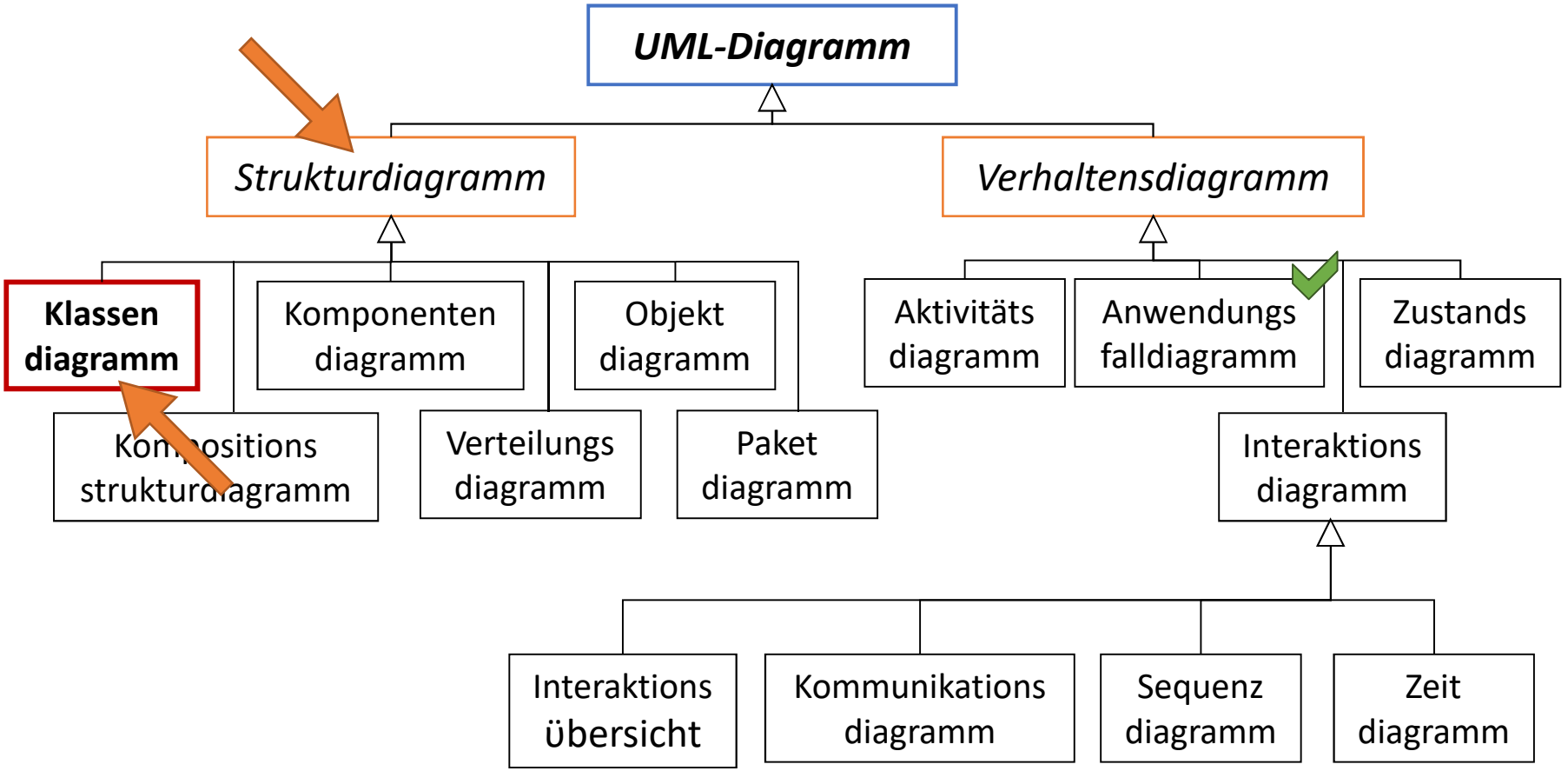
- Zeigt tatsächlich existierende **Objekte** im System und deren **tatsächlichen** Beziehungen
- Beschreibt **genau einen** Zustand des Systems
- Kann für **Dokumentation** und **Debugging** verwendet werden, nicht für Entwurf
- **Dynamisch** (Laufzeit)

Inhalt

Analyse und Entwurf - Struktur

- Grundlagen
- Klassendiagramm
- Objektdiagramm

UML Diagrammübersicht



Einführung Klassendiagramme

Klassendiagramme erlauben die Modellierung abstrakter **objektorientierter** Konzepte, die unabhängig von der tatsächlichen Implementierung sind

- Objektorientierung bekannt von objektorientierten Programmiersprachen wie Java
- Bspw. wird tatsächliche Implementierung von Methoden oder Vererbung nicht vorgegeben

Wesentliche Modellierungselemente

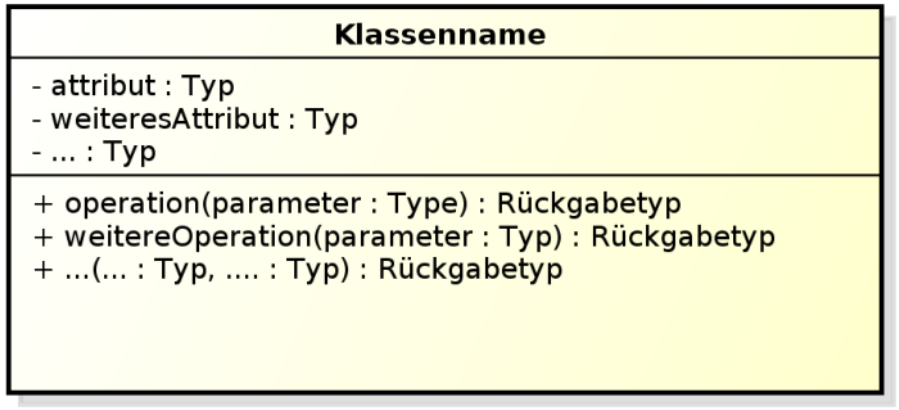
- Klassen
- Attribute
- Assoziationen
- Stereotype

Klassen

Bilden ein abstraktes Modell für eine Menge von ähnlichen Objekten

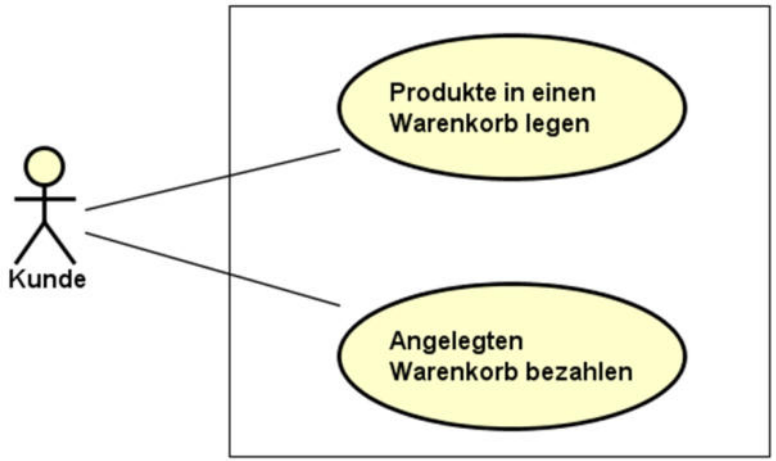
- Erlauben Instanziierung mehrerer Objekte desselben Typs

Darstellung in UML:



Fallbeispiel

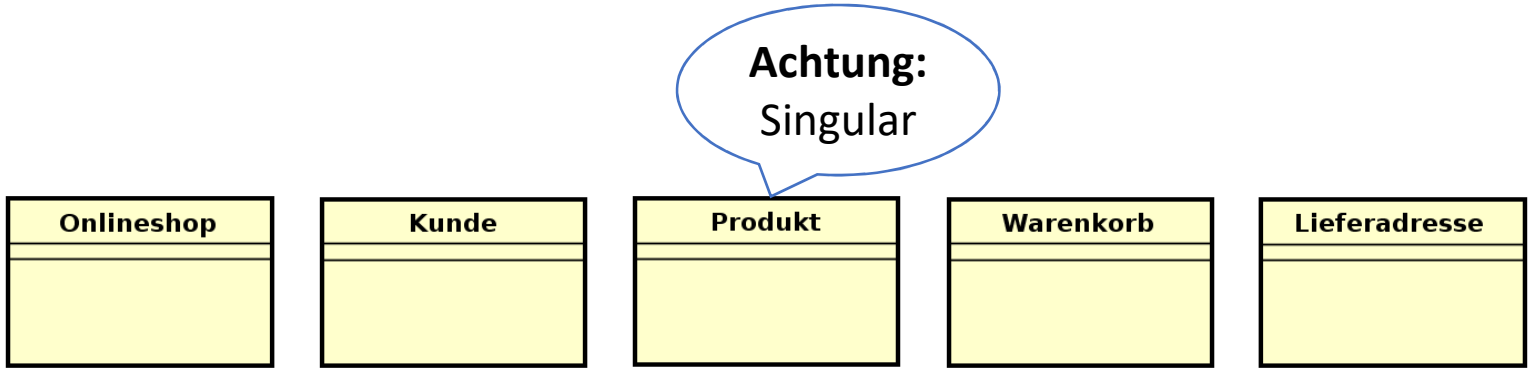
Sie werden gebeten für einen kleines Unternehmen, das Schuhe und Kleidung verkauft, die Verwaltungssoftware eines Online-Shops zu entwickeln. Der Onlineshop soll es dem Kunden ermöglichen, Produkte in einen Warenkorb zu legen und diesen zu bezahlen. Jeder Kunde hat mindestens eine Lieferadresse.



Was sind **sinnvolle Klassen** dieses Systems?

Fallbeispiel

Sie werden gebeten für einen kleines Unternehmen, das Schuhe und Kleidung verkauft, die Verwaltungssoftware eines Online-Shops zu entwickeln. Der Onlineshop soll es dem Kunden ermöglichen, Produkte in einen Warenkorb zu legen und diesen zu bezahlen. Jeder Kunde hat mindestens eine Lieferadresse.

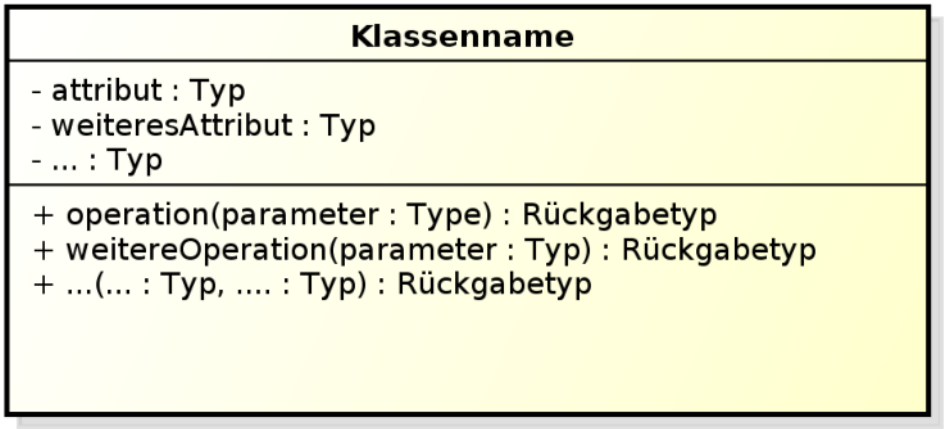


Attribute

Klassen können getypte Attribute enthalten

Attribute können mit einer Sichtbarkeit versehen werden

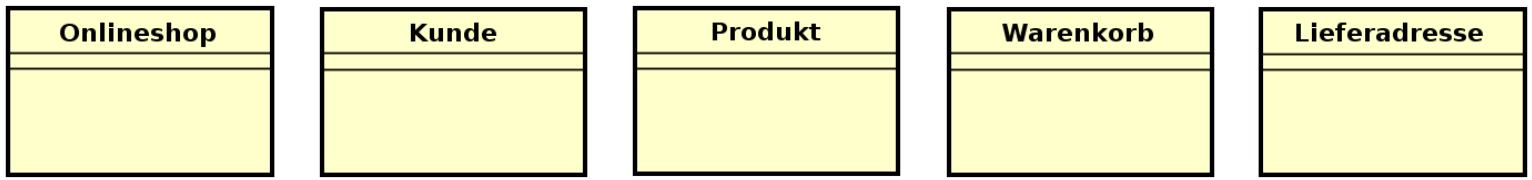
- public (+): jede andere Klasse kann auf eigenes Attribut zugreifen
- private (-): Zugriff nur von eigener Klasse
- protected (#): Zugriff nur von eigener Klasse und Unterklassen



Fallbeispiel (mehr Details)

Sie werden gebeten für einen kleines Unternehmen, das Schuhe und Kleidung verkauft, die Verwaltungssoftware eines Online-Shops zu entwickeln. Der Onlineshop soll es dem Kunden ermöglichen, Produkte in einen Warenkorb zu legen und diesen zu bezahlen. Jeder Kunde hat mindestens eine Lieferadresse.

Im System wird der Name und eine Email-Adresse jedes Kunden des Onlineshops hinterlegt. Außerdem wird für jedes Produkt eine Produktnummer und eine Bezeichnung sowie der Preis und die Lagermenge gespeichert.



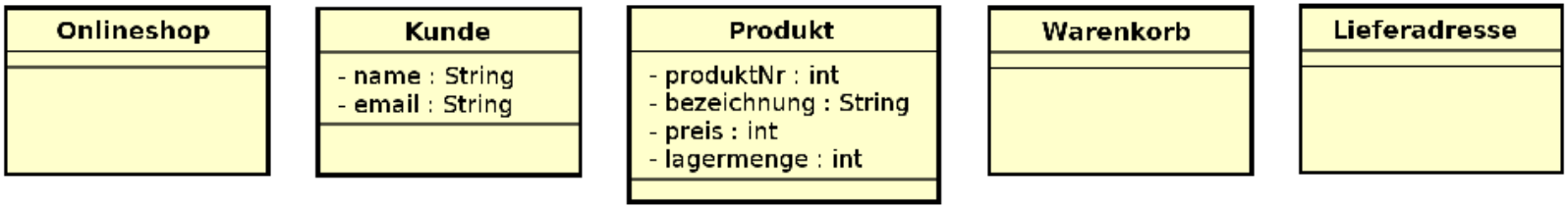
Welche **Attribute** werden angegeben?

Fallbeispiel (mehr Details)

Sie werden gebeten für einen kleines Unter
und Kleidung verkauft, die Verwaltungssof
zu entwickeln. Der Onlineshop soll es dem K
Produkte in einen Warenkorb zu legen und diesen zu be
Jeder Kunde hat mindestens eine Lieferadresse.

Warum sind die Kunden **kein Attribut** des Onlineshops?

Im System wird der Name und eine Email-Adresse jedes Kunden des Onlineshops hinterlegt. Außerdem wird für jedes Produkt eine Produktnummer und eine Bezeichnung sowie der Preis und die Lagermenge gespeichert.



Einfache Assoziationen

Stellen die Beziehungen zwischen Klassen dar

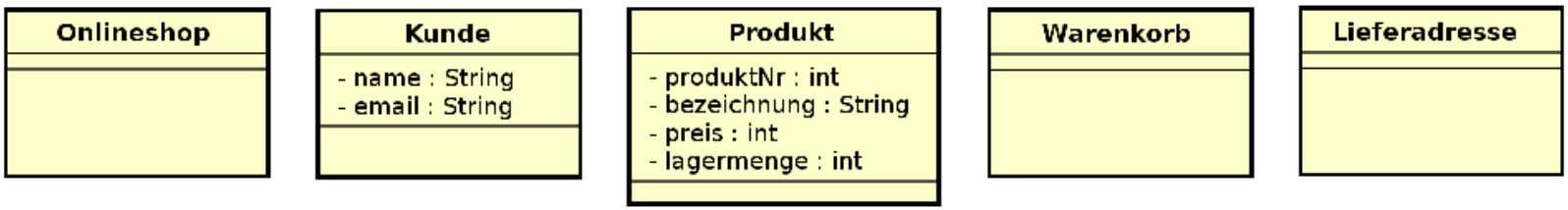
- Assoziationsbezeichnung hat eine Leserichtung (optional, aber hilfreich)

Assoziationen können als Objektreferenzen bzw. durch verschiedene Container implementiert werden

Fallbeispiel (mehr Details)

Sie werden gebeten für einen kleines Unternehmen, das Schuhe und Kleidung verkauft, die Verwaltungssoftware eines Online-Shops zu entwickeln. Der Onlineshop soll es dem Kunden ermöglichen, Produkte in einen Warenkorb zu legen und diesen zu bezahlen. Jeder Kunde hat mindestens eine Lieferadresse.

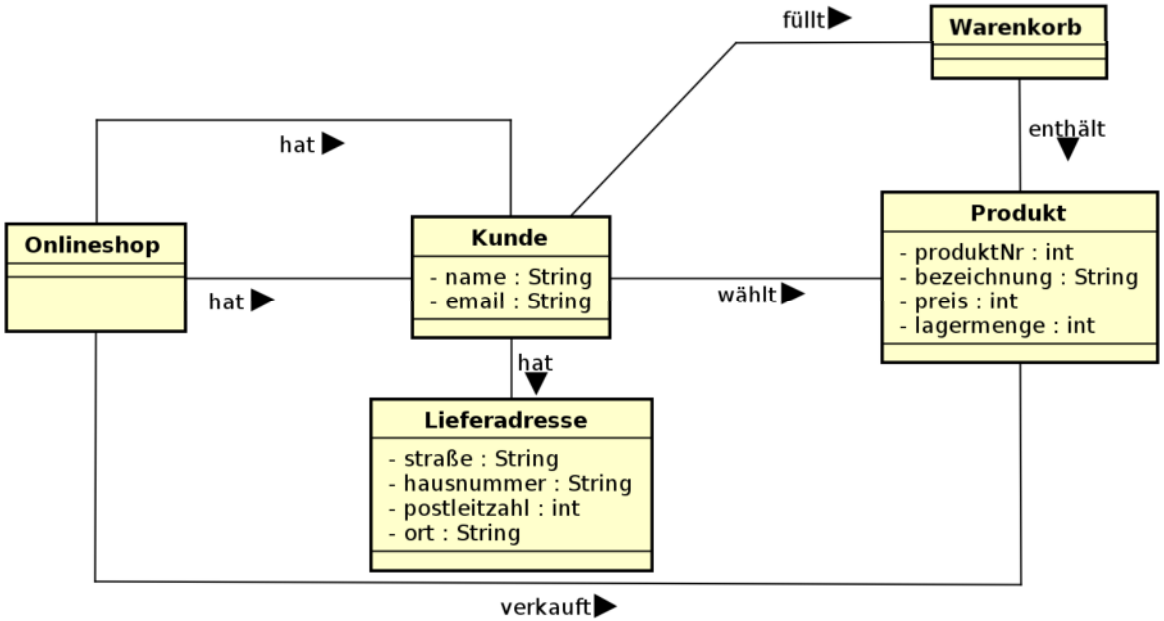
Im System wird der Name und eine Email-Adresse jedes Kunden des Onlineshops hinterlegt. Außerdem wird für jedes Produkt eine Produktnummer und eine Bezeichnung sowie der Preis und die Lagermenge gespeichert.



Welche **Assoziationen** fehlen?

Fallbeispiel (Assoziationen)

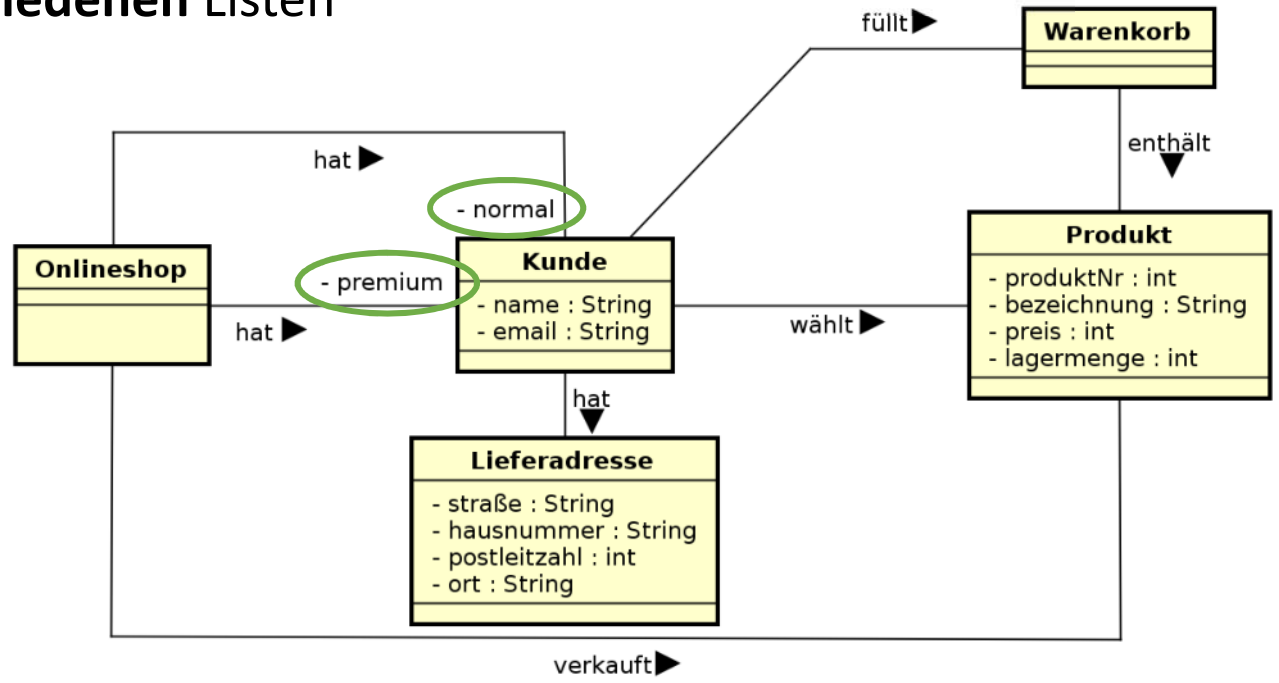
Sie werden gebeten für einen kleines Unternehmen, das Schuhe und Kleidung verkauft, die Verwaltungssoftware eines Online-Shops zu entwickeln. Der Onlineshop soll es dem Kunden ermöglichen, Produkte in einen Warenkorb zu legen und diesen zu bezahlen. Jeder Kunde hat mindestens eine Lieferadresse.



Rollen

Verschiedene Rollen (Namen am Assoziationsende) helfen mehrdeutige Assoziationen zu unterscheiden und Assoziationen passend zu benennen

Der Onlineshop führt normale und Premiumkunden auf zwei **verschiedenen** Listen



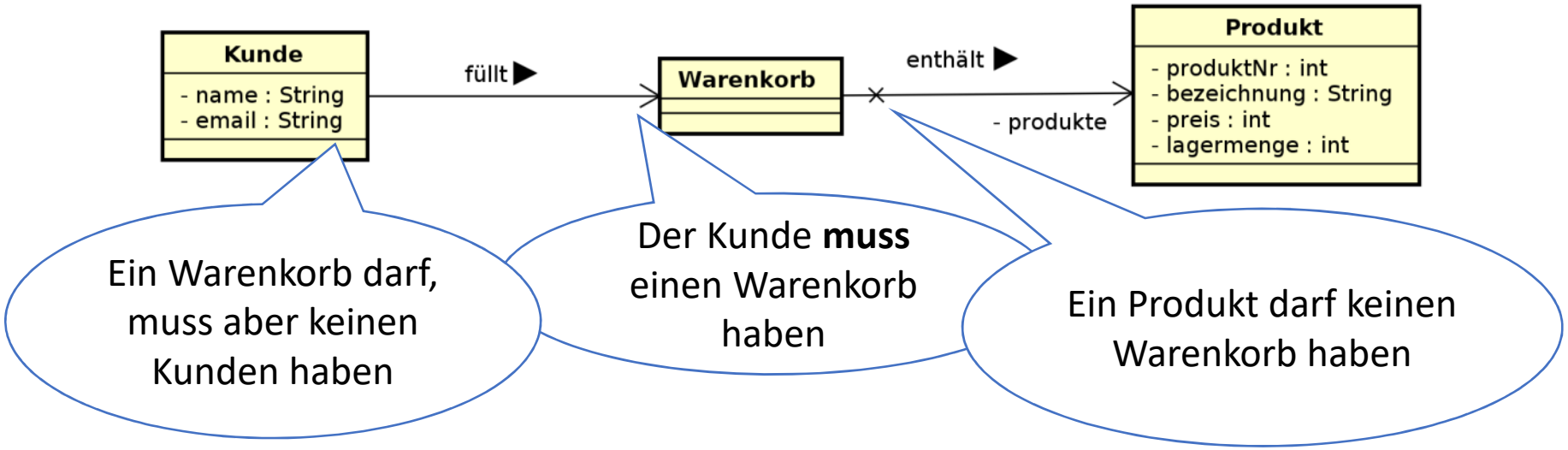
Navigierbarkeit

Gibt an, in welche Richtung die Assoziation (nicht) gelten soll

→: In diese Richtung **existiert** die Assoziation (als **Attribut**)

X: In diese Richtung darf sie **nicht gelten** (Es gibt kein **Attribut**)

Wenn keine Navigierbarkeit angegeben ist, ist sie unspezifiziert

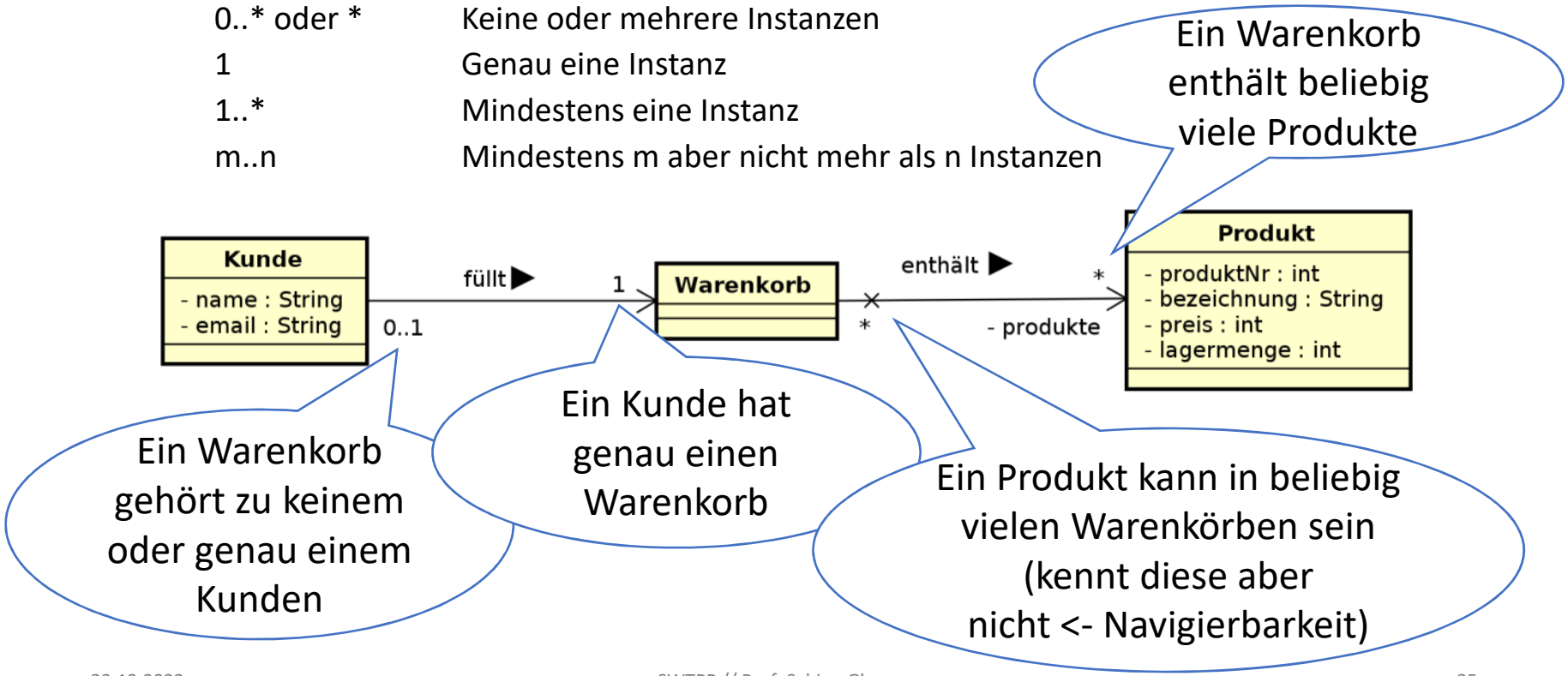


Multiplizitäten

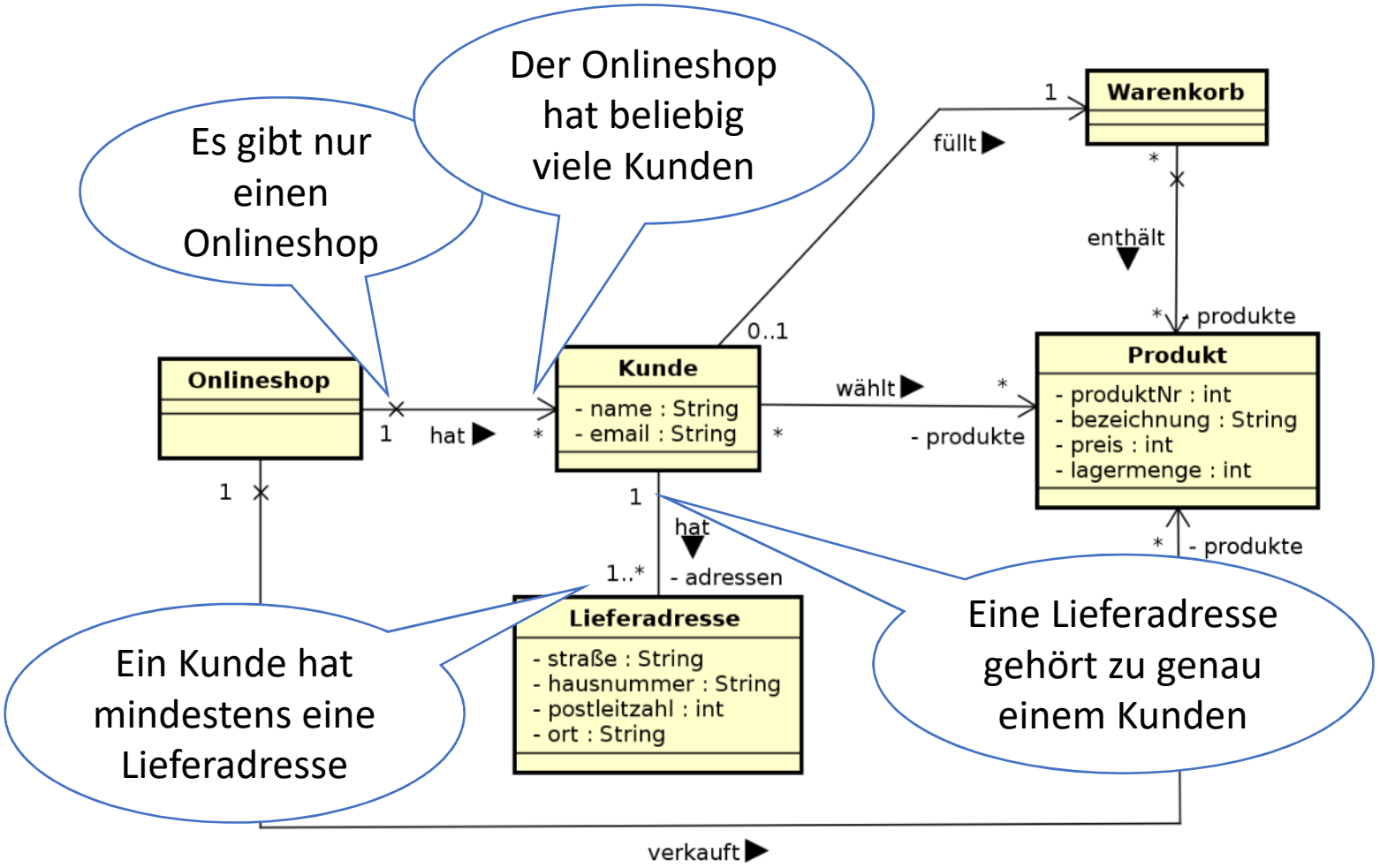
Erlauben die Anzahl der verbundenen Instanzen zu modellieren

- Typische Multiplizitäten

0..1	Keine oder genau eine Instanz
0..* oder *	Keine oder mehrere Instanzen
1	Genau eine Instanz
1..*	Mindestens eine Instanz
m..n	Mindestens m aber nicht mehr als n Instanzen



Beispiel mit mehr Details



Assoziationen vs. Attribute

Assoziationen sind eine grafische Darstellung der Attribute

- Ersetzen Attribute, wenn die Typklasse des Attributs im Modell vorhanden ist
- Sie werden auch als Attribute implementiert
- Eine Assoziation **auch noch** als Attribut zu modellieren ist falsch (doppelt)

Im Modell sind Assoziationen übersichtlicher

- Können mit Multiplizitäten genauer spezifiziert werden
- aber der Name ist optional

Vereinbarung für konsistente Modelle (diese Veranstaltung*):

1. Zwischen Klassen, die im Modell explizit modelliert werden, werden nur Assoziationen verwendet
2. Wenn zwischen zwei Klassen mehrere Assoziationen existieren, müssen Rollenbezeichner Eindeutigkeit herstellen

* prüfungsrelevant

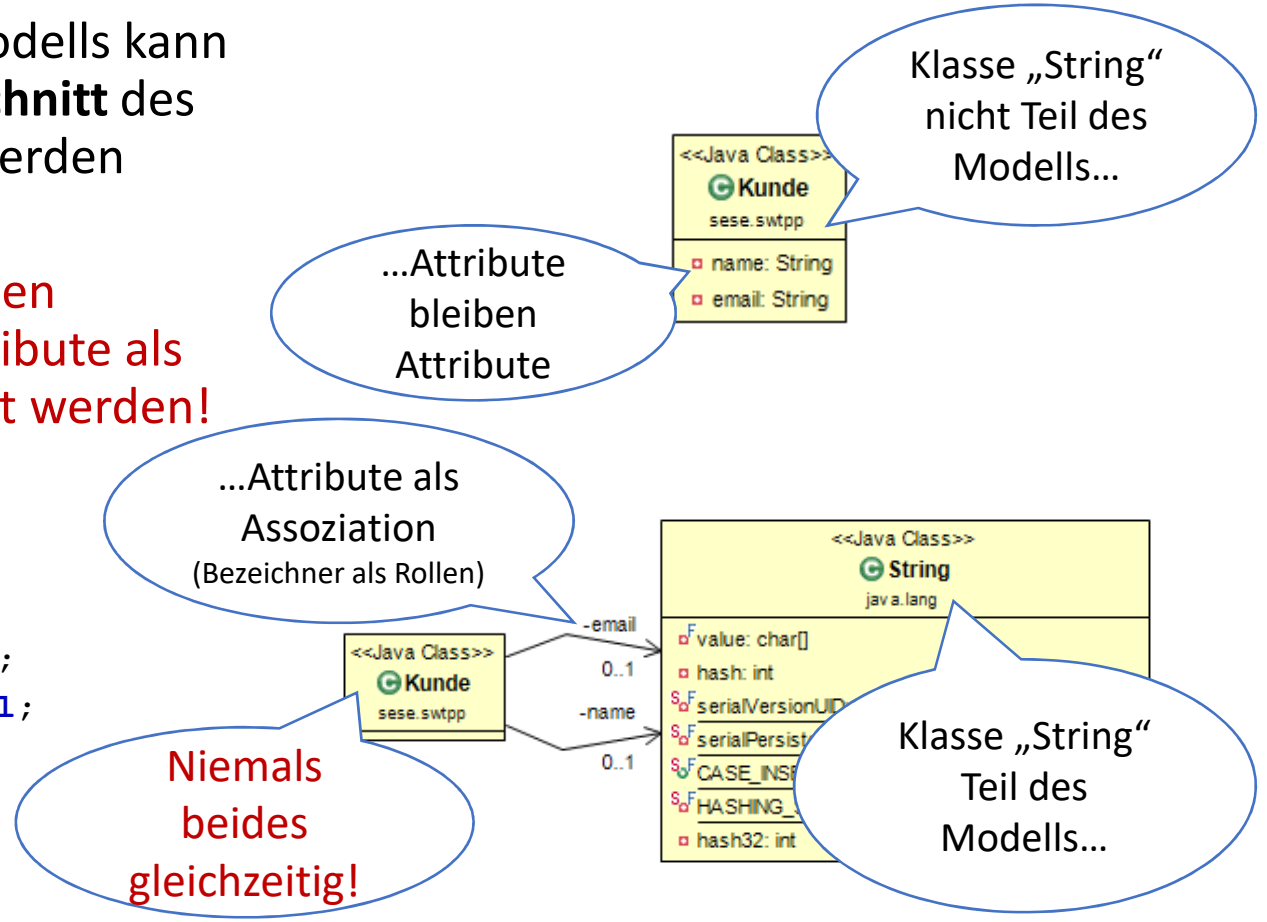
Detaillierungsgrad

Je nach Zweck des Modells kann ein spezifischer **Ausschnitt** des Systems dargestellt werden

Die Auswahl der Klassen bestimmt welche Attribute als Assoziation dargestellt werden!

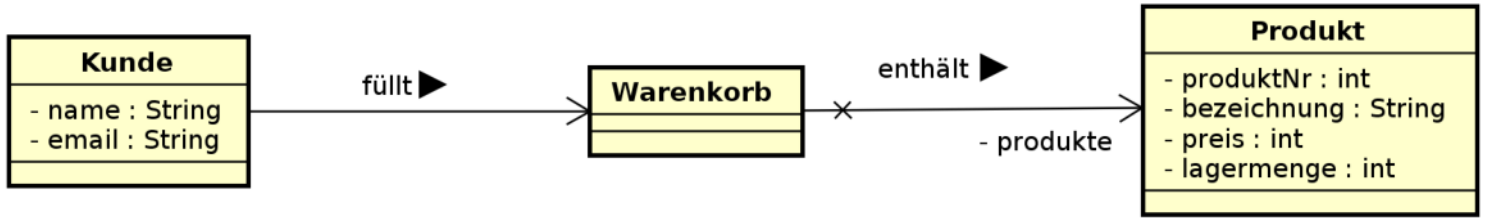
Beispiel (JAVA):

```
public class Kunde {
    private String name;
    private String email;
    /* ... */
}
```



Implementierungsdetails

Beispielhafte Implementierung der Navigierbarkeiten in JAVA




```java
public class Kunde {
    private String name;
    private String email;

    private Warenkorb warenkorb;
    /* ... */
}
```

```java
public class Warenkorb {

    private Produkt produkte[];

    private Kunde kunde;
    /* kunde ist optional
     * ... */
```

```java
public class Produkt {
    private int produktNr;
    private String bezeichnung;
    private int preis;
    private int lagermenge;

/* ... */
}
```

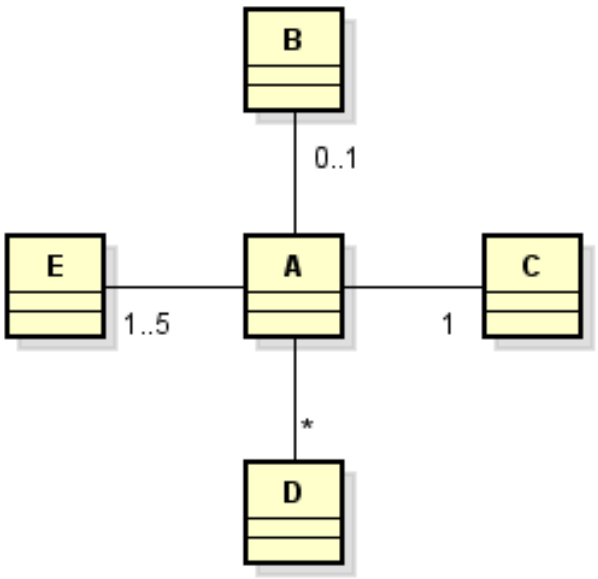
Optional, d.h. unspezifiziert. Es ist dem Entwickler überlassen

Und wie modelliere ich Datensätze?

Das Produkt kennt keine Warenkörbe

Implementierungsdetails

Beispielhafte Implementierung verschiedener Multiplizitäten in JAVA




```
public class A {
B b = null;     // null erlaubt
C c = new C(); // null nicht erlaubt
LinkedList<D> ds = new LinkedList<D>();
// Leer für ds erlaubt
LinkedList<E> es = new LinkedList<E>();

public A (E e){
  es.add(e);
   // mind. 1 Element, max. 5. Multiplizität
   // muss bei allen weiteren Änderungen
   // der Liste berücksichtigt werden
  }
}
```

Spezielle Assoziationen

In UML enthalten Beziehungen als Teil-Ganzes hervorzuheben

Aggregation (◇)

- Objekte der Klasse A sind aus anderen Objekten der Klasse B **zusammengesetzt**, bzw. die Objekte von B gehören zu A
- Multiplizitäten werden nicht eingeschränkt

  Beispiel: Eine Vorlesung besteht aus Studierenden, diese können aber mehrere VLs besuchen

Komposition (◆)

- Spezialfall der Aggregation: Die Existenz der beherbergten Klasse hängt von der Existenz der beherbergenden Klasse ab
- Kann Multiplizitäten 0..1 oder 1 bedeuten:

  Beispiel 0..1: Ein Blatt gehört zu höchstens einem Baum, kann aber auch ohne existieren

  Beispiel 1: Haus hat Räume (Haus stürzt ein → Räume sind weg)

Generalisierung/Spezialisierung

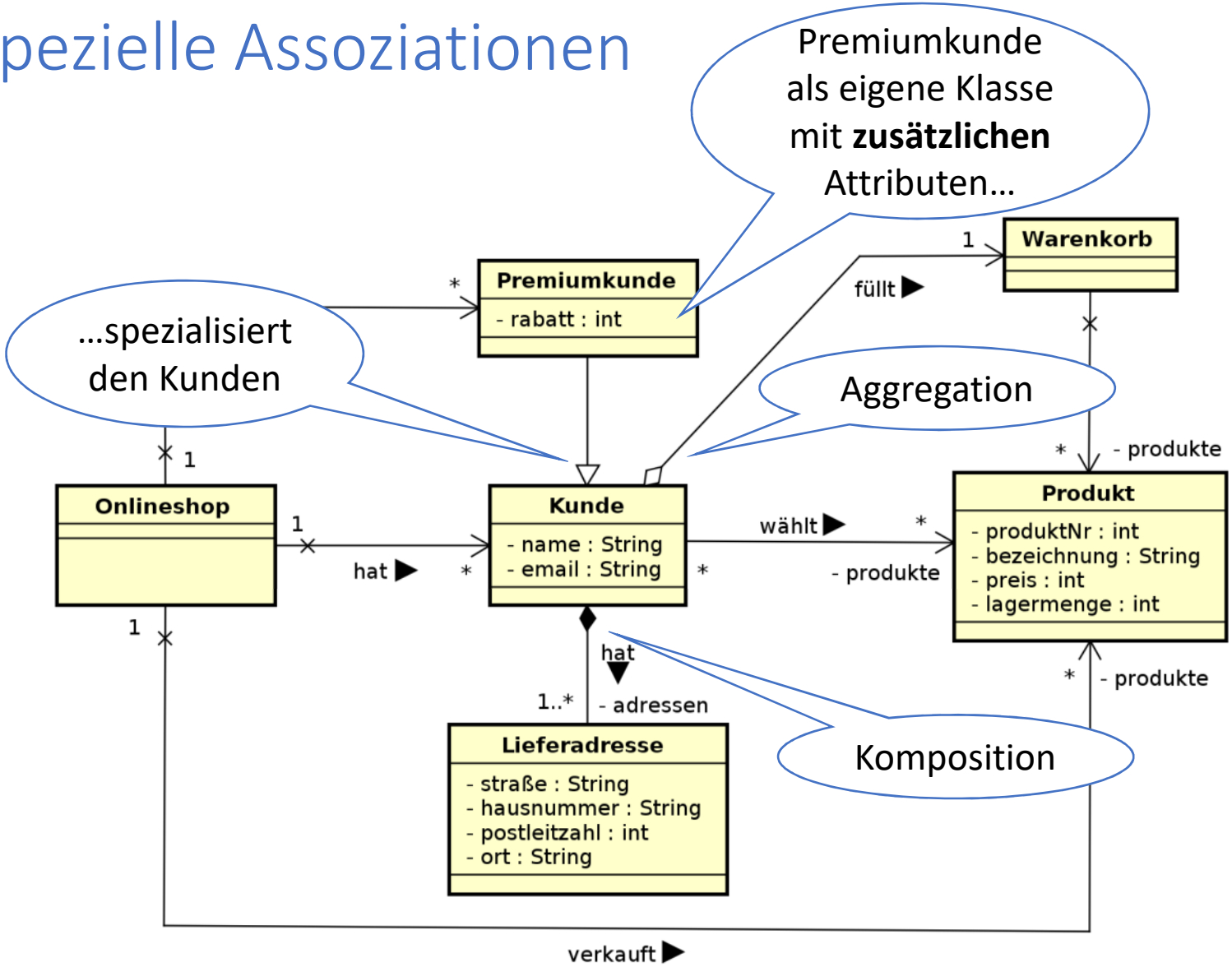
Ermöglicht die Modellierung von Vererbung in Klassendiagrammen

- Eigenschaften (Attribute/Assoziationen) der generellen Klasse (Oberklasse) werden an spezielle Klassen (Subklassen) vererbt
- Abstrakte Klassen (Oberklasse kann nicht instanziiert werden) oder Operationen (müssen durch Subklassen implementiert werden) werden kursiv dargestellt

In UML ist Mehrfachvererbung möglich

- In Java z.B. durch Interfaces umsetzbar

Spezielle Assoziationen



Stereotype

Stereotype trennen Klassen nach ihrer **Bedeutung** im System

Entity-Control-Boundary-Pattern (ECB)

Verwendet drei Stereotype zur Trennung von Daten und Funktionalität

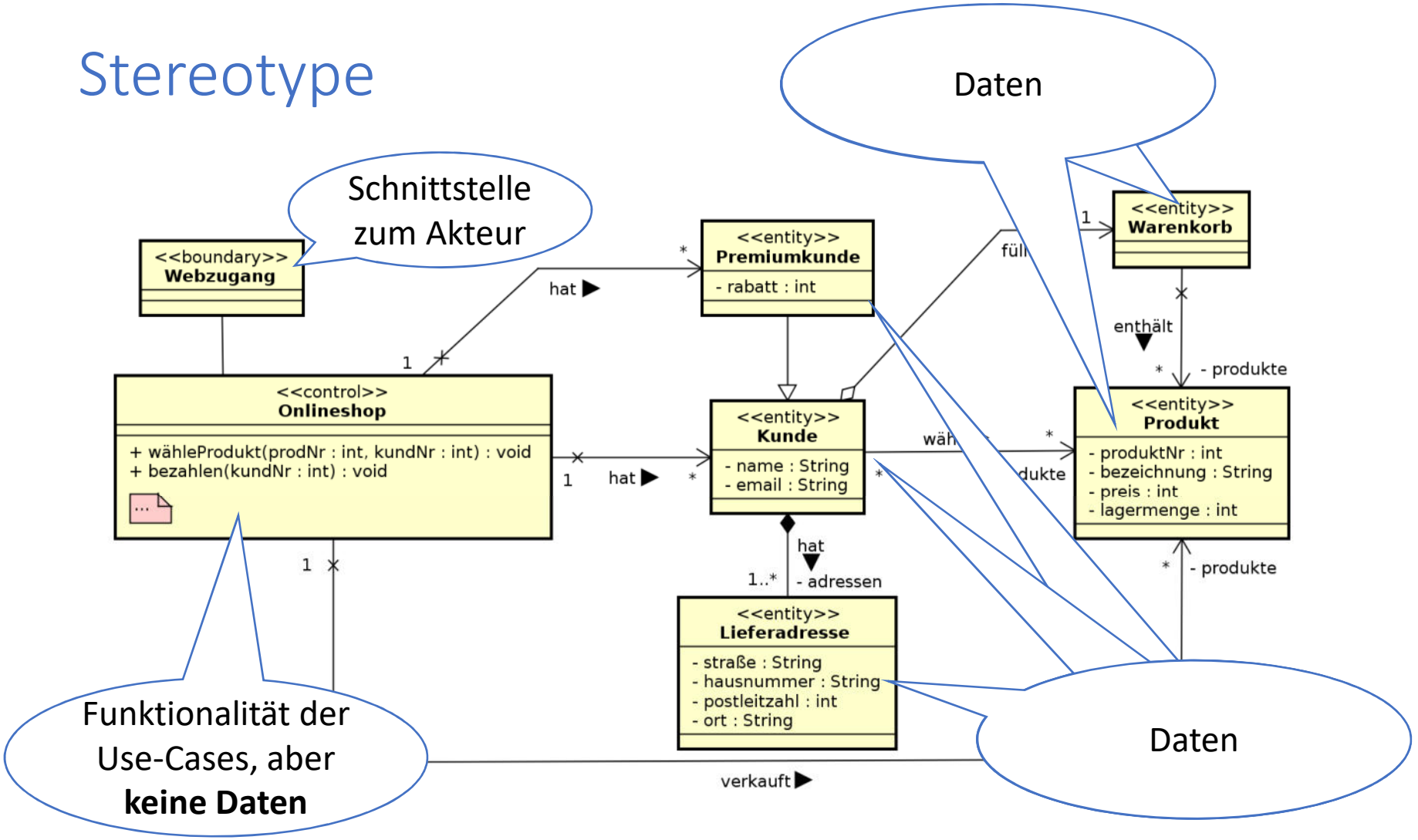
- **Entity** Klasse repräsentiert Daten im System (lokale Funktionalität erlaubt)
- **Boundary** Schnittstelle zu Akteuren außerhalb des Systems.
- **Controller** Übergeordnete Funktionalität, z.B. die Use-Cases

Controller-Klassen halten **keine persistenten Daten**, Entity-Klassen haben **keine klassenübergreifenden Operationen**

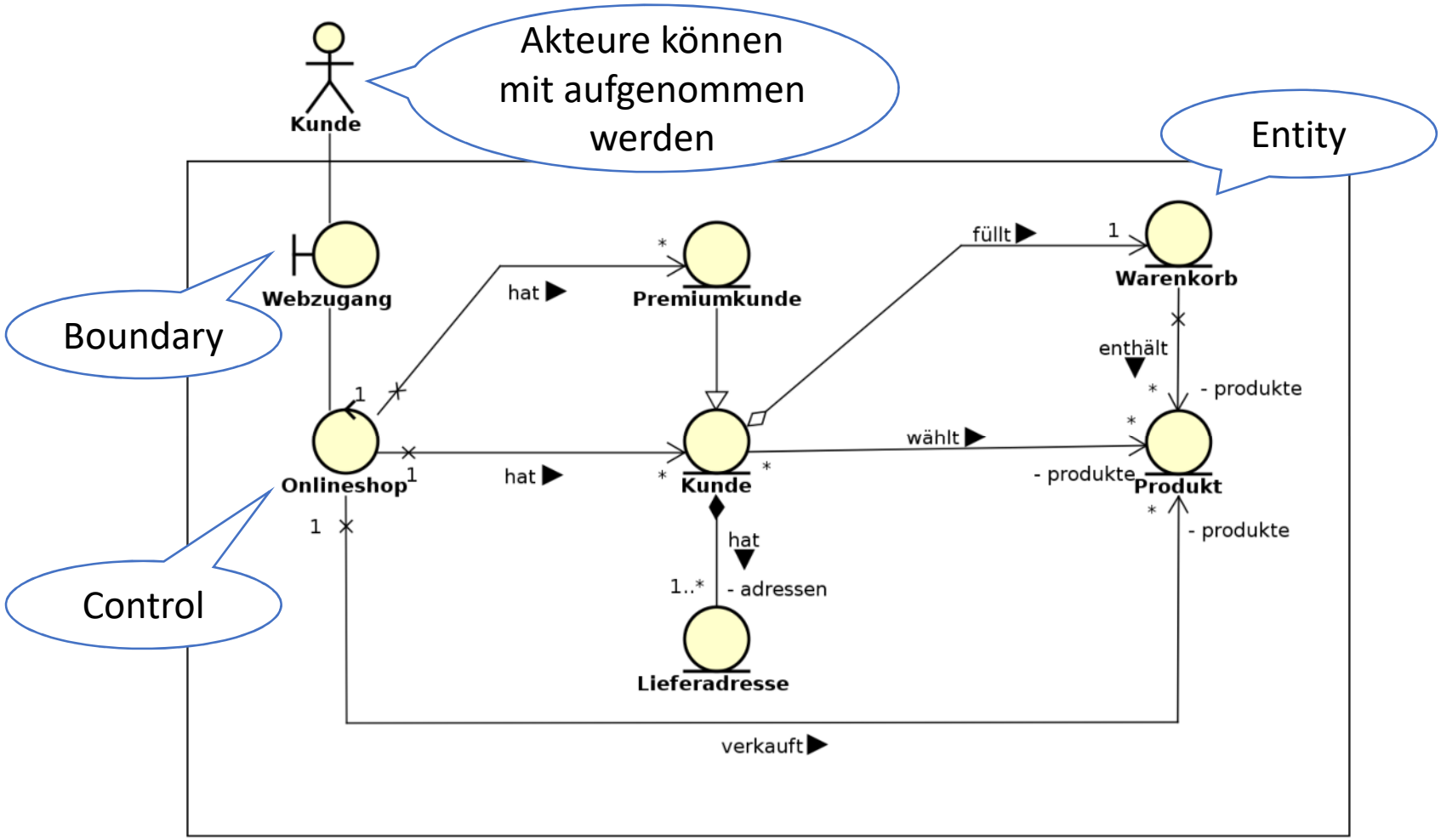
- Vereinfachte Form des Architekturmodells Model-View-Controller (MVC)

Im Allgemeinen existiert im System nur eine Instanz jedes Controllers

Stereotype

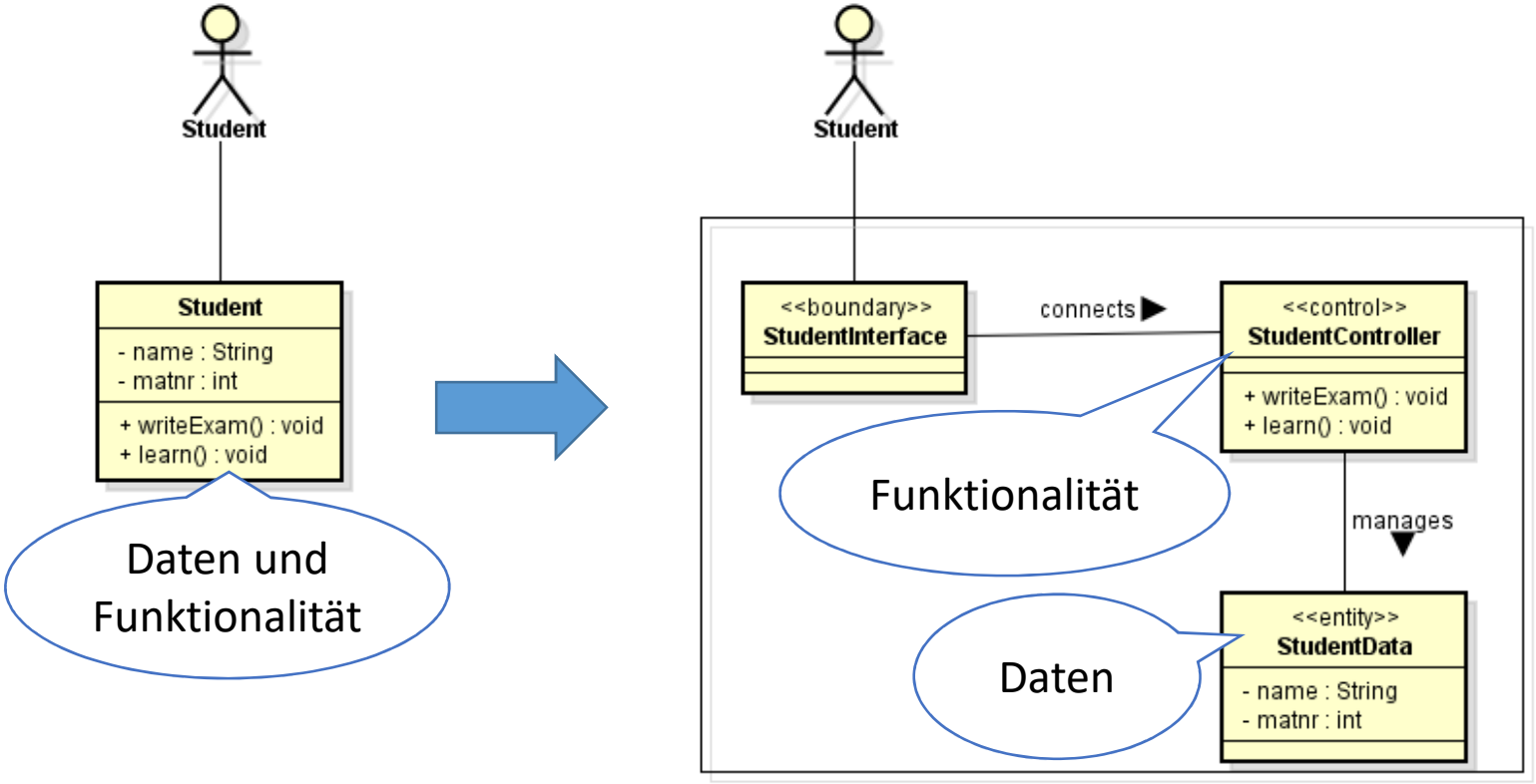

Daten
Schnittstelle zum Akteur
<<boundary>>
Webzugang
<<entity>>
Premiumkunde
- rabatt : int
<<entity>>
Warenkorb
hat
füll
enthält
- produkte
<<control>>
Onlineshop
+ wähleProdukt(prodNr : int, kundNr : int) : void
+ bezahlen(kundNr : int) : void
<<entity>>
Kunde
- name : String
- email : String
<<entity>>
Produkt
- produktNr : int
- bezeichnung : String
- preis : int
- lagermenge : int
hat
- adressen
<<entity>>
Lieferadresse
- straße : String
- hausnummer : String
- postleitzahl : int
- ort : String
verkauft
Funktionalität der Use-Cases, aber **keine Daten**
Daten

Alternative Darstellung



Stereotype

Unter Umständen müssen einzelne Klassen zerlegt werden, um eindeutige Stereotype vergeben zu können

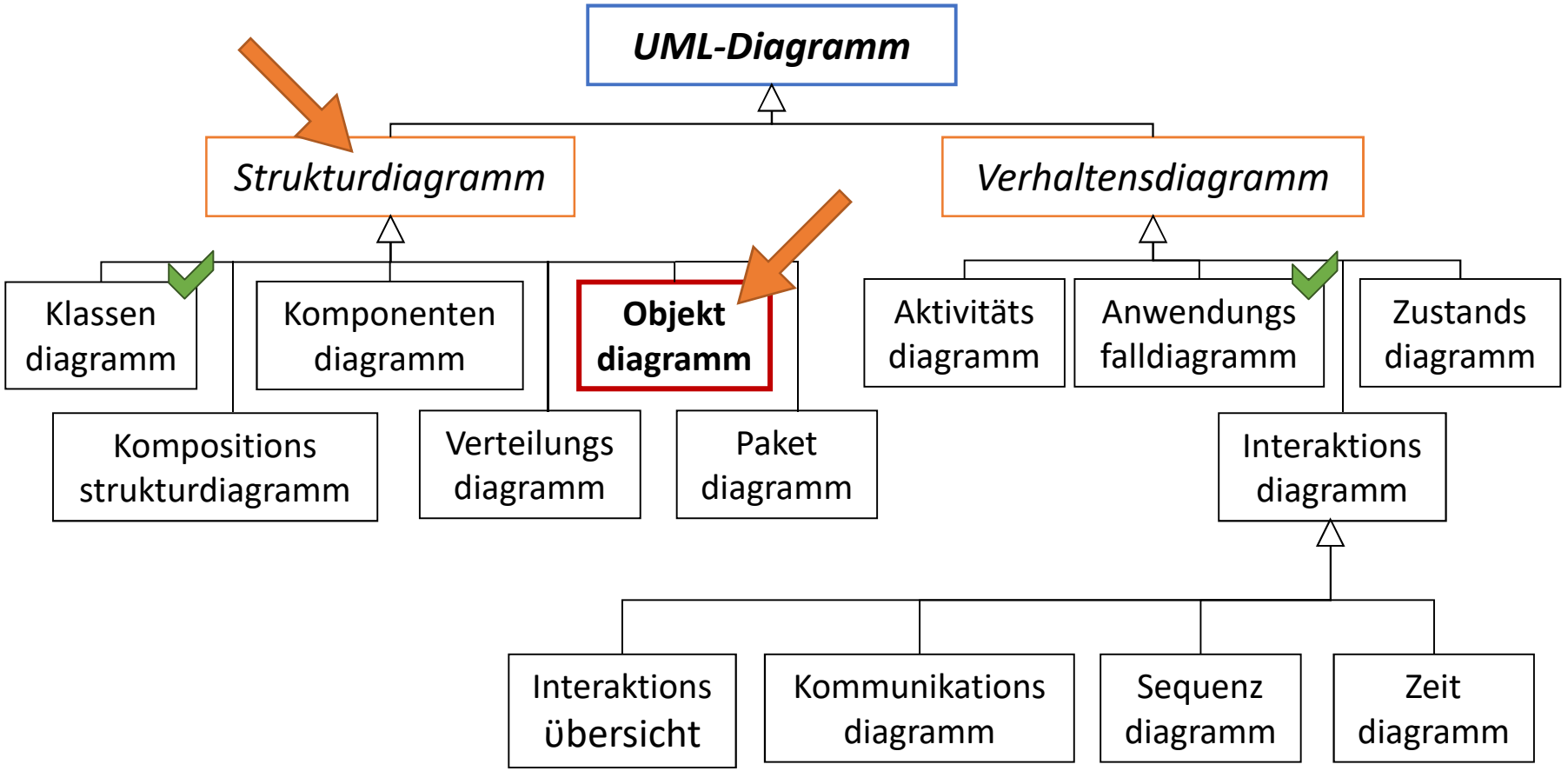


Inhalt

Analyse und Entwurf - Struktur

- Grundlagen
- Klassendiagramm
- Objektdiagramm

UML Diagrammübersicht

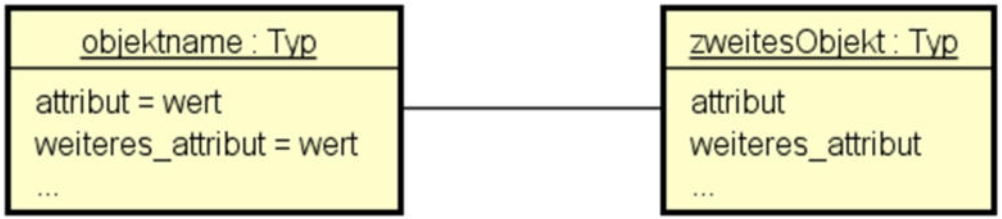


Objektdiagramme

Objekt-Diagramm ist eine Instanz des im Klassendiagramm beschriebenen Systems, d.h. **genau ein Zustand**

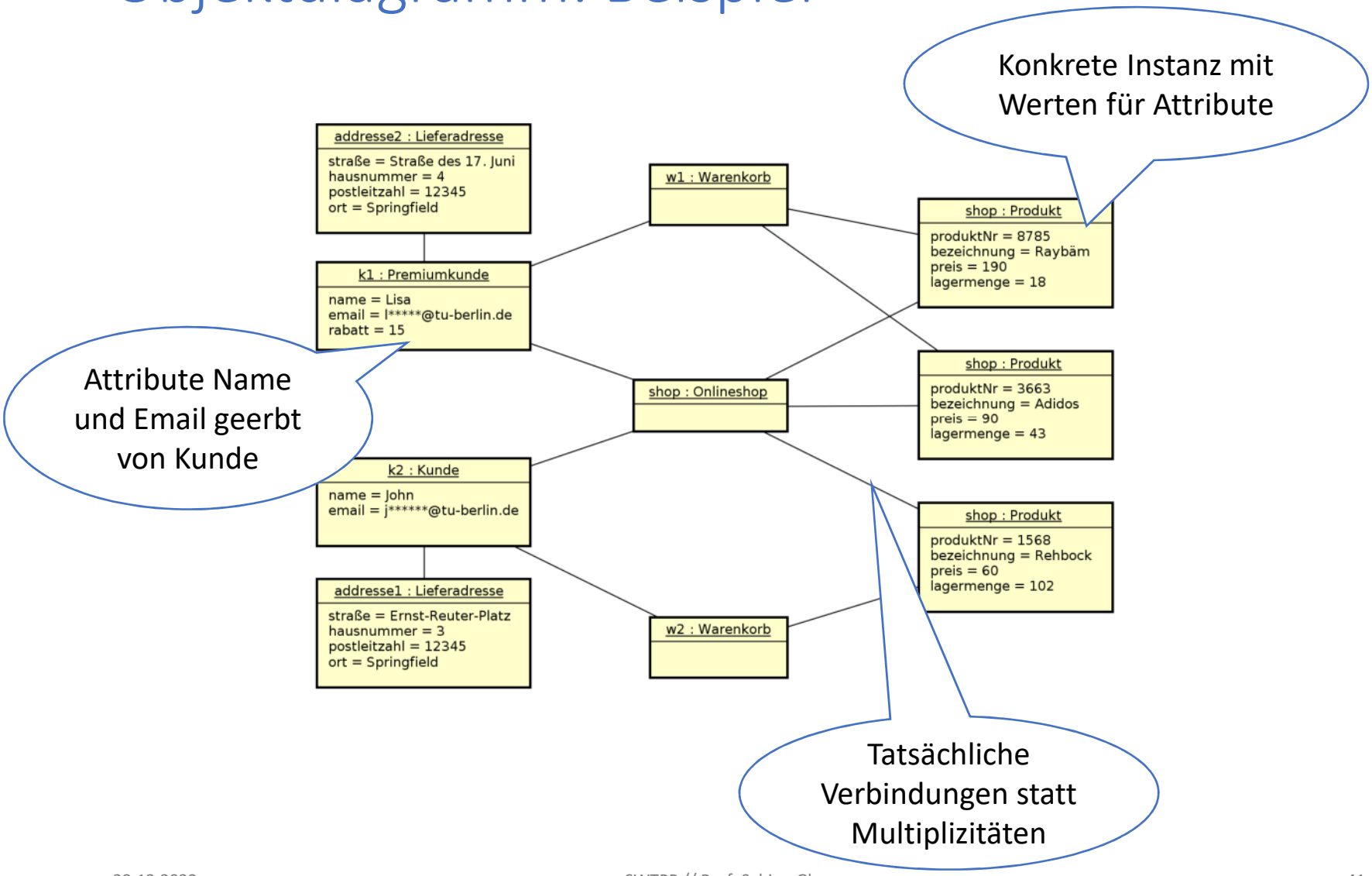
- Sinnvoll für Debugging, veranschaulichende Beispiele oder z.B. Startzustände

Objekte ähneln in der Darstellung Klassen:



Verbindungen im Objektdiagramm entsprechen instanziierten Assoziationen im Klassendiagramm

Objektdiagramm: Beispiel



Lernziele

- ❑ Was ist das Ziel der Strukturmodellierung?
- ❑ Welche Diagramme bietet UML dafür an?
- ❑ Was stellt das Klassendiagramm dar?
- ❑ Wofür kann es eingesetzt werden?
- ❑ Wie werden Klassen im Klassendiagramm dargestellt?
- ❑ Wie werden die Beziehungen zwischen Klassen modelliert?
- ❑ Was ist der Unterschied zwischen einem Attribut und einer Assoziation?
- ❑ Wann verwendet man ein Attribut, wann eine Assoziation?
- ❑ Wie lassen sich gleichartige Assoziationen eindeutig unterscheiden?
- ❑ Womit kann angegeben werden in welche Richtung eine Assoziation gilt?
- ❑ Wie werden Datensätze als Assoziation modelliert?
- ❑ Wann verzichtet man auf eine genaue Spezifizierung einer Assoziation?
- ❑ Was versteht man unter einer Aggregation/Komposition im Klassendiagramm?
- ❑ Wie modelliert man Generalisierung/Spezialisierung im Klassendiagramm?
- ❑ Was lässt sich mit Stereotypen modellieren?
- ❑ Welche Stereotype gibt es im ECB-Konzept? Was bedeuten sie?
- ❑ Wie unterscheiden sich Klassen- und Objektdiagramm?
- ❑ Was stellt das Objektdiagramm dar?
- ❑ Kann das Objektdiagramm zum Systementwurf verwendet werden?
- ❑ Wie werden Objekte im Objektdiagramm dargestellt?