Technische Universität
Kaiserslautern

o1o1seda o1o1oo

software engineering dependability

Software Entwicklung 2

UML im Entwurf

Inhalt

- UML im Entwurf
- Objekt/Klasse
- Attribut
- Operation
- Assoziation
- Polymorphismus
- Vererbung
- Pakete
- Szenario
- Zustandsautomat

Lernziele

- Die Unterschiede zwischen der UML im Entwurf und in der Analyse kennen und erläutern können
- Ein gegebenes UML-Modell erklären können
- Eine verbale Entwurfsbeschreibung in eine UML-Notation umsetzen können

UML im Entwurf

- Zur Historie
  - Grady Booch
    * 27.2.1955 in Texas
    Chief Scientist
    Rational Software Corporation
  - Pionier auf dem Gebiet des
    modularen und objektorientierten Softwareentwurfs
    - 1983: Buch *Software Engineering with Ada*
    - 1987: Buch *Software Components with Ada*
    - 1991/94: Buch *Object Oriented Design with Applications*
  - Pionier wiederverwendbarer Bibliotheken
    - 1987: Komponentenbibliothek in Ada
    - 1991: Klassenbibliothek in C++

UML im Entwurf

- Grundlage für den objektorientierten Entwurf
  - In der Regel das OOA-Modell
  - Das entstehende OOD-Modell wiederum bildet die Grundlage für die Implementierung
  - Die Implementierung erfolgt in einer oder mehreren konkreten Programmiersprachen
- Bezeichner-Syntax
  - Alle Namen des OOD-Modells müssen der Syntax der Ziel-Programmiersprache entsprechen
  - Bezeichnungen aus dem OOA-Modell, die dieser Syntax nicht ent-sprechen, müssen daher manuell oder automatisch umgewandelt werden

UML im Entwurf

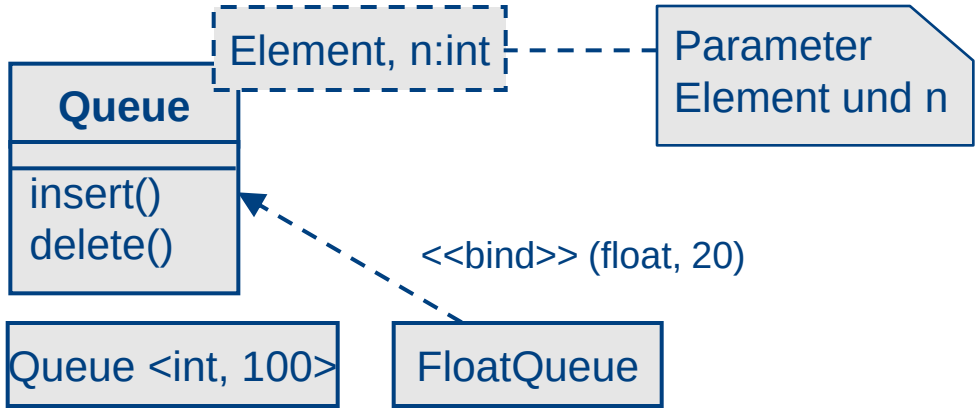
- Deutsch vs. Englisch
  - In der Systemanalyse
    - In der Regel Deutsch und die jeweilige Fachterminologie
  - In der Entwurfsphase und in der Implementierung
    - Üblich Englisch
      - Ermöglicht kürzere Bezeichnungen
      - Wird in Klassenbibliotheken durchgängig verwendet
    - Neue Klassen, die in der Entwurfsphase hinzukommen, u. U. mit englischen Bezeichnern
    - Die Verwendung von deutschen Bezeichnern hat jedoch den Vorteil, dass man leicht zwischen selbstgeschriebenen und benutzten Klassen unterscheiden kann.

Objekt/Klasse

- Stereotyp
  - Zugehörigkeit einer Klasse zu einer bestimmten Entwurfskomponente, z. B. zur GUI-Schicht, kann gezeigt werden
  - Beispiel
    - Alle Klassen, die zur Datenverwaltung gehören, werden mit «DB» (*data base*) und alle Klassen zur Realisierung der Benutzungsoberfläche mit «GUI» (*graphical user interface*) gekennzeichnet
- Generische Klasse
  - Beschreibt eine Familie von Klassen mit einem oder mehreren formalen Parametern
  - Parameter einer generischen Klasse sind
    - Typ-Parameter: Bezeichner, der innerhalb der Klasse wie ein gewöhnlicher Typ verwendet werden kann
    - Konstanten-Parameter: Bezeichner, der innerhalb der Klasse wie eine Konstante des angegebenen Typs verwendet werden kann

Objekt/Klasse

- Formale Parameter müssen an aktuelle Parameter gebunden werden
  - Für Typ-Parameter: ein konkreter Typ
  - Für Konstanten-Parameter: ein fester Wert
  - Durch die Bindung entsteht eine neue Klasse
  - Beispiel für Meta-Klassen, also Klassen, deren Exemplare (»Objekte«) wieder Klassen sind
- Beispiel: Generische Klasse Queue

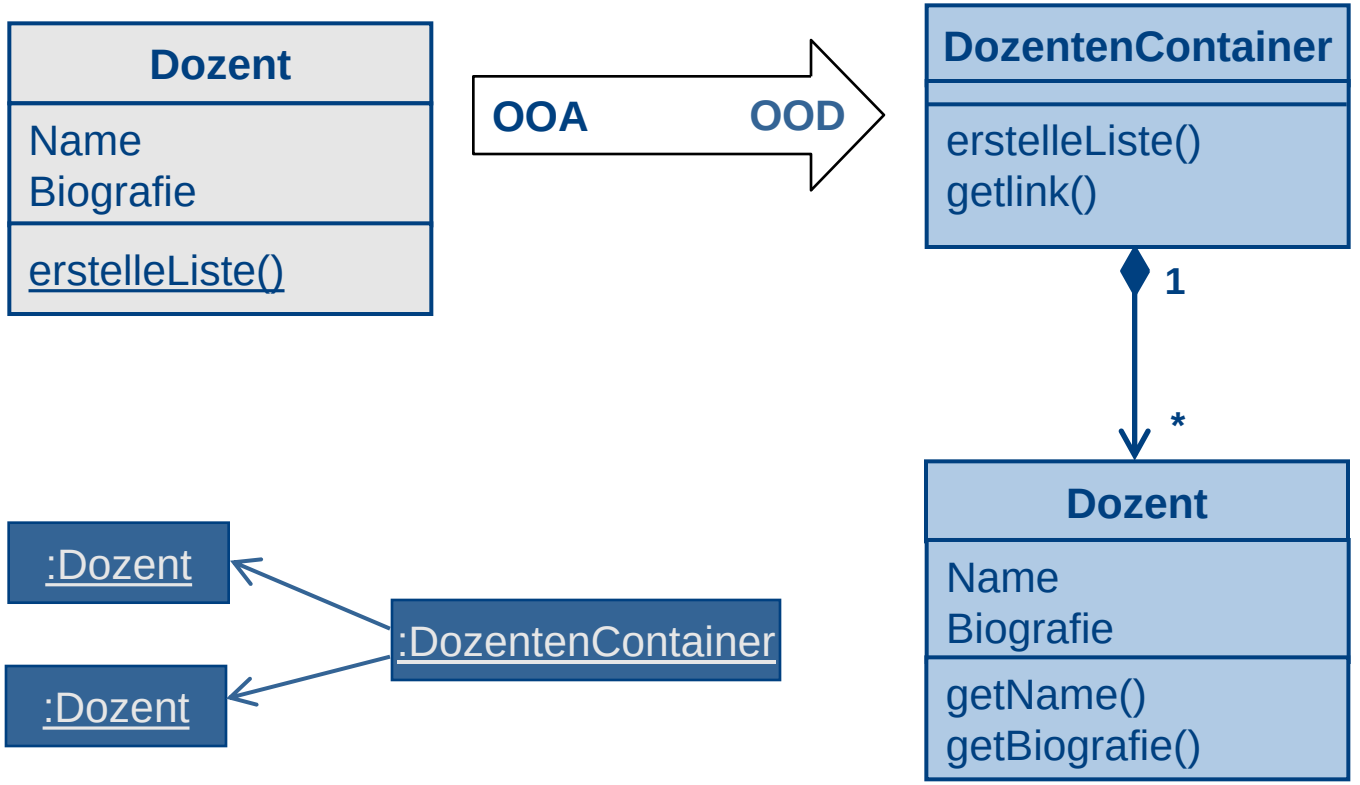


Objekt/Klasse

- Container-Klasse
  - Verwaltet Menge von Objekten einer Klasse
  - Stellt Operationen bereit, um auf die verwalteten Objekte zuzugreifen
  - Ein Objekt der Container-Klasse wird als Container bezeichnet
  - Typische Container
    - Felder (*arrays*) und Mengen (*sets*)
  - Namenskonvention
    - Plural des Klassennamens, u. U. gefolgt von dem Namen `Container`
    - Fachkonzept-Klasse `Kunde`
    - Zugehörige Container-Klasse `Kunden` oder `KundenContainer`

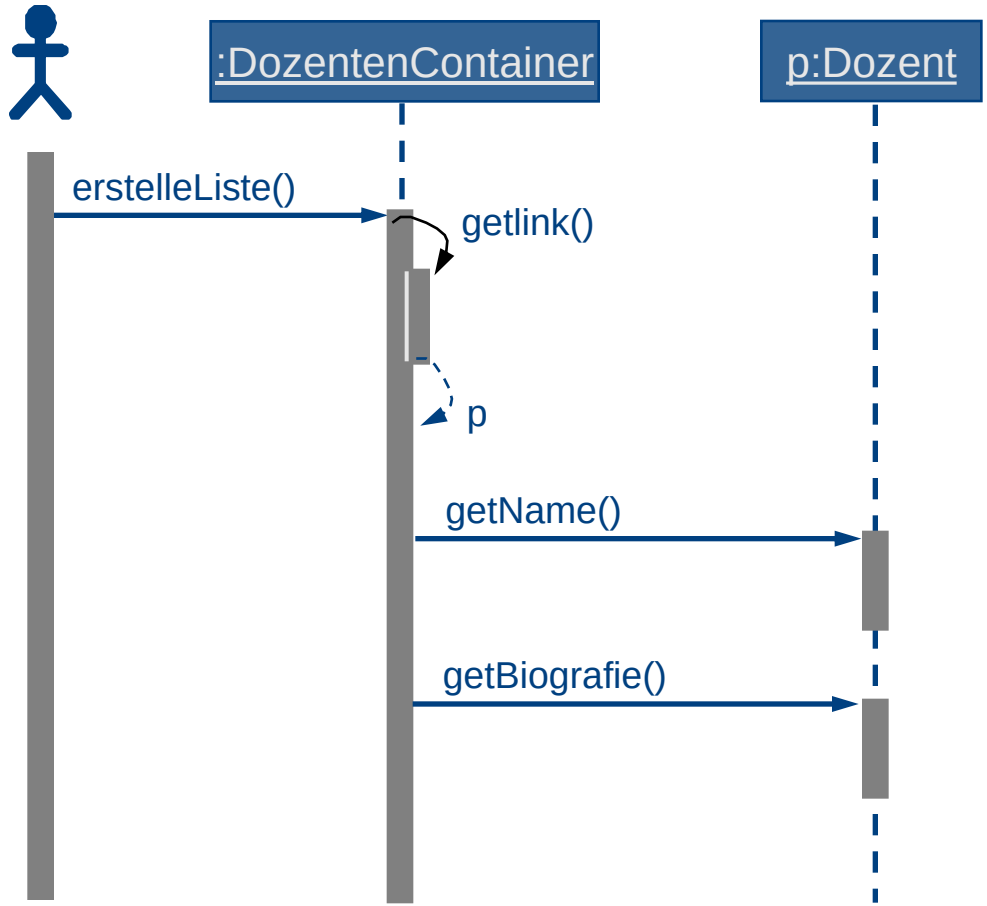
Objekt/Klasse

- Container für die Verwaltung von Dozenten



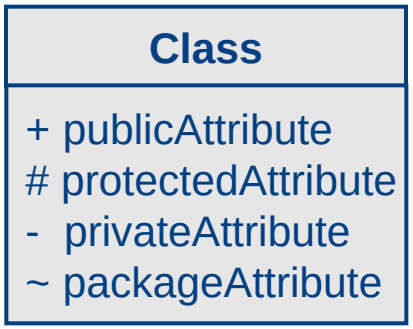
Objekt/Klasse

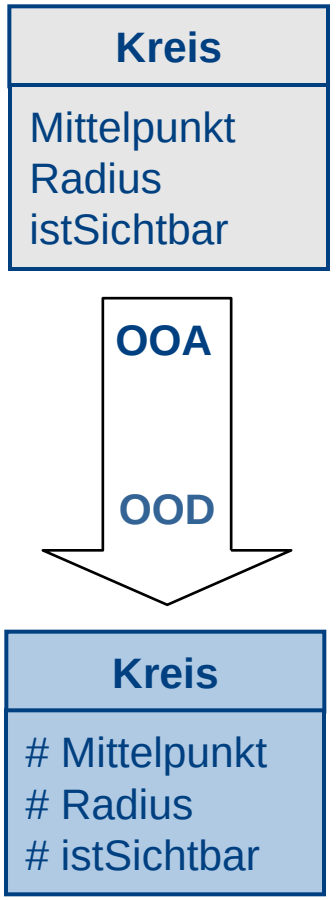
- Sequenzdiagramm für die Verwaltung von `Dozenten` im Container



Attribut

- Sichtbarkeit (visibility) in OOD
  - public: sichtbar für alle anderen Klassen
  - protected: sichtbar innerhalb der Klasse und in deren Unterklassen
  - private: sichtbar nur innerhalb der Klasse
  - package: sichtbar nur innerhalb des jeweiligen Paketes (nur in Java)





Attribut

- Attribute sollen prinzipiell als protected oder private vereinbart werden
  - Jedes Objekt sieht alle protected-Attribute seiner Oberklassen und kann direkt darauf zugreifen
  - Sind Attribute private, dann sehen Objekte die Attribute ihrer Oberklassen nicht, sondern dürfen nur über entsprechende Operationen zugreifen
  - Vorteil
    - Veränderungen der Attribute wirken sich nicht auf die Unter-klassen aus
    - Realisierung der Unterklasse unabhängig von der Darstellung der Attribute der Oberklasse
  - Nachteil
    - Zusätzliche Lese-/Schreiboperationen

Attribut

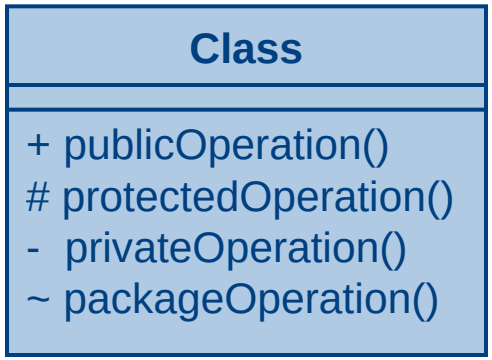
- Klassenattribut
  - Kann auf zwei Arten im OOD-Modell realisiert werden
    - Als Klassenattribut
    - Als Objektattribut einer separaten Klasse
      - Diese Klasse besitzt dann nur ein einziges Objekt mit dem Wert des Klassenattributs
      - Alle Objekte, für die das Attribut gelten soll, müssen dasselbe Objekt der separaten Klasse referenzieren
- Abgeleitetes Attribut
  - Kann entweder als Attribut – mit entsprechender Konsistenzprüfung – oder durch eine Operation realisiert werden, die stets den aktuellen Wert ermittelt

Attribut

- Verkapselung vs. Geheimnisprinzip
  - Geheimnisprinzip (information hiding)
    - Zustand eines Objekts und Implementierung der Operationen außerhalb der Klasse nicht sichtbar
  - Verkapselung (encapsulation)
    - Zusammengehörende Attribute und Operationen, in einer Einheit – der Klasse – verkapselt
    - Entgegen dem Geheimnisprinzip können die Attribute und die Realisierung der Operationen durchaus nach außen sichtbar sein

Operation

- Sichtbarkeiten (analog: Attribute)
  - Private Operation (private)
    - Kann nur von Operationen derselben Klasse aus aufgerufen werden
    - Für alle anderen Klassen bzw. deren Objekte unsichtbar
  - Geschützte Operation (protected)
    - Kann von Operationen der eigenen Klasse und ihren Unterklassen aus aufgerufen werden
  - Öffentliche Operation (public)
    - Kann von Operationen aller Klassen bzw. deren Objekten aufgerufen werden

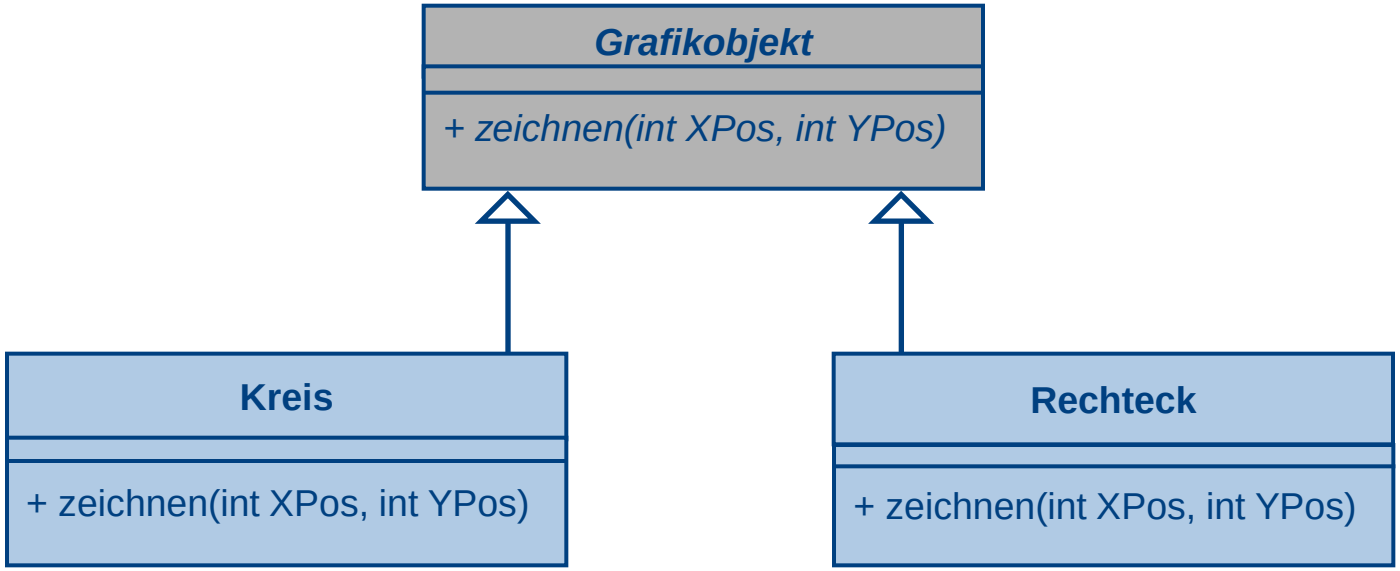


Operation

- Signatur
  - Für jede Operation vollständige Signatur angeben
  - Besteht aus Namen der Operation, den Namen und Typen aller Parameter, Ergebnistyp der Operation
  - Menge aller Signaturen: Schnittstelle der Klasse
- Beschreibung einer Operation
  - Bei Bedarf kann eine Beschreibung mittels Vor- und Nachbedingungen erfolgen
  - Vorbedingung beschreibt, welche Bedingungen vor dem Aktivieren einer Operation erfüllt sein müssen
  - Nachbedingung beschreibt die Änderung, die durch die Operation bewirkt wird
  - Für die Implementierung einer Operation wird auch der Begriff »Methode« verwendet.

Operation

- Abstrakte Operation
  - Besteht nur aus der Signatur
  - Besitzt keine Implementierung
  - Werden verwendet, um für Unterklassen eine gemeinsame Schnittstelle zu definieren

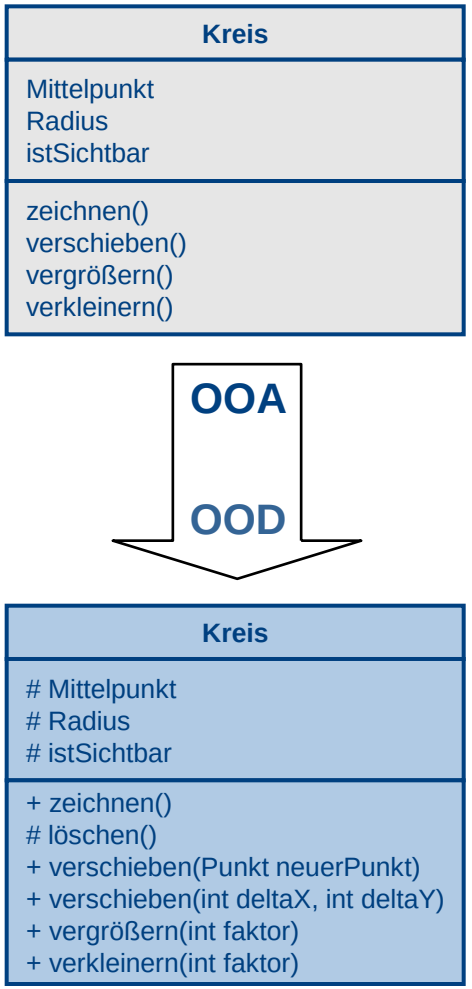


Operation

- Syntax einer Operation in UML
  - `Sichtbarkeit Operation (Parameterliste): Ergebnistyp {Merkmalsliste}`
  - Parameterliste enthält formale Parameter, die jeweils durch Kommata getrennt sind
  - Für jeden Parameter der Parameterliste gilt
    - `[in | out | inout] Name:Typ = Anfangswert`
  - »Name« steht für den formalen Parameter

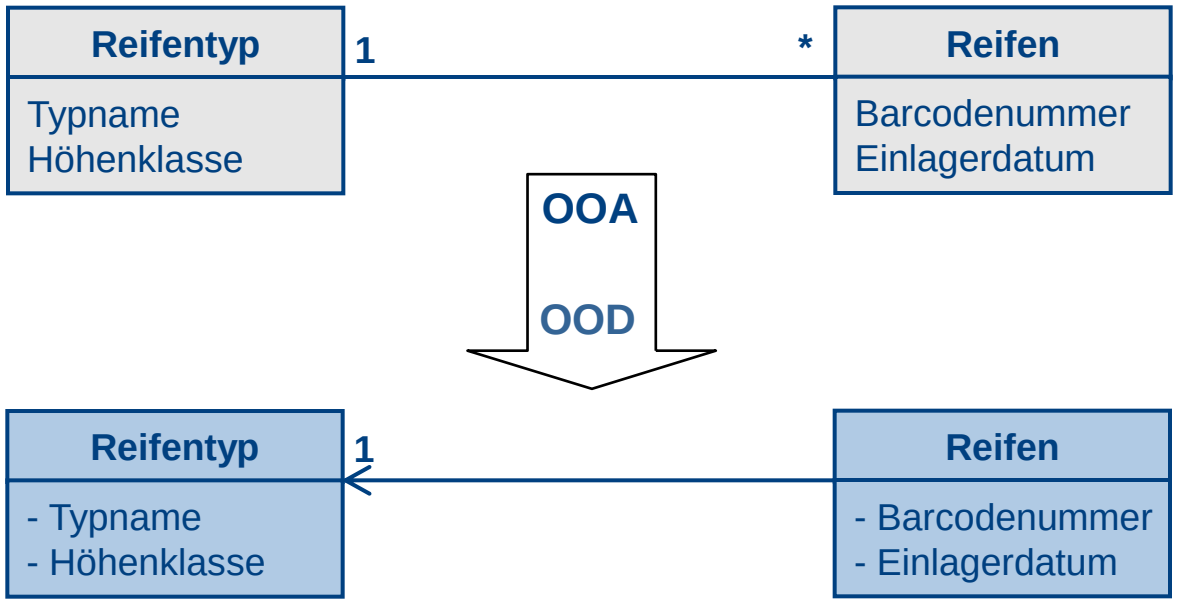
Operation

- Beispiel OOA → OOD
- Überladen
  - In der Analyse wird gefordert, dass der Operationsname innerhalb einer Klasse eindeutig ist
  - In Entwurf und Programmierung darf der gleiche Operationsname innerhalb einer Klasse mehrfach verwendet werden
    - Die sich entsprechenden Operationen müssen sich in ihrer Parameterliste unterscheiden



Assoziation

- Navigation
  - Im Entwurf wird festgelegt, ob Assoziationen uni- oder bidirektional implementiert werden
  - Man spricht von der Navigation der Assoziation

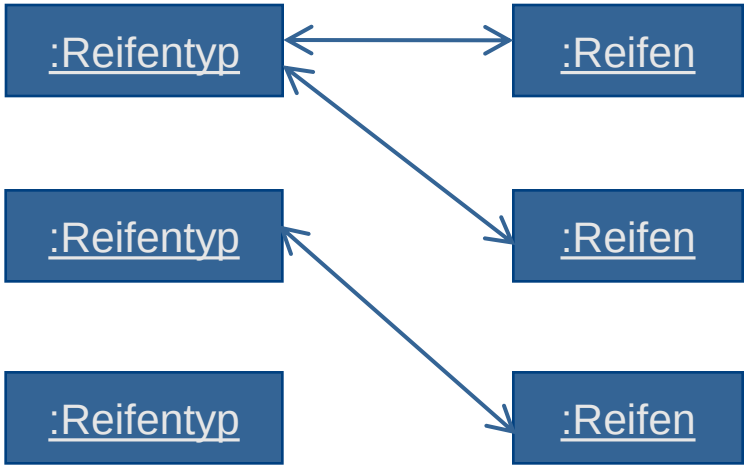


Assoziation

- Realisierung mittels Zeigern
  - Jede Richtung einer Assoziation kann mittels Zeigern zwischen Objekten realisiert werden
  - Dann kennt jedes Objekt seine assoziierten Objekte
  - Durch die Operationen muss sichergestellt werden, dass alle Verbindungen konsistent auf- und abgebaut werden
  - Eine Kardinalität von 0..1 oder 1 wird dabei durch einen einzelnen Zeiger realisiert
  - Liegt eine Kardinalität größer 1 vor, dann muss eine Menge von Zeigern gespeichert werden
    - Wenn keine Ordnung der Assoziation definiert ist, dann können Container-Klassen wie Set, Bag etc. verwendet werden

Assoziation

- Beispiel
  - Realisierung einer bidirektionalen Assoziation mittels Zeigern

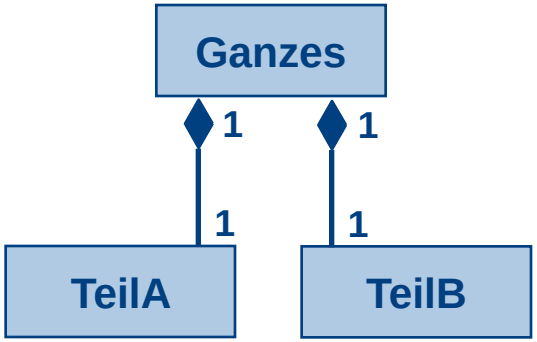


Assoziation

- Aggregation
  - Realisierung wie die »normale« Assoziation
  - Ein Ganzes muss jedoch stets seine Teile kennen
    - Navigation vom Ganzen zu den Teilen möglich
- Komposition
  - Navigation vom Ganzen zu den Teilen möglich
  - Operationen, die das Ganze betreffen, müssen sich auch auf seine Teile auswirken
  - Das Ganze und die Teile sind als Einheit zu betrachten
    - Z. B. das Sperren/Entsperren und die Autorisierung
    - Der Zugriff im Dialog und das Erzeugen der Teile erfolgen immer über das Aggregatobjekt

Assoziation

- Realisierung Komposition
  - Kann auch über echtes physisches Enthaltensein (by value) realisiert werden
  - by value-Realisierung besitzt den Vorteil, dass die Teile automatisch mit dem Ganzen erzeugt bzw. gelöscht werden
  - Auch das Kopieren des Aggregatobjekts bezieht sich immer automatisch auf seine Teile
  - by value & by reference-Realisierung in C++




```
class Ganzes
{  TeilA   einTeilA;
   TeilB*  einTeilB;

public:
    Ganzes()
    { einTeilB = new TeilB;}
    ~Ganzes()
    {delete einTeilB;}
};
```

Assoziation

- ordered & sorted
  - `{ordered}`
    - Assoziationen, deren Objektverbindungen *(links)* geordnet sind
    - Zur Realisierung eine Container-Klasse verwenden, die eine Ordnung ihrer Elemente ermöglicht (z. B. *Array, Vector)*
  - `{sorted}`
    - Ordnungskriterium sind die Elementwerte
    - Z. B. können alle Kundenobjekte nach der Kundennummer sortiert sein
    - Genauere Informationen durch eine separate Restriktion formulieren

Assoziation

- Merkmale
  - Können auf jeder Seite einer Assoziation angegeben werden
  - `{frozen}`
    - Objektverbindungen können weder hinzugefügt noch gelöscht oder geändert werden, nachdem ein Objekt der Klasse B erzeugt und initialisiert wurde

Assoziation

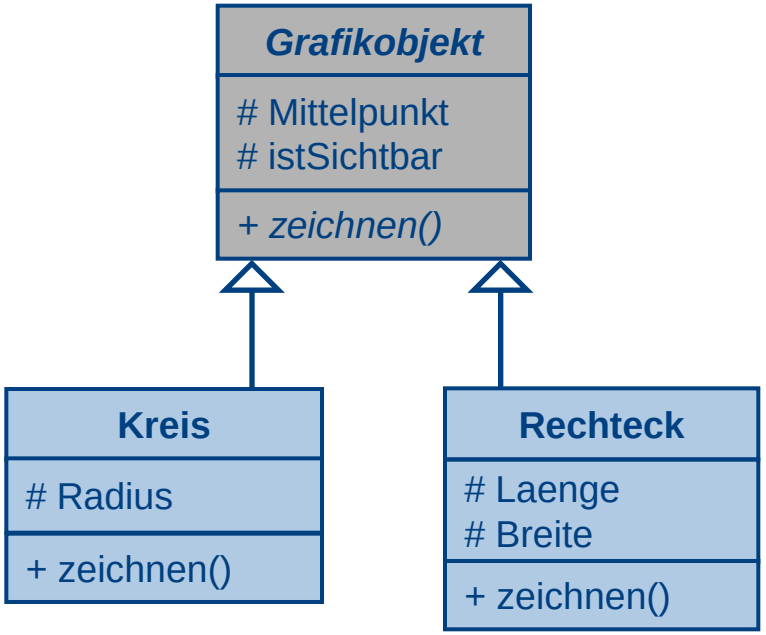
- `{addOnly}`
  - Dieses Merkmal gibt an, dass für ein Objekt der Klasse D – bei einer many-Kardinalität – zwar weitere Verbindungen eingetragen, vorhandene Verbindungen aber nicht entfernt werden dürfen
- Wird kein Merkmal angegeben, dann können Objektverbindungen beliebig hinzugefügt und entfernt werden

Polymorphismus

- Ermöglicht es, den gleichen Namen für gleichartige Operationen zu verwenden, die auf Objekten verschiedener Klassen auszuführen sind
- Der Sender muss nur wissen, dass ein Empfängerobjekt das gewünschte Verhalten besitzt
- Er muss nicht wissen, zu welcher Klasse das Objekt gehört
- Dieser Mechanismus ermöglicht es, flexible und leicht änderbare Softwaresysteme zu entwickeln

Polymorphismus

- Beispiel
  - Es wird ein Zeiger `pGrafik` deklariert
    - `Grafikobjekt pGrafik;`
    - Operationsaufruf `pGrafik.zeichnen()` kann unterschiedliche Wirkungsweisen besitzen
    - Gilt `pGrafik = new Kreis`, dann wird die Operation `Kreis.zeichnen()` aktiviert
    - Gilt `pGrafik = new Rechteck`, dann wird `Rechteck.zeichnen()` ausgeführt



Polymorphismus

- Spätes Binden/dynamisches Binden
  - Erst zur Laufzeit wird bestimmt, ob der Zeiger `pGrafik` auf ein Kreis- oder Rechteck-Objekt zeigt
  - Polymorphismus und spätes Binden *(late binding)* sind untrennbar verbunden
  - Ist zur Übersetzungszeit die Klasse des Objekts nicht bekannt, dann kann noch nicht bestimmt werden, welche Operation ausgeführt wird
  - Spätes Binden: Operation wird erst zur Ausführungszeit an ein bestimmtes Objekt gebunden
    - Polymorphe Operation
    - Java: Alle Operationen sind polymorph
    - C++: Operationen müssen explizit polymorph deklariert werden (Ableitung aus einer abstrakten Operation der Elternklasse, Schlüsselwort: *virtual*)

Polymorphismus

- Erspart umfangreiche *switch*-Anweisungen
- Beispiel
  - Bei herkömmlicher prozeduraler Programmierung (z. B. in C) wäre für obiges Beispiel folgende Konstruktion notwendig

```
enum Grafikart {istRechteck, istKreis};
void zeichnenGrafik (Grafik Grafikdaten)
switch (Grafikdaten.Art)
{ case istRechteck: zeichnenRechteck(Grafikdaten); break;
  case istKreis:       zeichnenKreis(Grafikdaten); break;
}
```

Polymorphismus

- Beispiel
  - In Java sieht das obige Beispiel wie folgt aus

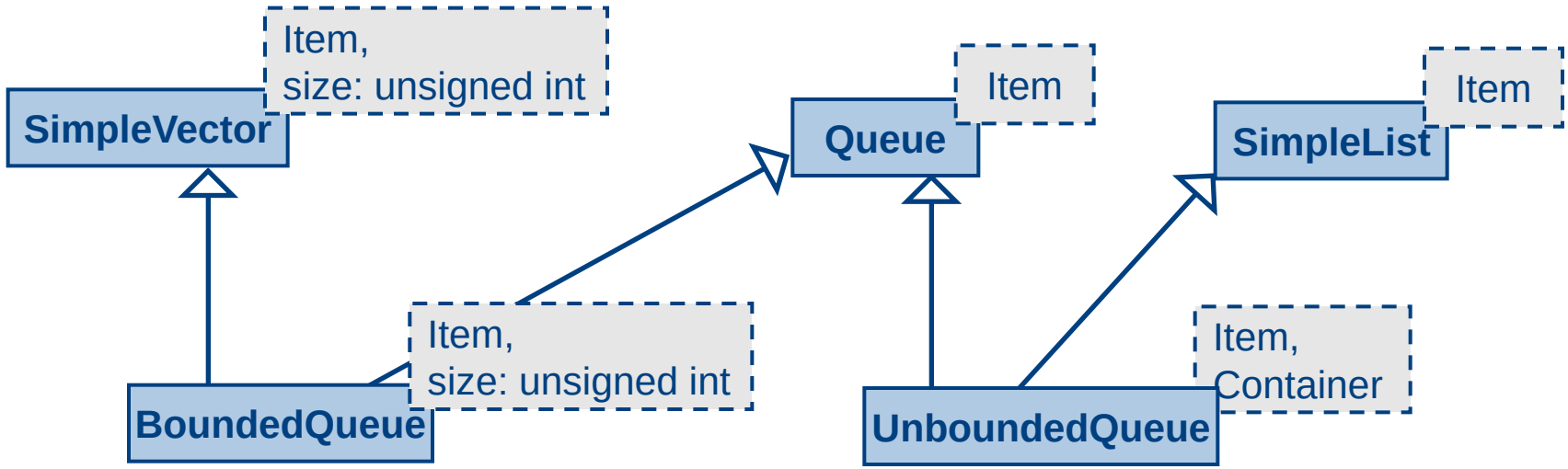
```
abstract class Grafikobjekt
{ abstract public void zeichnen(); ...}
class Kreis extends Grafikobjekt
{ public void zeichnen() {...} ...}
class Rechteck extends Grafikobjekt
{ public void zeichnen() {...} ...}

Grafikobjekt eineGrafik;
eineGrafik = new Kreis();
eineGrafik.zeichnen();              //zeichnet einen Kreis

eineGrafik = new Rechteck();
eineGrafik.zeichnen();              //zeichnet ein Rechteck.
```

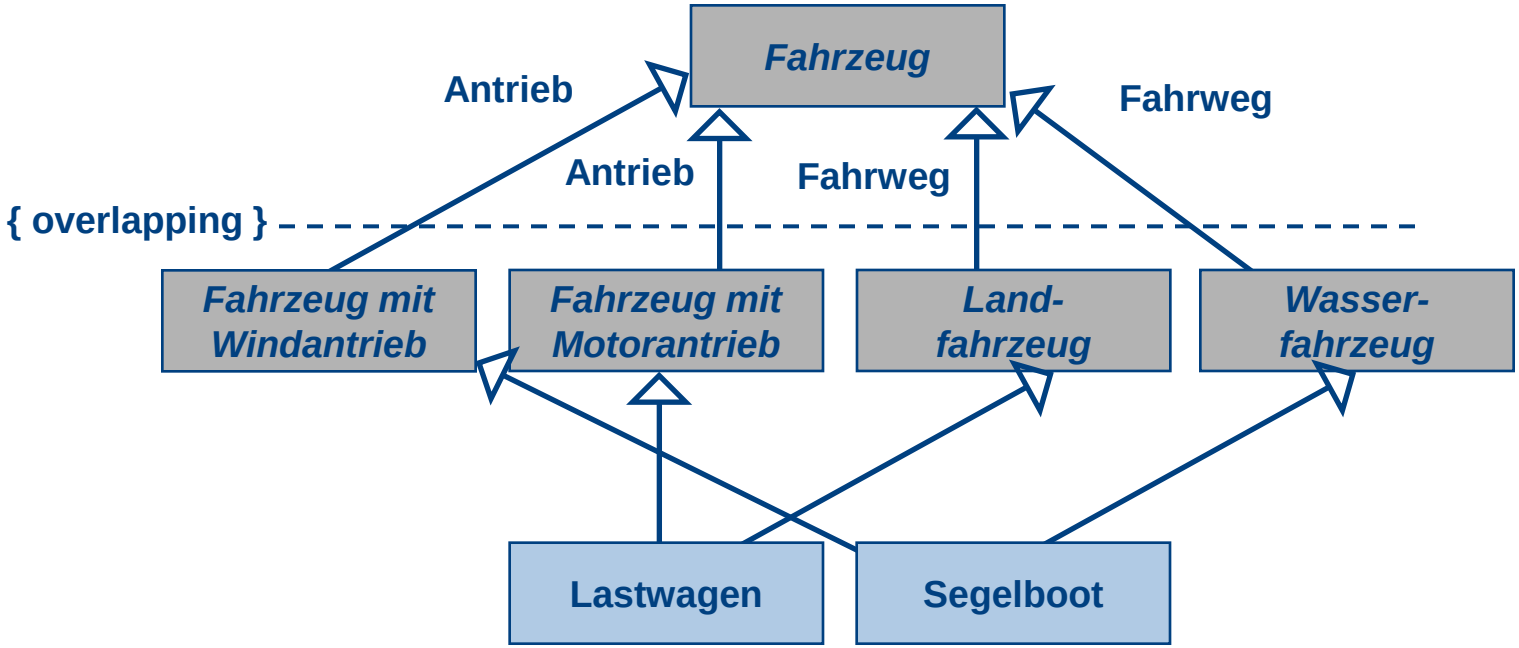
Vererbung

- OOA: Einfachvererbung
- OOD: + Mehrfachvererbung (in C++, aber nicht in Java)
  (enthalten in C++ mehrere Oberklassen Operationen mit identischer Signatur, so muss der Aufruf mit expliziter Angabe des Gültigkeitsbereichs erfolgen: sonst Fehlermeldung)



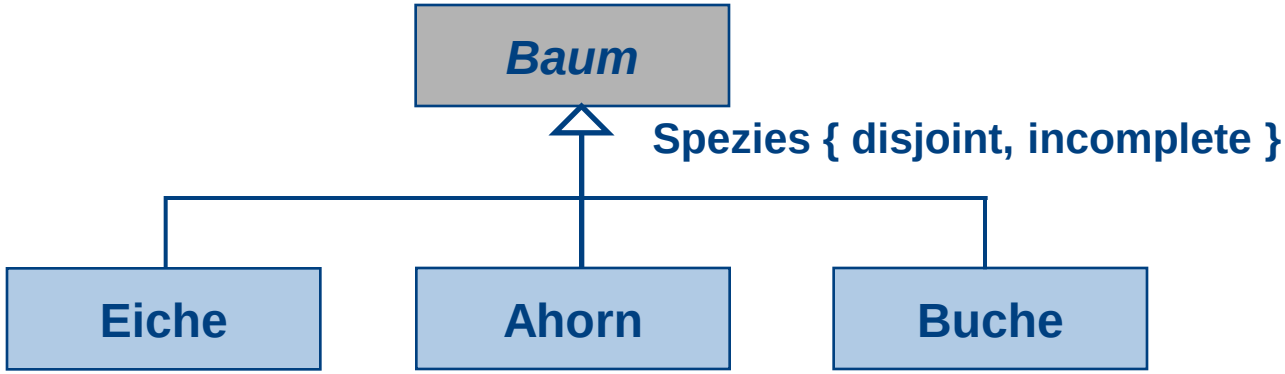
Vererbung

- Restriktionen
  - overlapping
    - Segelboot: Eigenschaften von Fahrzeugen mit Windantrieb und von Wasserfahrzeugen



Vererbung

- disjoint
  - Die Eigenschaften der Unterklassen überschneiden sich nicht



Vererbung

- complete
  - Die Menge der Unterklassen ist vollständig
  - Weitere Unterklassen werden aufgrund der Problemstellung nicht erwartet
- incomplete
  - Die betreffende Vererbungsstruktur enthält einen Teil der Unter-klassen
  - Es gibt weitere Unterklassen, die das Modell noch nicht enthält

Vererbung

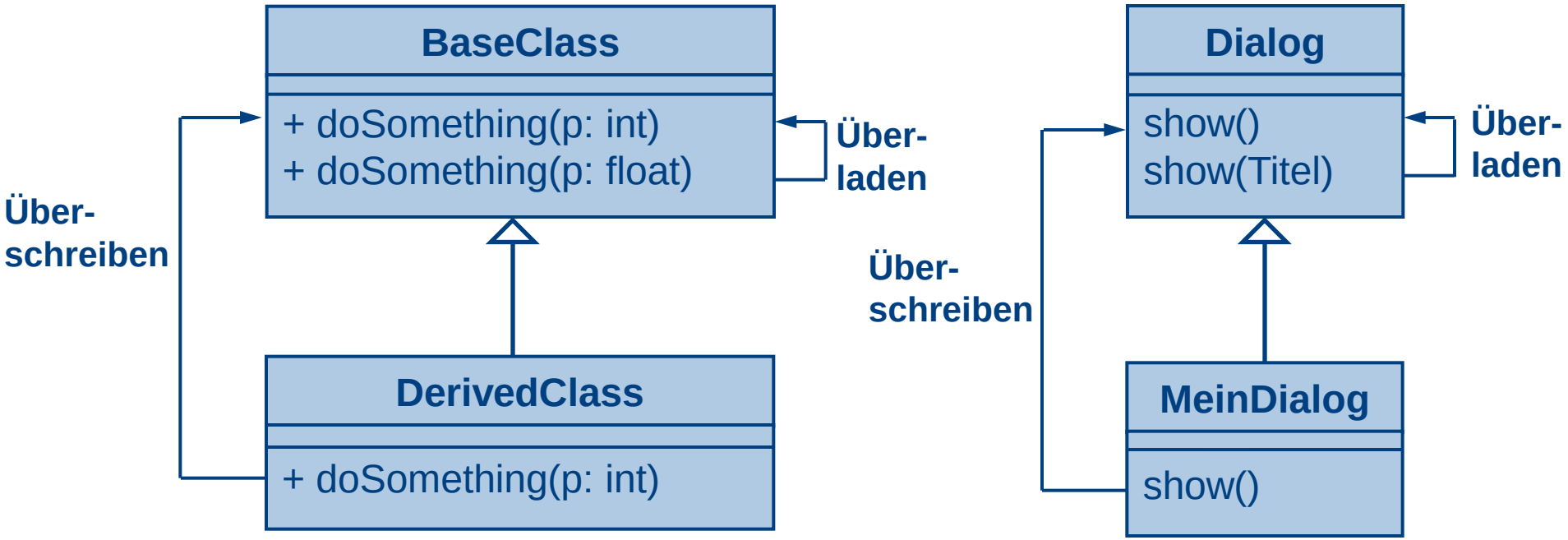
- Überschreiben bzw. Redefinition
  - Wenn eine Unterklasse eine Operation der Oberklasse – unter dem gleichen Namen – neu implementiert
  - Vorteil: Ein Programmierer, der eine Vererbungsstruktur benutzt, kann die verschiedenen (Unter-) Klassen verwenden und muss sich keine Gedanken darüber machen, zu welcher Unterklasse ein spezielles Objekt gehört
  - Diese Eigenschaft erfordert spätes Binden bzw. die Verwendung polymorpher Operationen
  - Beim Überschreiben einer Operation müssen die Anzahl und Typen der Ein-/ Ausgabeparameter gleich bleiben

Vererbung

- Überschreiben vs. Überladen
  - Überladen, wenn derselbe Operationsname innerhalb einer Klasse mit verschiedenen Parameterlisten verwendet wird
  - Beim Überschreiben muss die Operation der Unterklasse kompatibel mit derjenigen der Oberklasse sein
    - Dabei kommt es häufig vor, dass bei der Implementierung von `DerivedClass.doSomething()` die gleichnamige Operation der Oberklasse aufgerufen wird
    - Hierdurch zeigt sich das typische Verhalten der Unterklasse als Erweiterung der Oberklasse

Vererbung

- Beispiel
  - Überschreiben und Überladen von Operationen

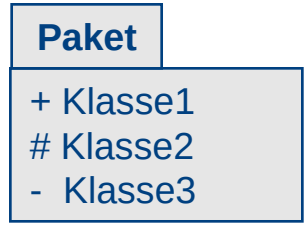


Pakete

- Dienen dazu, (Modell-)Elemente – insbesondere Klassen – zu Gruppen zusammenzufassen und als Ganzes zu behandeln
- Unterstützen die Darstellung von alternativen Entwürfen oder von Entwürfen für verschiedene Plattformen
- Pakete können ineinandergeschachtelt werden
  - Ermöglicht eine Modellierung des Systems auf verschiedenen Abstraktionsebenen

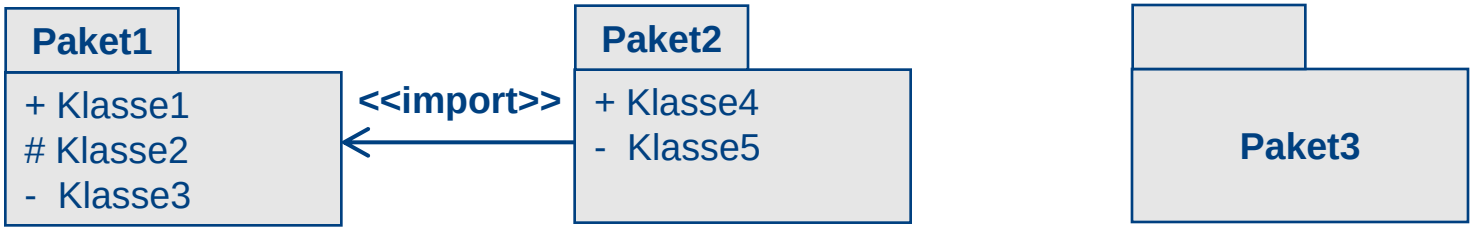
Pakete

- Sichtbarkeiten
  - Analog zu den Attributen und Operationen einer Klasse
  - +: Für alle Pakete sichtbar, die das betreffende Paket importieren
  - #: Für alle Pakete sichtbar, die das betreffende Paket spezialisieren
  - -: Nur in dem betreffenden Paket sichtbar
  - Wenn ein Element in einem Paket A sichtbar ist, dann ist es auch in allen Paketen A1, A2 sichtbar, die in A enthalten sind



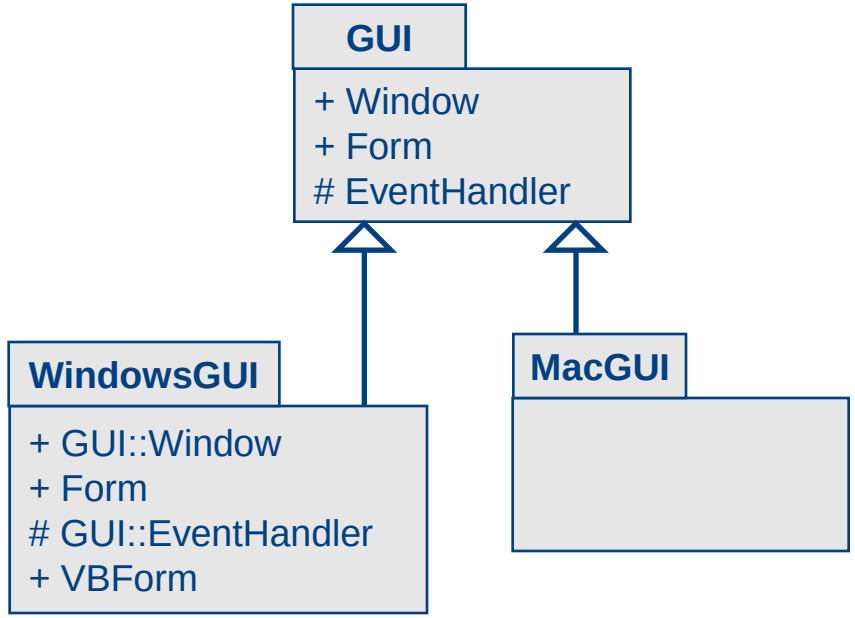
Pakete

- import
  - Zwischen zwei Paketen kann eine import-Beziehung definiert werden
  - In der Abbildung importiert Paket2 das Paket1
    - Das bedeutet, dass Paket2 die public-Klasse Klasse1 sieht
    - Für Paket3 sind die Klassen in Paket1 und Paket2 unsichtbar, weil keine import-Beziehung besteht
    - Die import-Beziehung ist nicht transitiv



Pakete

- Paketvarianten
  - Vererbungssymbol
  - Das spezialisierte Paket kann geerbte Elemente neu definieren und zusätzliche Elemente hinzufügen
  - Ein spezialisiertes Paket kann überall dort benutzt werden, wo das allgemeinere Paket verwendet werden kann



Pakete

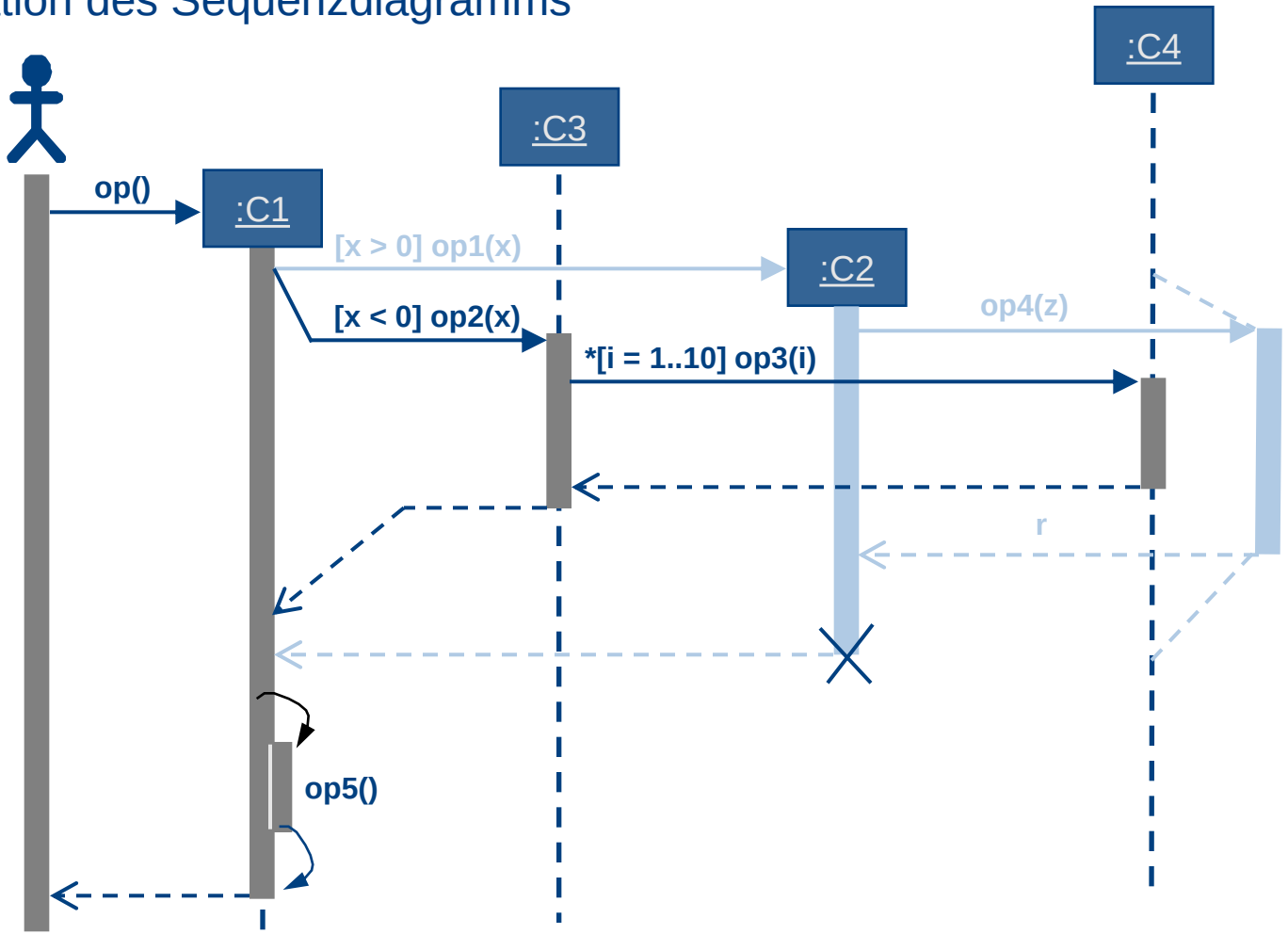
- Stereotypen
  - Der Paketname kann durch einen darüberstehenden Stereotypen ergänzt werden, um die Bedeutung des Pakets im System deutlich zu machen
  - Die UML definiert für Pakete eine Reihe von Standard-Stereotypen
    - *subsystem*: Das betreffende Paket modelliert ein unabhängiges Teilsystem
    - *system*: Das betreffende Paket repräsentiert das gesamte System

Szenario

- Zusammenarbeit der Objekte
  - Kann nur mittels geeigneter Szenarios beschrieben werden
    - Sequenz- und Kommunikationsdiagramme
- Sequenzdiagramme
  - Eine Verzweigung des Kontrollflusses tritt auf, wenn mehrere Botschaftspfeile vom selben Punkt ausgehen
  - Jeder Pfeil ist mit einer Bedingung *(guard condition)* beschriftet
  - Die Objektlinie kann in zwei oder mehrere Linien verzweigen, die zu einem späteren Zeitpunkt wieder zusammengeführt werden

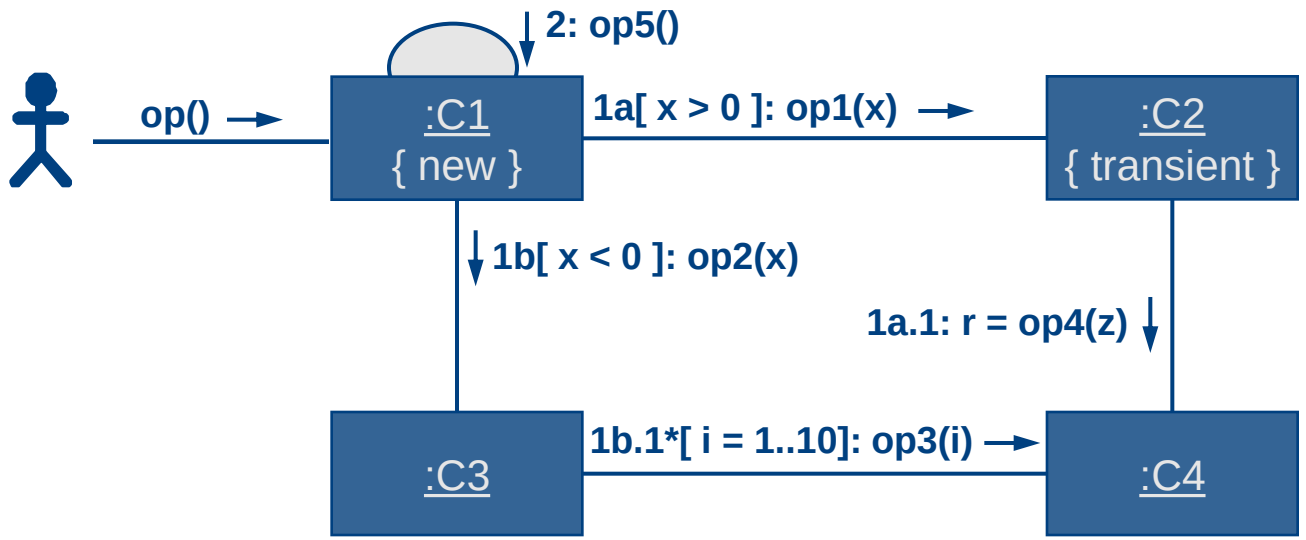
Szenario

- Notation des Sequenzdiagramms



Szenario

- Kommunikationsdiagramm
  - Ausgangsbasis bilden die Objekte und ihre Verbindungen untereinander
  - Die Reihenfolge der Operationen kann man zum Schluss durch entsprechende Nummern hinzufügen

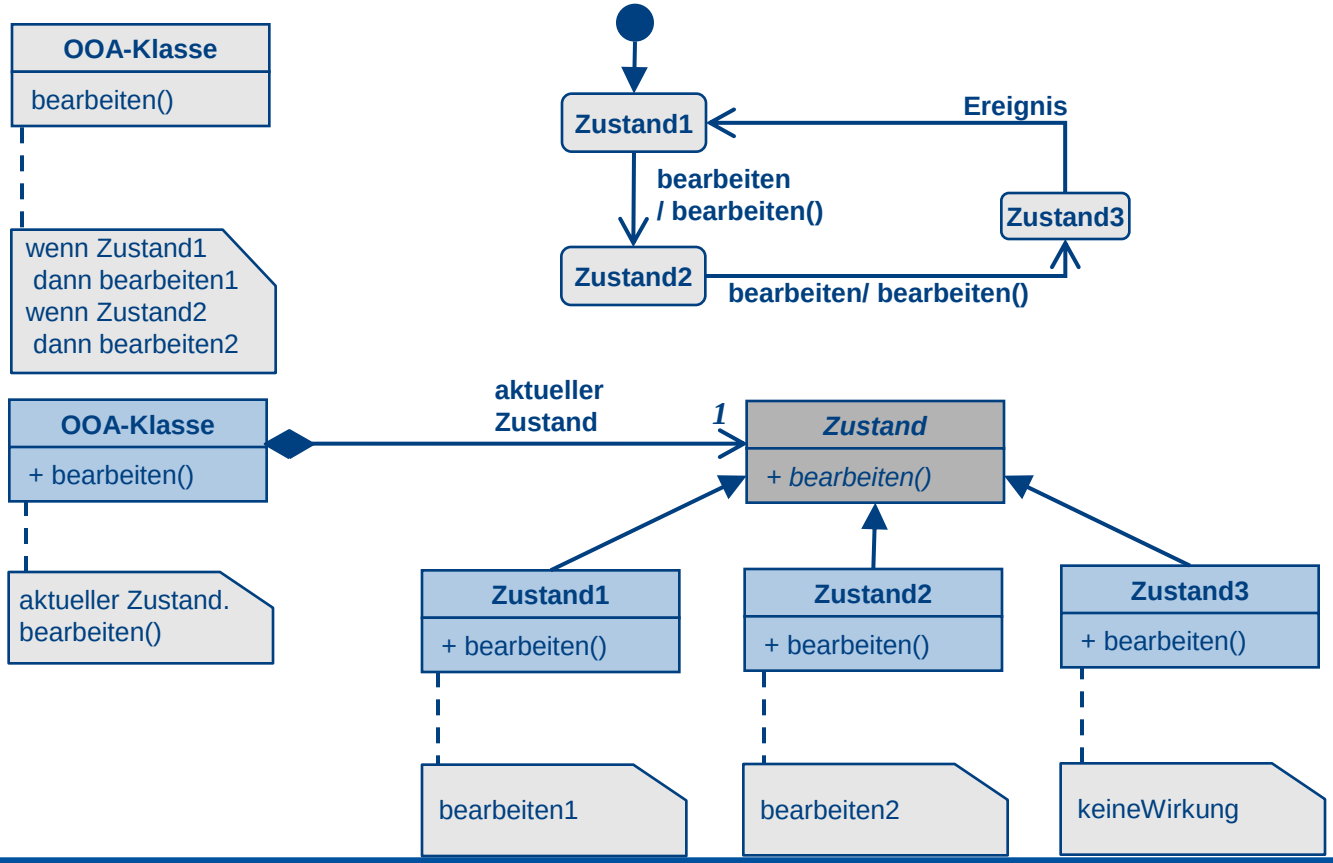


Zustandsautomat

- Lebenszyklus
  - Lebenszyklen aus Entwurfssicht überarbeiten und mit den entsprechenden Operationen ergänzen
  - Realisierung
    - Jede Klasse mit einem Lebenszyklus erhält im Entwurf ein private-Attribut `classState`
    - In diesem Attribut wird der aktuelle Zustand des Objekts gespeichert
    - Jede Operation, die im Lebenszyklus aufgeführt ist, muss dieses Attribut abfragen, bevor sie ihre Verarbeitung durchführt
    - Ist mit dieser Operation ein Zustandswechsel verbunden, dann muss sie das Zustandsattribut aktualisieren

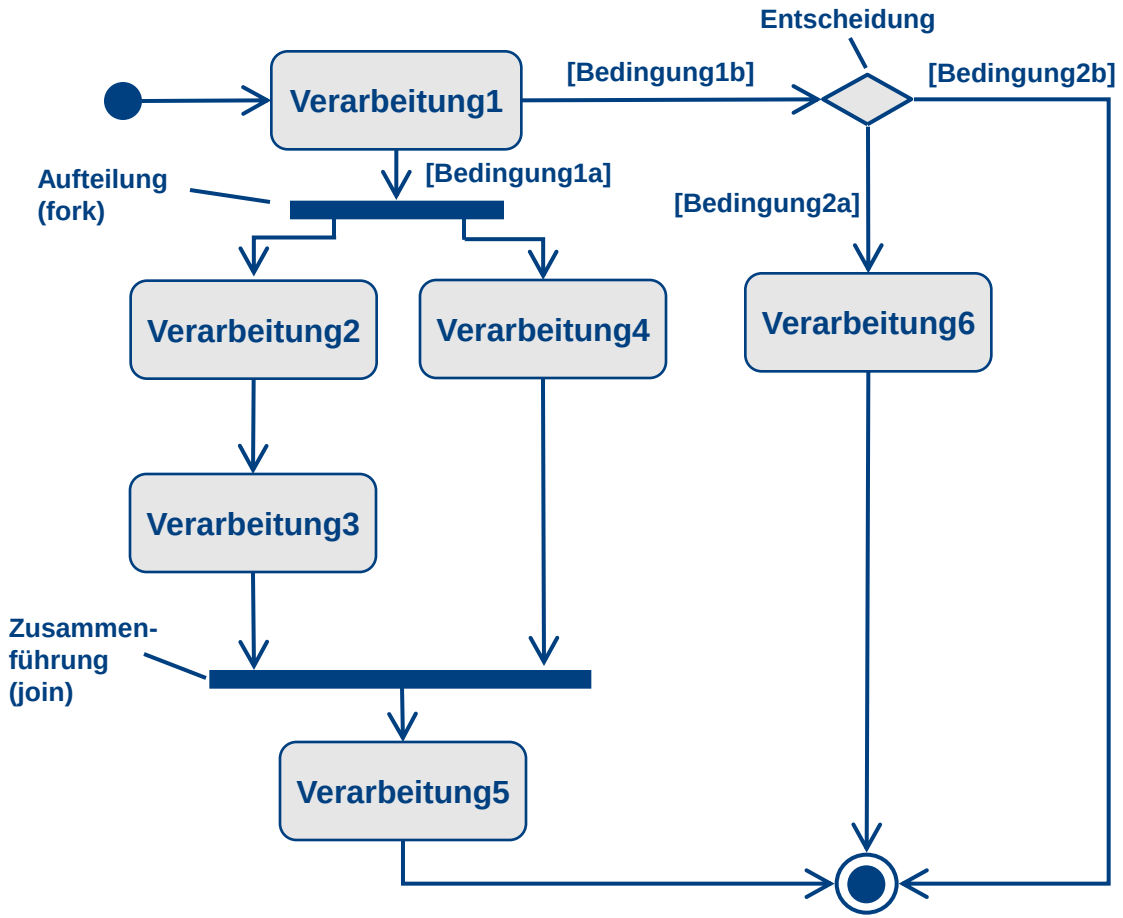
Zustandsautomat

- Zustandsmuster
  - Sinnvoll, wenn die Operationen in Abhängigkeit vom jeweiligen Zustand verschiedene Teilaufgaben ausführen



Zustandsautomat

- Aktivitätsdiagramm *(activity chart)*
  - Sonderfall des Zustandsdiagramms
  - Dient dazu, die interne Verarbeitung zu spezifizieren, wobei jeder Zustand einen Schritt eines Algorithmus beschreibt
  - Im Entwurf kann es sinnvoll zur Beschreibung von komplexen Operationen eingesetzt werden



Aus einem Kundengespräch

…und wie Sie sehen können, haben alle Artikel in unserem Lager einen festen Platz mit Lagerplatznummer. Außerdem soll zu jedem Artikel auch die Bezeichnung, der Preis und eine kurze Beschreibung angezeigt werden.

Eine Rechnung enthält, neben den Rechnungsposten, eine eindeutige Rechnungsnummer, die Kundennummer, ein Rechnungsdatum und zwei Adressen. Eine ist die Rechnungsadresse, die andere die Lieferadresse. Wir trennen beide selbst dann, wenn die Adressen identisch sind. Wir unterscheiden zwischen Privatadressen und Firmenadressen. Bei Firmenadressen gibt es die Felder Vorname und Name nicht, dafür aber den Firmen- und Abteilungsnamen. Die Kundennummer ist einfach eine Zahl. Vielleicht können wir sie in Zukunft nutzen um alle Bestellungen eines Kunden zu suchen. Ist es möglich, die Software bereits in 4 Wochen…