Technische Hochschule
Brandenburg
University of
Applied Sciences
Fachbereich
Wirtschaft

Objektorientierter Systementwurf

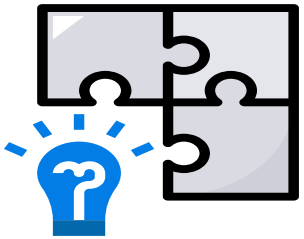
UML-Klassendiagramme

Prof. Dr. Kai Jander

Motivation (1)

- Klassen sind in Java in einzelnen Dateien
- Die Klassendateien sind u.U. in unterschiedlichen Paketen (Ordnern/Verzeichnissen)
- Klassen sind auf verschiedene Arten miteinander verknüpft (gekoppelt)
  - Klassen können Unterklassen von anderen Klassen sein
  - Klassen können Variablen vom Typ anderer Klassen beinhalten
  - Klassen können Methoden von Instanzen anderer Klassen oder (statische) Methoden anderer Klassen aufrufen
- Klassen enthalten viele Methoden und Attribute
- Wie behält man die Übersicht, welche Daten und Methoden wie und wo benutzt werden?

Motivation (2)



- Sie starten ein neues Projekt
- In diesem Projekt müssen Sie die Geschäftsobjekte der Anwendung erstellen
- Dies wird voraussichtlich viele Klassen erfordern
- Sie arbeiten im Team mit anderen zusammen
- Die Klassen zu implementieren ist aufwendig, sie hinterher nochmal zu ändern noch aufwendiger
- Wie können schon vor der Implementierung mit Ihrem Team das zukünftige Klassenmodell besprechen?

UML-Klassendiagramme (1)

- UML-Klassendiagramme können helfen, ein Klassenmodell (grafisch) zu visualisieren und zu dokumentieren
- Die Universal Modeling Language (UML) ist eine standardisierte Modellierungssprache für die Softwareentwicklung
- Standardisiert durch die Object Management Group (OMG)
- Erste Version offizielle Version 1.1, veröffentlicht 1997
- Aktuelle Version 2.5 vom Juni 2015

UML-Klassendiagramme (2)

- UML-Klassendiagramme sind nur ein Typ von Diagramm in UML, welches auch weitere Diagrammtypen enthält, wie etwas Komponentendiagramme, Aktivitätsdiagramme, Use Case-Diagramme, Sequenzdiagramme, ...
- UML dient der Modellierung und Dokumentation von Softwaresystemen
- UML-Klassendiagramme können genutzt werden, um
  - Klassen vor der Implementation zu modellieren
  - Klassen nach der Implementation zu dokumentieren
- UML-Klassendiagramme können entweder in Editoren wie Modelio erstellt werden oder mit Generatorprogramme aus vorhandenen, implementierten Klassen generiert werden

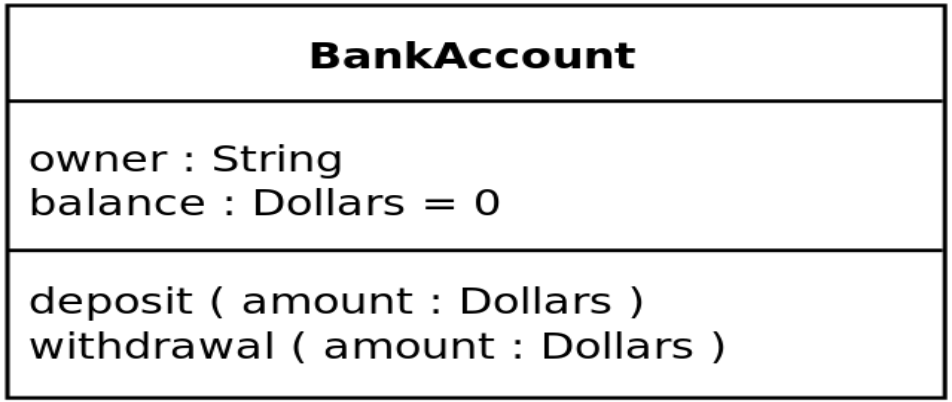
UML-Klassendiagramme (3)

- UML-Klassendiagramme sollen vor allem durch Menschen gelesen und interpretiert werden
- Man kann das Modell mehr oder weniger abstrakt auslegen
- In einem Extrem ist der Detailgrad so hoch, dass sich automatisch Codevorlagen (Stubs) aus dem Modell generieren lassen, welche nur noch mit Methodenimplementationen gefüllt werden müssen
- Das andere Extrem bildet z.B. nur grob die Beziehungen zwischen den Klassen ab, geht bei den einzelnen Klassen nicht stark ins Detail
- Welcher Detailgrad nötig ist, hängt vom konkreten Grund für das Erstellen des Diagramms ab
- Wir werden hier nur die wichtigsten Modellierungsformen von UML-Klassendiagrammen kennenlernen

Beispielklasse in UML

- Klassen werden in UML-Klassendiagrammen als Rechtecke dargestellt

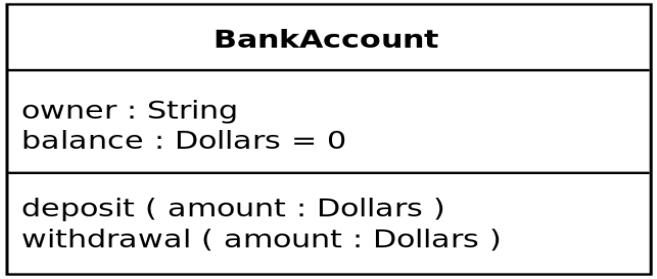
- Die Rechtecke sind in drei Abschnitte eingeteilt:
  - Der erste Abschnitt enthält den Namen der Klasse
  - Der zweite Abschnitt enthält die Datenfelder der Klasse
  - Der dritte Abschnitt enthält die Methoden der Klasse



https://en.wikipedia.org/wiki/File:BankAccount1.svg

Datenfelder und Methoden: Allgemein

- Datenfelder werden UML-Klassendiagramme haben, wie in Java, einen Datentyp und einen Namen
  - Es können zusätzliche Angaben für z.B. Schnittstellen oder abstrakte Klassen gemacht werden
- Der Name und der Datentyp werden durch einen Doppelpunkt voneinander getrennt
- Zusätzlich kann ein Initialwert angegeben werden



https://en.wikipedia.org/wiki/File:BankAccount1.svg

Datenfelder und Methoden: Sichtbarkeit

- Auch die Sichtbarkeit von Datenfelder können in UML-Klassendiagramme abgebildet werden

- Wie in Java unterscheidet man 4 verschiedene Sichtbarkeiten
- Diese werden definiert, indem man dem Namen eines der folgenden Symbole voranstellt:

+	public
-	private
#	protected
~	package

- Die Sichtbarkeiten können auch für Methoden verwendet werden

Datenfelder und Methoden: Zugehörigkeit (Scope)

- Bei der Zugehörigkeit von Datenfelder und Methoden werden in UML-Klassendiagrammen zwei Arten unterschieden:

  - Instanzvariable/-methoden gehören zu den Instanzen der Klassen und können sich z.B. in Wert und Verhalten zwischen Instanzen unterscheiden
    - Dies entspricht den “normalen” Instanzvariablen und -methoden in Java

  - Klassifikatorvariablen/-methoden gehören zu der Klassen und nicht der Instanz und sind somit für alle Instanzen gleich
    - Dies entspricht der Bedeutung des Java-Schlüsselworts “static” für Datenfelder und Methoden

Datenfelder und Methoden: Weitere Eigenschaften

ordered – Rückgabedaten sind geordnet

redefines – Die Methode überschreibt eine Operation einer Oberklasse

read-only – Die Variable kann nur gelesen werden

in – Der Methodenparameter wird nur gelesen

out – Der Methodenparameter wird nur beschrieben

inout – Der Methodenparamter wird gelesen und beschrieben

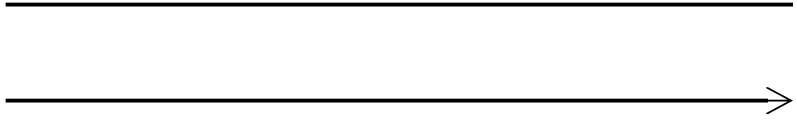
Klassenbeziehungen: Assoziation

- Eine **Assoziation** beschreibt eine allgemeine Beziehung zwischen zwei oder mehr Klassen
- Die Beziehung kann einseitig gerichtet (Pfeil) oder beidseitig ungerichtet (Linie) sein
- Zusätzlich kann eine **Multiplizität** vermerkt werden
  - Diese beschreibt, wieviele Instanzen einer Klasse mit wievielen Instanzen einer andere Klassen assoziiert sind
  - Mögliche Werte sind:

    1..1

    1..n

    n..m



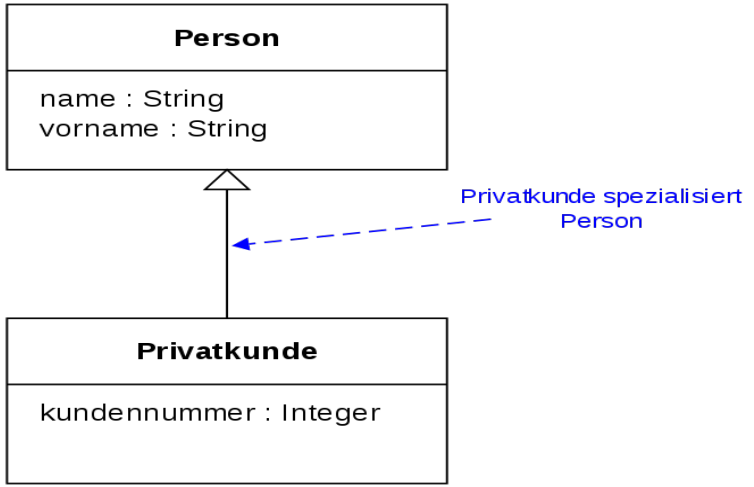
Klassenbeziehungen: Generalisierung / Vererbung

- Eine **Generalisierung** legt fest, dass eine Klasse eine Spezialisierung (oder Generalisierung in die andere Richtung) einer andere Klasse darstellt
- Die Beziehung ist eine “ist ein” (is a)-Beziehung

- Beispiel: Ein Privatkunde “ist eine” Person

- Dies entspricht der Vererbung in Java

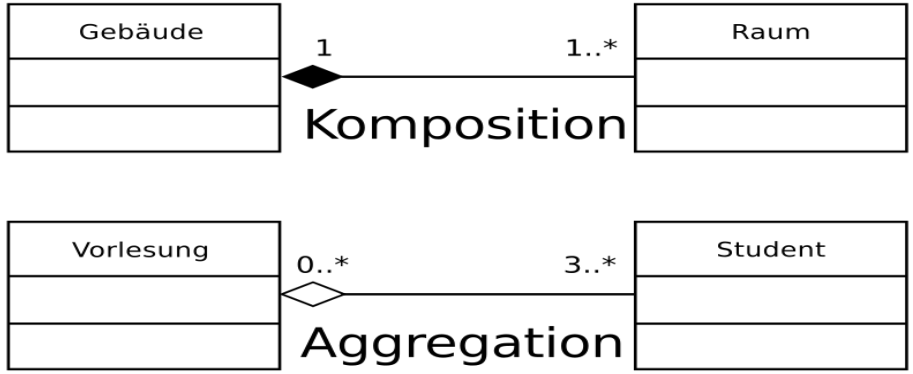
- Die Verwendung “ist ein” eignet sich auch gut um zu prüfen, ob eine solche Beziehung Sinn ergibt



https://de.wikipedia.org/wiki/Datei:UmlCd_Generalisierung-1.svg

Klassenbeziehungen: Komposition

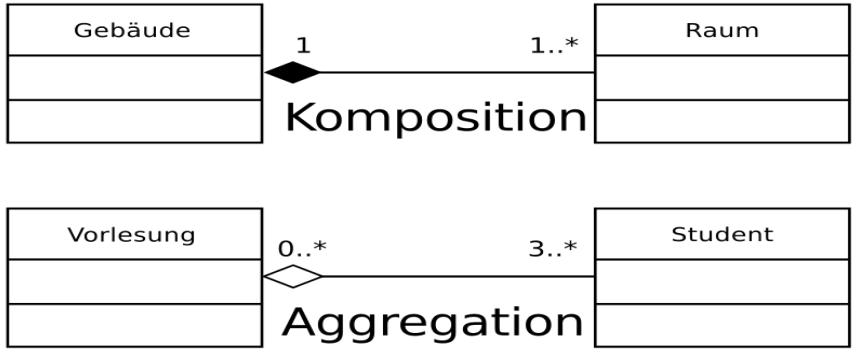
- Eine **Komposition** legt fest, dass Instanzen einer Klasse einen Teil von Instanzen einer anderen Klasse darstellt
- Es kann eine Kardinalität festgelegt werden, welche definiert, wie viele Instanzen auf beiden Seiten in eine solchen Beziehung stehen können
- Beispiel: Ein Gebäude besteht aus einem oder mehreren Räumen
- Bei der Komposition ergeben die Teile ohne das Ganze keinen Sinn: Die Räume sind ohne das zugehörige Gebäude sinnlos



https://de.wikipedia.org/wiki/Datei:Komposition_Aggregation.svg

Klassenbeziehungen: Aggregation

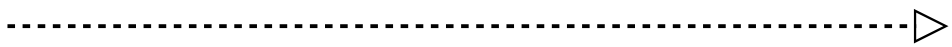
- Eine **Aggregation** legt fest, dass Instanzen einer Klasse Instanzen einer anderen Klasse beinhaltet
- Auch hier kann eine Kardinalität festgelegt werden
- Allerdings ist die Interpretation der Beziehung gegenüber der Komposition schwächer:
- Beispiel: Eine Vorlesung kann von 3 oder mehr Studenten besucht werden
- Bei der Aggregation ergeben die Teile ohne das Ganze weiterhin Sinn: Wenn die Vorlesung nicht mehr angeboten wird, sind die Studenten weiterhin da.



https://de.wikipedia.org/wiki/Datei:Komposition_Aggregation.svg

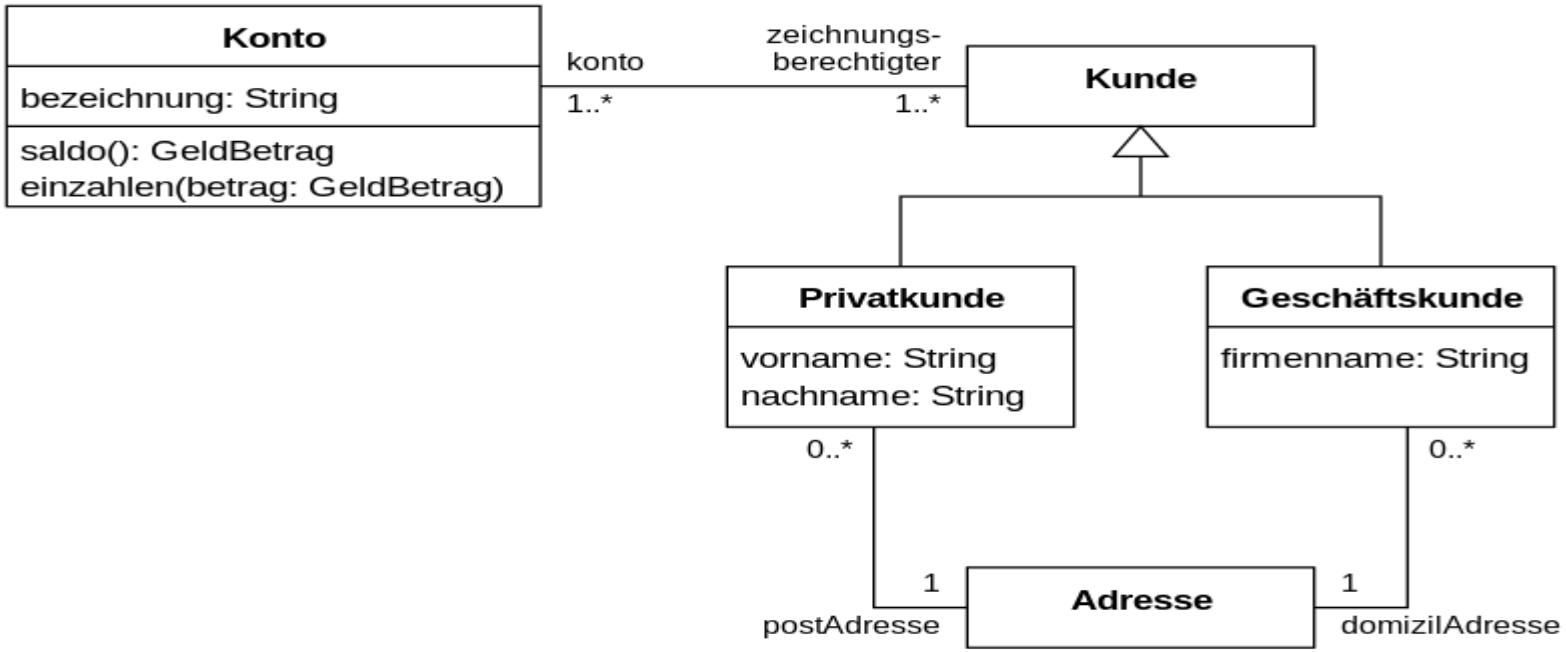
Weitere Klassenbeziehungen

- Realisierung oder Implementierung drückt aus, dass eine Klasse z.B. eine Schnittstellen implementiert (Java: implements)



- Abhängigkeiten drücken aus, dass eine Änderung in einer Klasse Änderungen in einer andere Klassen zur Folge haben kann

Beispieldiagramm



Diskussion: Vererbung vs. Komposition (1)

- Es gibt grundsätzlich zwei Möglichkeiten, die Funktionalität einer Klasse in einer anderen zu benutzen

- Grundsätzlich ist die Benutzung von vorhandener Funktionalität gut: Man kann vorhandenen Code wiederverwenden und muss ihn nicht nochmal schreiben

- Gleichzeitig mussen man den Code nur einmal warten, bei Änderungen muss der Code nur an einer Stelle angepasst werden

Diskussion: Vererbung vs. Komposition (2)

- Vererbung ist ein Kernkonzept der Objektorientierung
- Weil es so ein zentrales Konzept ist, wird es gerade von neuen Programmierern zu oft verwendet
- Vererbung koppelt die Unterklasse stark an die Oberklasse
- Vererbung führt zu einer “ist ein”-Beziehung
- Ein Auto “ist ein” Fahrzeug

Diskussion: Vererbung kann schief gehen (1)

- Auch wenn Vererbung richtig erscheint, ist es nicht immer die richtige Lösung

- Beispiel: Eine Oberklasse “Tier” hat eine Methode fressen() und zwei Unterklassen:
  - Die Klasse “Katze” hat die Methode miauen()
  - Die Klasse “Hund” hat die Methode bellen()
- Diese Aufteilung erscheint sinnvoll: Hunde und Katzen sind Tiere und beide können fressen

- Zusätzlich gibt es eine Klasse “Roboter” mit der Methode fahren() und zwei Unterklassen:
  - Die Klasse “PutzRoboter” mit der Methode putzen()
  - Die Klasse “KillerRoboter” mit der Methode toeten()
- Auch hier erscheint die Aufteilung zunächst sinnvoll

Diskussion: Vererbung kann schief gehen (2)

- Unser Softwareprojekt entwickelt sich weiter
- Die Klassen “Tier”, “Hund”, “Katze”, “Roboter”, “PutzRoboter” und “KillerRoboter” haben sich gut entwickelt
- Eines Tages kommt der Kunde herein und möchte einen “KillerRoboterHund”
- Dieser soll fahren(), bellen() und toeten(), aber nicht fressen()

- Nun haben wir ein Problem:
  - Wir können von KillerRoboter erben und bellen() nachträglich einbauen, aber jetzt haben wir bellen() zweimal gebaut
  - Wir können für Tier und Roboter eine neue Oberklasse “Klaeffer” einführen, die bellen() implementiert, aber jetzt kann die Katze, der Putzroboter und der KillerRoboter auch bellen()

Diskussion:
Lösung mit Komposition

- Lösung: Statt unser Projekt mit Vererbungshierarchien aufzubauen, verwenden wir Kompositionen
- Wir führen 6 Klassen ein, welche wir als Teile verwenden wollen und die Funktionalität implementieren:
  Fresser, Klaeffer, Miauer, Fahrer, Putzer und Killer
- Nun können wir unsere Anwendungsklassen aus diesen Klassen zusammensetzen:
  - Hund = Fresser + Klaeffer
  - Katze = Fresser + Miauer
  - PutzRoboter = Fahrer + Putzer
  - KillerRoboter = Fahrer + Killer
  - KillerRoboterHund = Fahrer + Killer + Klaeffer
- Fazit: Mit Vererbung vorsichtig umgehen, im Zweifel Kompositionen verwenden

Komposition vs. Vererbung

https://youtube.com/watch?v=wfMtDGfHWpA

Zusammenfassung

- Die Unified Modeling Language ist eine Modellierungssprache für Softwaresysteme
- Sie enthält eine Vielzahl an Diagrammtypen, eines dieser Diagrammtypen ist das UML-Klassendiagramm
- Mit UML-Klassendiagramme lassen sich Klassen grafisch modellieren
- Es gibt Generatoren und Modellierungstools wie Modelio https://www.modelio.org/
- Diskussion: Komposition vs. Vererbung

Die drei Tugenden eines guten Programmierers

Nach Larry Wall (Entwickler der Programmiersprache Perl):

- **Faulheit:** Die Tugend, welche den Programmierer zu großem Einsatz verleitet, um den Gesamtaufwand zu reduzieren: Programmiere Werkzeuge, die einem die Arbeit verleiten, dokumentiere alles, damit man nicht so viel erklären muss.

- **Ungeduld:** Die Wut, die einen packt, wenn der Computer faul ist. Dies führt dazu, dass man Programme schreibt, welche nicht nur den Anforderungen genügen sondern diese antizipieren (oder es zumindest versuchen).

- **Stolz:** Die Tugend, die einen dazu bringt Programme zu schreiben und zu warten über die andere nicht schlecht reden.

Fragen