Technische Hochschule Wildau
Technical University of Applied Sciences

Skript zur Vorlesung

Einführung in die Wirtschaftsinformatik

Prof. Dr. Mathias Walther

Wintersemester 2021/22

Inhaltsverzeichnis

5 Modellierung von Software

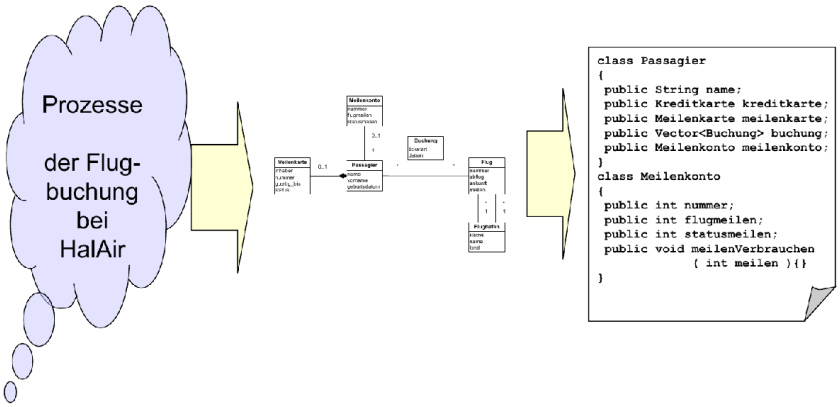
5.1 UML

Was ist UML? 1

- „Unified Modeling Language“
- eine Sprache zur Beschreibung von Softwaresystemen
- UML wird benutzt für:
  - Analyse
  - Entwurf
  - Implementierung
  - Dokumentation
- im Rahmen der *objektorientierten* Softwareentwicklung

UML in der Softwareentwicklung 2

UML ist eine wichtige Modellierungsebene zwischen der Realität und Implementierung der Software



UML heute 3

- UML liegt in der Version 2.5 vor
- Konzepte aus vielen Modellen integriert und konsolidiert
- als Standard in der Objektorientierten Softwareentwicklung akzeptiert
- wird von einer Vielzahl von Tools unterstützt
- aktive Weiterentwicklung durch die OMG (Object Management Group)

⇒ internationales Konsortium aus über 800 Unternehmen

Objektorientierte Softwareentwicklung

- Objektorientierung (OO) ist eine natürliche Sichtweise auf komplexe Systeme

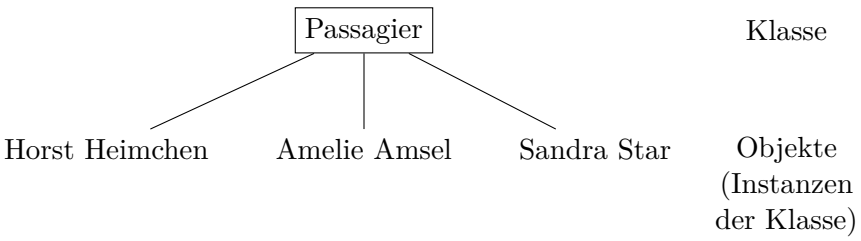
⇒ Systeme als Zusammenspiel kooperierender Objekte

- diese Sichtweise wird in der OO-Softwareentwicklung in allen Phasen eingenommen:
  - OOA: Objektorientierte Analyse der Realität
  - OOD: Objektorientiertes Design (Entwurf) der Software
  - OOP: Implementierung in einer objektorientierten Programmiersprache (z. B.: Java, C++, C#, Python ...)

Objektorientierung: Klassen und Objekte

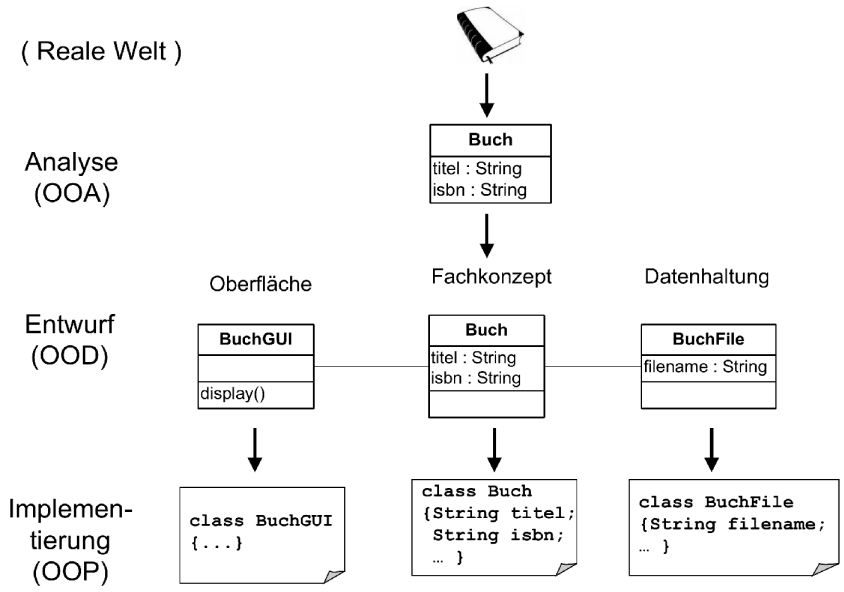
„Klasse“ ist ein zentraler Begriff in der Objektorientierung

- eine Klasse ist ein Objekttyp
- ein Objekt ist eine Instanz einer Klasse



UML in der Softwareentwicklung

UML ist eine wichtige Modellierungsebene zwischen der Realität und Implementierung der Software



Werkzeuge zu UML

7

- UML-Modelle können auf dem Papier entwickelt werden
- meist wird jedoch Software genutzt:

  Zeichnen von Diagrammen:

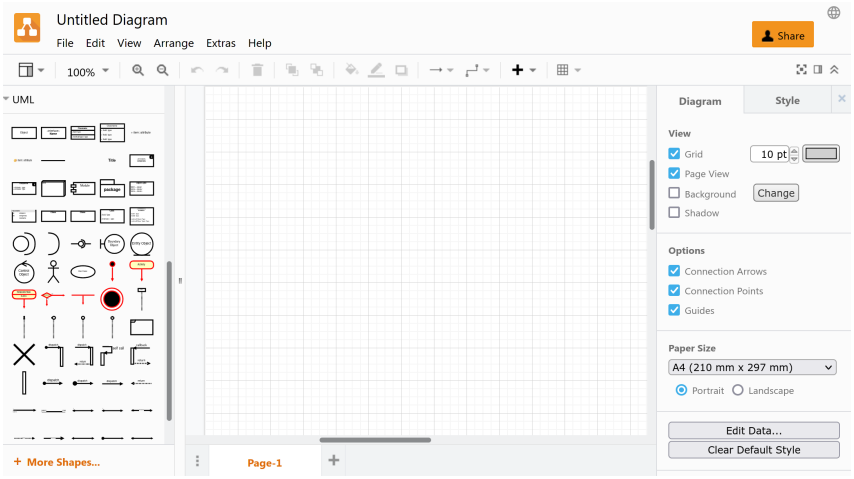
  - spezielle Werkzeuge, z.B. violet
  - allgemeine Zeichentools bieten Schablonen für UML an:
    - MS Visio
    - dia (http://dia-installer.de/index.html.de)
    - violet UML (https://sourceforge.net/projects/violet/)
    - https://diagrams.net/ (ehemals draw.io)

  Codegenerierung:

  - einige Tools können darüber hinaus aus UML- Diagrammen auch Quellcode oder zumindest Codefragmente erzeugen

Beispiel diagrams.net

8



Diagrammarten (I)

9

Strukturdiagramme:

- modellieren die Struktur des Systems, d. h.
  - die statischen, zeitunabhängigen Elemente
  - Elemente, die den Zustand des Systems beschreiben

Verhaltensdiagramme:

- modellieren das Verhalten des Systems, d.h.
  - dynamische Abläufe im System und insb.
  - zeitabhängige Interaktionen der Elemente untereinander

Diagrammarten (II)

10

- insgesamt gibt es 13 Diagrammarten, die jeweils eine bestimmte Sicht auf das System modellieren
- Unterteilung in:

Strukturdiagramme:

- *Klassendiagramm*
- Komponentendiagramm
- Kompositionsstrukturdiagramm
- Packagediagramm
- Objektdiagramm
- Verteilungsdiagramm

Verhaltensdiagramme:

- *Use-Case-Diagramm*
- *Aktivitätsdiagramm*
- Interaktionsübersichtsdiagramm
- Kommunikationsdiagramm
- Sequenzdiagramme
- Zeitverlaufsdiagramm
- Zustandsdiagramm

5.2 Klassendiagramm

Klassendiagramm

11

- bekanntestes, aber nicht einziges Diagramm der UML
- Entity-Relationship-Modell ist ein Vorläufer
- beschreibt die Klassen (Objekttypen) des Systems
  - deren Eigenschaften (Attribute)
  - deren Operationen (d. h. Methoden)
- und die Beziehungen der Klassen untereinander

Unterschiede zu ERM:

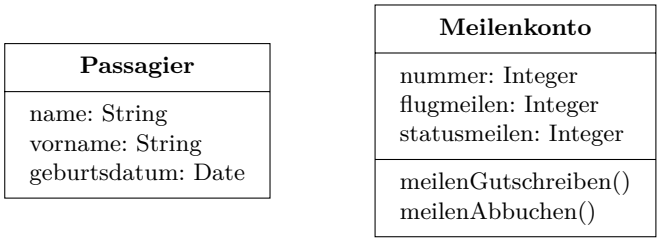
- Klassendiagramme ⇒ ERM für DB + Methoden für Klassen
- Objekt ⇒ Entität
- Klasse = Objekttyp ⇒ Entitätstyp
- Assoziation = Beziehung zw. Klassen ⇒ Relationship

Klassen

12

- „Schablonen“ für Objekte je nach Bedarf sind unterschiedliche Detaillierungsstufen für die Modellierung von Klassen anwendbar
  - einfachste Darstellungsform als Rechteck:

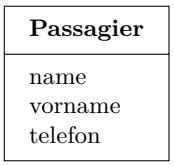
– komplexere Darstellung mit Attributen und Operationen



Klassen: Attribute

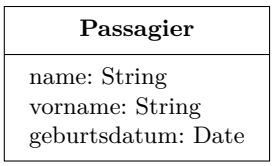
13

- Eigenschaften/Daten von Klassen
- Attribute stehen im zweiten Abschnitt des Klassendiagramms



Typen von Attributen:

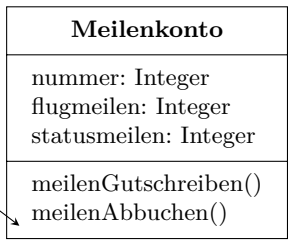
- für jedes Attribut kann ein Typ angegeben werden, z. B.
  - String: eine Zeichenkette
  - Integer: eine ganze Zahl
  - Double: eine Dezimalzahl
  - Date: ein Datum
- Typen können auch andere Klassen sein



Klassen: Attribute

14

- Objekte können auch Operationen haben
- Operationen stehen im dritten Abschnitt

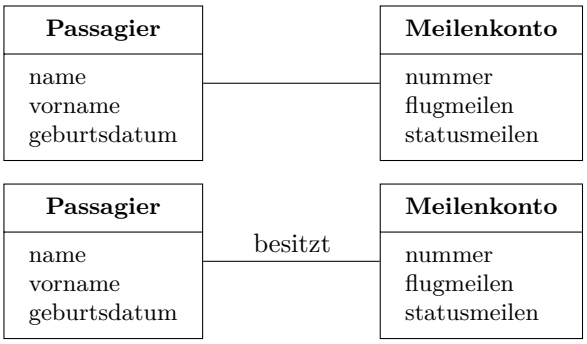


- je nach Zweck der Darstellung können die Abschnitte ein- oder ausgeblendet werden

Assoziationen

15

Wenn eine Beziehung zwischen zwei Klassen besteht, wird eine Linie zwischen diesen Klassen gezogen. Eine allgemeine Beziehung nennt man *Assoziation*.



Namen für Assoziationen

16

- man kann auch Namen bzw. Bezeichnungen für Assoziationen angeben
- Darstellung: an der Assoziation kennzeichnet ein kleines schwarzes Dreieck die Leserichtung

Multiplizitäten

17

- Multiplizitäten kennzeichnen, auf wie viele Objekte von der „Zielseite“ der Assoziation sich ein „Quellobjekt“ bezieht

  - ein Kunde kann mehrere Aufträge erteilen
  - jeder Auftrag wird von genau einem Kunden erteilt

Beispiele für weitere mögliche Kennzeichnungen:

1 ⇒ genau ein Objekt

0..1 ⇒ 0 oder ein Objekt

* ⇒ 0 bis viele Objekte

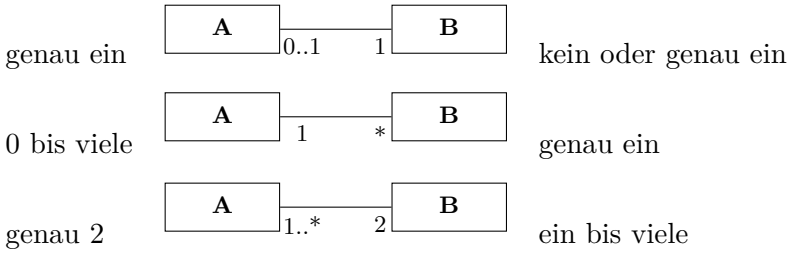
1..* ⇒ 1 bis viele Objekte

2..5 ⇒ 2 bis 4 Objekte

ℹ wird keine Multiplizität angegeben, so nimmt man automatisch die 1 an

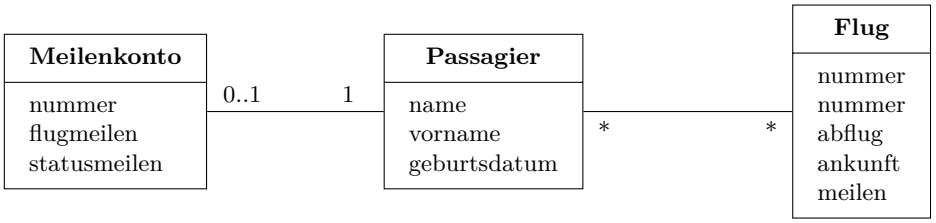
Lesart für Multiplizitäten

Ein Objekt vom Typ A „kennt“... *Ein* Objekt vom Typ B kennt...



...Objekt(e) vom Typ B. ...Objekt(e) vom Typ A

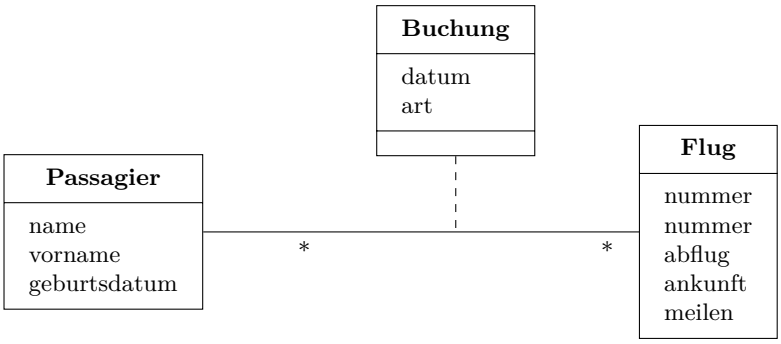
Beispiel



- ein Passagier hat maximal ein Meilenkonto
- ein Meilenkonto gehört zu genau einem Passagier
- ein Passagier kann mehrere Flüge buchen
- ein Flug kann mehrere Passagiere haben

Assoziationsklassen

- manchmal enthalten Assoziationen relevante Daten
- in solchen Fällen bildet man Assoziationsklassen mit diesen Daten als Attributen
- Darstellung mit einer gestrichelten Linie an der Assoziation

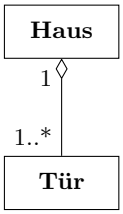


Aggregation und Komposition

21

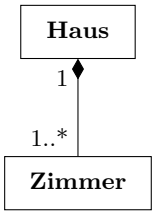
Aggregation:

- Aggregation ist eine spezielle Assoziation
- Abbildung einer „ist Teil von“ – Beziehung
- leere Raute



Komposition:

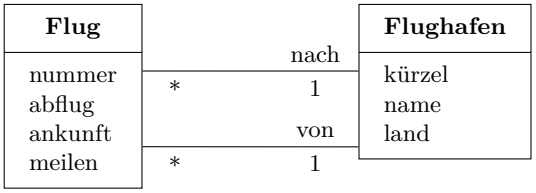
- Komposition ist eine strenge Form der Aggregation
- das Teil besteht nur, solange das Ganze besteht
- wird das Ganze gelöscht, so wird dasTeil ebenfalls gelöscht
- schwarze Raute



Rollen

22

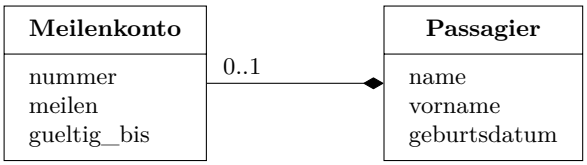
- Bezeichnung an einem Ende einer Assoziation
- kennzeichnet die Rolle, die das entsprechende Objekt für das andere Objekt spielt



Komposition im Softwareentwurf

23

- das Haus- /Zimmer-Beispiel zeigt eine Kompositionsbeziehung in der Realwelt
- beim Softwareentwurf kann es sinnvoll sein, „künstliche Kompositionen“ einzuführen
- wenn das Passagier-Objekt gelöscht wird, wird automatisch das zugehörige Meilenkarten-Objekt gelöscht

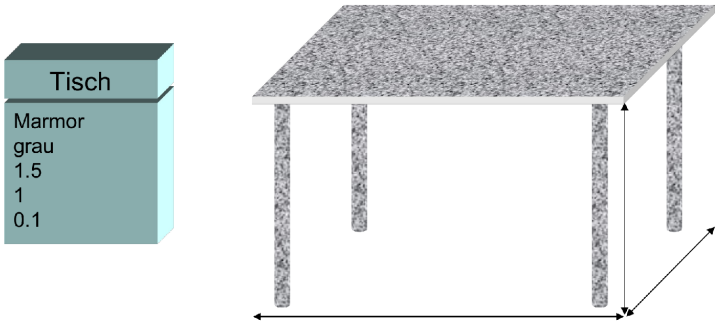


Beispiel

24

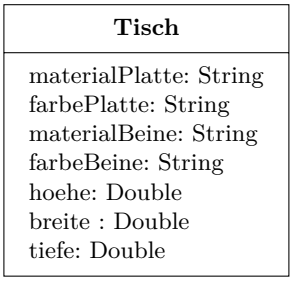
Modellierung eines Tischs mit:

- Material,
- Farbe,
- Höhe,
- Breite und
- Tiefe

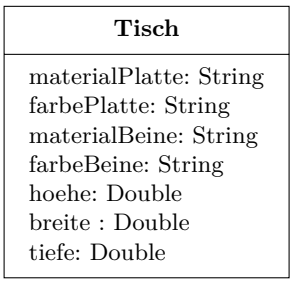


Lösung

25

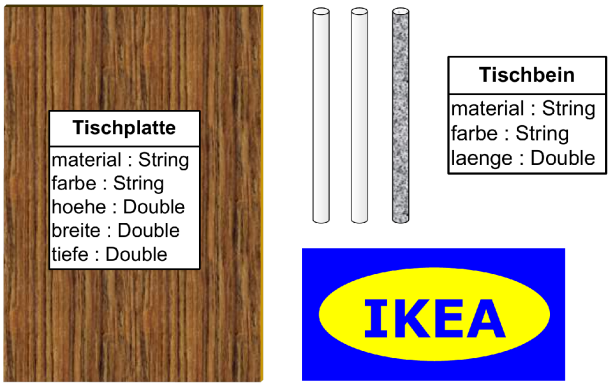


Was passiert, wenn Platte und Beine verschieden sein sollen?



Das IKEA-Prinzip

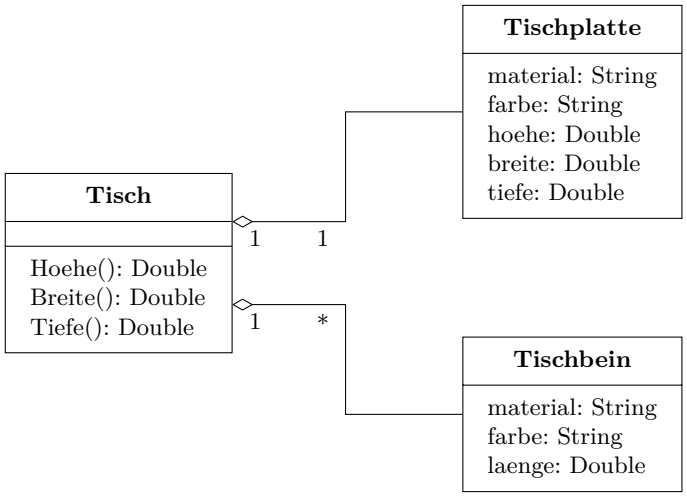
26



Lösung: Modellierung durch Aggregation

27

Ein Tisch besteht aus einer Tischplatte und mehreren Tischbeinen



Generalisierung und Vererbung

28

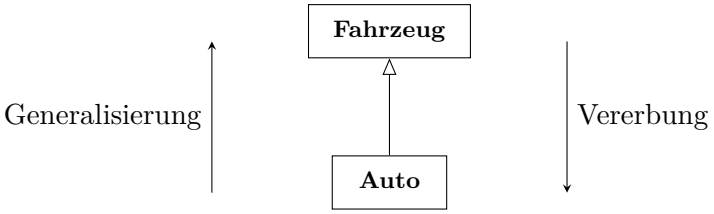
- „is a“-Beziehung zentrales Element der Objektorientierung

- wird durch einen Pfeil mit weißer Spitze dargestellt, der vom Speziellen zum Allgemeinen geht

Generalisierung und Vererbung (I)

29

- „ist ein“-Beziehung zentrales Element der Objektorientierung
- wird durch einen Pfeil mit weißer Spitze dargestellt, der vom Speziellen zum Allgemeinen geht

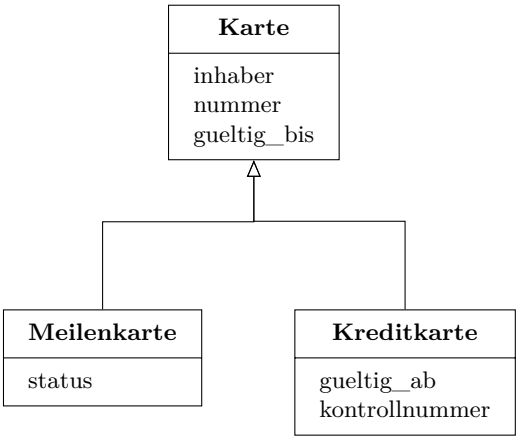


- Lesart: „ein Auto ist ein Fahrzeug“ ⇒ guter „Trick“, um zu prüfen, ob eine Vererbungsbeziehung vorliegt

Generalisierung und Vererbung (II)

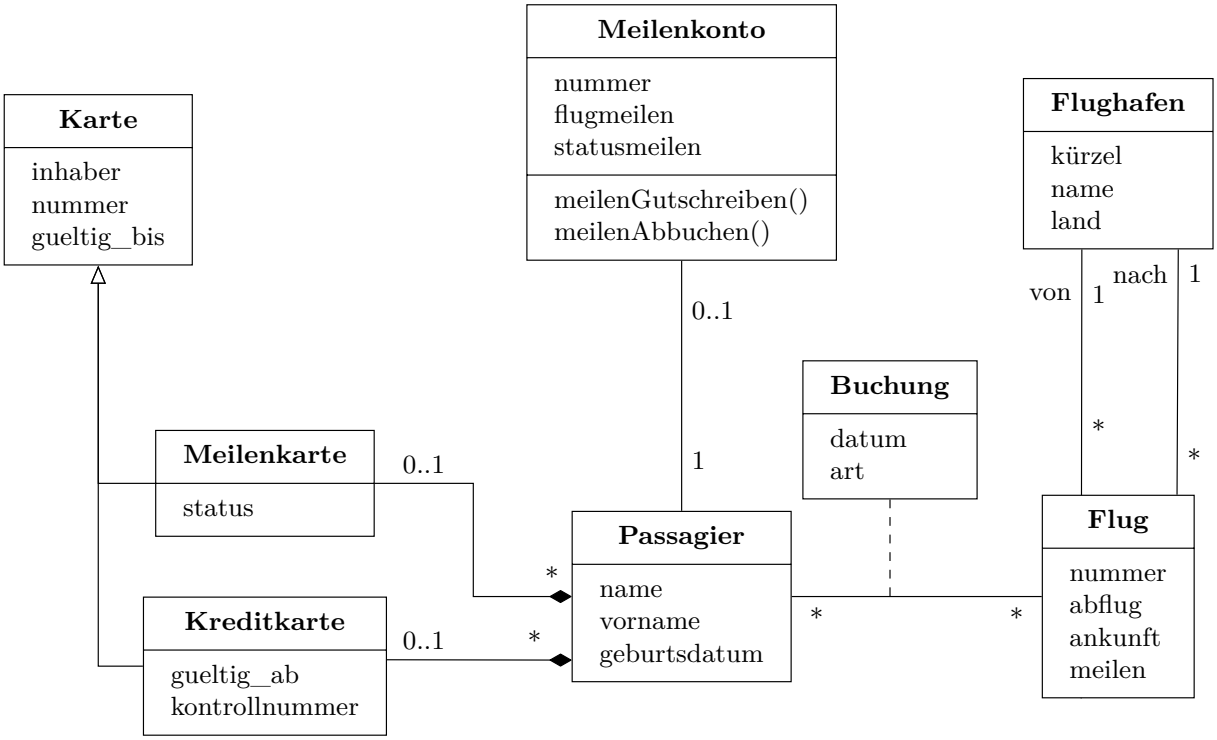
30

- alle Attribute und Operationen der allgemeinen Klasse werden auf die speziellen Klassen vererbt
- das gleiche gilt für Assozationen und Aggregationen
- Meilenkarte und Kreditkarte erben die Attribute aus Karte: `inhaber, nummer` und `gueltig_bis`



Besipiel gesamt

31



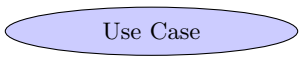
5.3 Use-Case-Diagramme

UML-Use-Case-Diagramme (I)

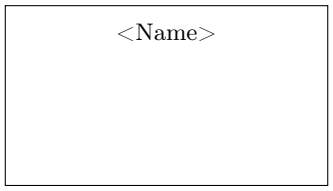
32

- Leicht verständliche Modelle zur Dokumentation von
  - Funktionalitäten des betrachteten Systems
  - deren Beziehungen
  - Beziehungen des Systems zu dessen Umgebung

Use-Case



- Werden durch Ovale dargestellt, die den Namen des Use Case enthalten Alternativ: Name unter dem Use Case



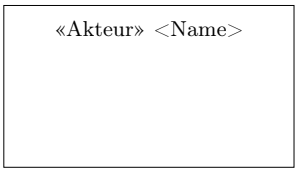
Systemgrenzen:

- Separieren die Teile des Use Cases, die zum System gehören von den außerhalb der Systemgrenzen liegenden Teilen
- Optional: Angabe des Systemnamens an der Systemgrenze

UML-Use-Case-Diagramme (II)

Akteure:

- liegen außerhalb der Systemgrenzen
- stehen für Personen oder Systeme, die mit betrachtetem System interagieren
- Werden durch Rechteck mit dem Zusatz «actor» dargestellt
- Wenn der Akteur eine Person ist, kann dies durch Strichmännchen angezeigt werden
- Ist der Akteur ein System, Rechteck oder Strichmännchen mit Zusatz «system»

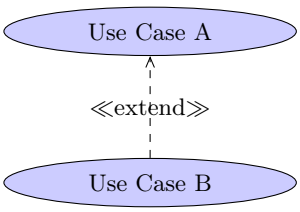


UML-Use-Case-Diagramme (III)

34

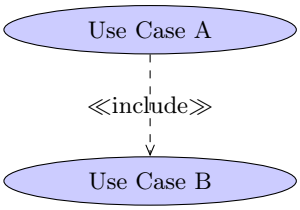
Extend-Beziehung:

- Use Case A erweitert an einem definierten Punkt (Extension Point) Use Case B
- Diese Erweiterung ist abhängig vom Eintreten einer definierten Bedingung



Include-Beziehung:

- Use Case A inkludiert ohne Bedingungen die in Use Case B definierte Interaktionsfolge

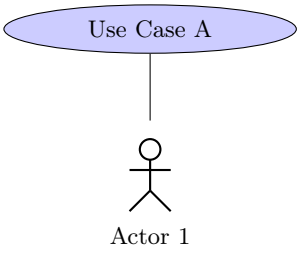


UML-Use-Case-Diagramme (IV)

35

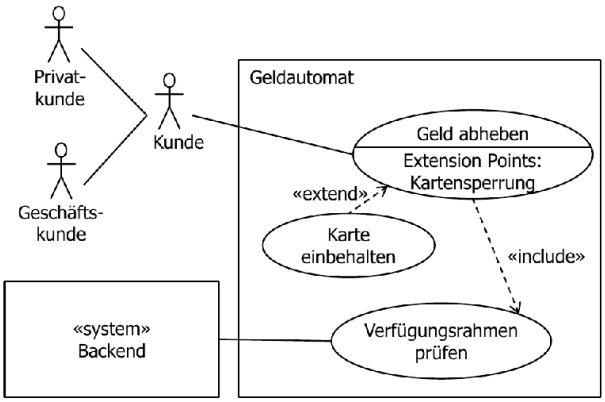
Beziehungen zwischen Akteuren und Use Cases:

- Stellt die während der Ausführung eines Use Cases stattfindende Kommunikation mit einem oder mehreren Akteuren in der Umgebung dar



Beispiel: Geldautomat

36



include: Der Use Case „Geld abheben“ beinhaltet die im Use Case „Verfügungsrahmen prüfen“

definierten Interaktionsschritte

extend: Die im Use Case „Karte einbehalten“ enthaltenen Interaktionsschritte erweitern bei Eintreten bestimmter Bedingungen den Use Case „Geld abheben“ am Extension Point „Kartensperrung“

Generalisierungsbeziehung: spezialisierte Use Cases bzw. Akteure erben Eigenschaften des generalisierenden Use Cases bzw. Akteurs

Literatur

Fachbücher

Balzert, H. (2004). *Lehrbuch Grundlagen der Informatik. Konzepte und Notationen in UML 2, Java 5, C# und C++, Algorithmik und Software-Technik, Anwendungen.* Spektrum Akademischer Verlag. ISBN: 9783827414106.

Balzert, H. (2010). *UML 2 kompakt: mit Checklisten.* IT kompakt. Spektrum Akademischer Verlag. ISBN: 9783827425065.

Rupp, C., S. Queins und S.-G. für Innovatives Software-Engineering (2012). *UML 2 glasklar: Praxiswissen für die UML-Modellierung.* Hanser. ISBN: 9783446430570.

Seemann, J. und W. von Gudenberg (2006). *Softwareentwurf mit UML 2. Objektorientierte Modellierung mit Beispielen in Java.* 2. Aufl. Berlin: Springer. ISBN: 978-3-540-30949-9.