Objektorientierte Modellierung

KLASSENDIAGRAMM

- Klasse = Typebene zum Beschreiben mehrerer Objekte der selben Struktur
- Objekt = konkrete Ausprägung einer Klasse
- Instanz = Objekt
- Klassendiagramm = beschreibt den strukturellen Aspekt eines Systems auf Typebene in Form von Klassen und Beziehungen

Notation von Klassen

- **Klasse = Viereck, mehrere Abschnitte**
  - 1. Abschnitt: Name, abstrakt, etc.
  - 2. Abschnitt: Attribute
  - 3. Abschnitt: Operationen
  - nach oben hin offen z.B. weitere Constraints (selten genutzt)
  - Abschnitte können auch in anderer Reihenfolge erscheinen
- **Attribute:**
  - Sichtbarkeitsoperator (+ public, - private, # protected, ~ package)
  - Name, Doppelpunkt
  - Datentyp
- **Eigenschaften von Attributen:**
  - „/“ attributname = Attribut kann berehnet werden, = abgelietetes Attrbut
  - {optional} Nullwert erlaubt
  - [n..m] Multiplizität (bei Attributen eher nicht verwenden!)
  - unterstrichenes Attribut: Klassenattribut (für alle Instanzen der Klasse gleich!)
  - <<key>> Object-Identifier, nicht Teil der Attribute/ UML
  - sollten nicht Klassenwertig sein = Attribut hat selbst wieder Atrribute/Struktur
  - sollten nicht Mehrwertig sein = Multiplizität > 1 → Verzichten auf Multiplizitätsangabe
- **Operationen:**
  - Sichtbarkeitsoperator
  - Name
  - Input-Parameter (Datentyp!) der Operation in Klammern, Doppelpunkt
  - Output-Parameter

Zugriffs-modifikator	Mitglieder der eigenen **KLASSE** können zugreifen	Mitglieder einer **UNTERKLASSE** der eigenen Klasse können zugreifen	Mitglieder von **KLIENTEN** können zugreifen
public +	ja	ja	ja
protected #	ja	ja	nein
private -	ja	nein	nein

Abbildung 1: Scope von Sichtbarkeitsoperatoren

Assoziationen – Beziehungen

- **Darstellung:**
  - Linie zwischen Klassen
  - Name
  - Leserichtungspfeil
  - Navigationsrichtung
  - Multiplizität

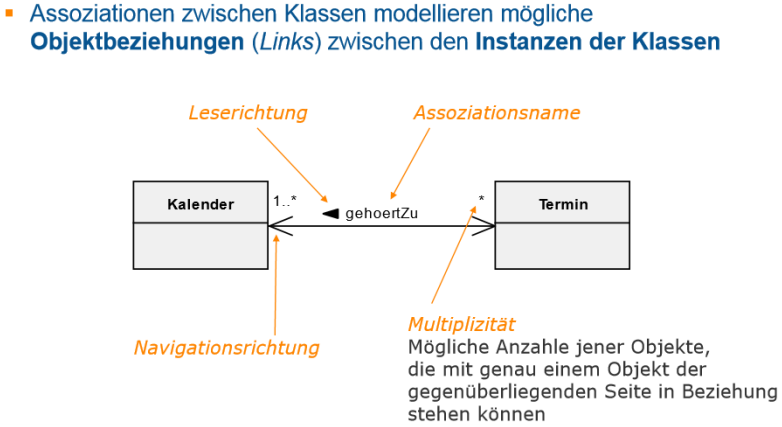


Abbildung 2: Darstellung von Beziehungen

- **Navigationsrichtung:**
  - gerichtete Kante: gibt an, in welche Richtung die Navigation von einem Ojekt zu seinem Partnerobjekt erfolgen kann
  - nicht-navigierbares Assoziationsende: „X“ am Assoziationsende
  - ungerichtete Kante: keine Angabe über Navigationsmöglichkeiten, undefined
  - „Habe ich einen Kalender, so kann ich auf dessen Termine zugreifen.“
  - „Habe ich einen Termin, so kann ich feststellen, in welchen Kalender dieser eingetragen ist.“
  - Best Practice: ungerichtete Kanten = in beide Richtungen navigierbar
  - Best Practice: auf „X“ wird verzichtet, Pfeil nur in navigierbare Richtung
- Assoziation kann auch als Attribut angeführt werden (irrelevant)
- Klasse kann auch mit sich selbst in Beziehung stehen (unäre Beziehung)
- **Multiplizität bei Assoziationen:**
  - Defaultwert = 1
  - Multiplizität befindet sich immer auf der gegenüberliegenden Seite der Assoziation (Partnerklasse)
  - immer vom Maximum ausgehen
- **Rollen bei Assoziationen:**
  - Rollen können von den einzelnen Objekten in den Assoziationen gespielt werden
  - hat Sichtbarkeitsoperator

genau 1:	1
>= 0:	* *oder* 0..*
0 oder 1:	0..1 *oder* 0,1
fixe Anzahl (z.B. 3):	3
Bereich (z.B. >= 3):	3..*
Bereich (z.B. 3 - 6):	3..6
Aufzählung:	3,6,7,8,9 *oder* 3, 6..9

Abbildung 3: Multiplizitäten

- **Exklusive Assoziationen:**
  - XOR-Beziehung
  - für ein bestimmtes Objekt kann zu einem bestimmten Zeitpunkt nur eine von verschiedenen möglichen Beziehungen instanziert werden

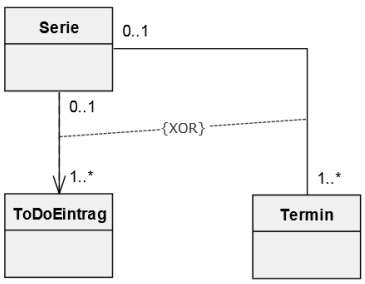


Abbildung 4: XOR-Assoziation

- **Asszoziationsklassen:**
  - Merkmale der **Beziehung** (nicht der Klassen!) werden festgehalten
  - bei n:m Beziehungen mit Attributen notwendig
  - bei 1:1 und 1:n Beziehungen auflösbar → Attribute der Assoziationsklasse auf der n-Seite hinzufügen
  - Attribute, die von beiden Klassen abhängen
  - könnte auch als eigene Klasse mit normalen Assoziationen zu den anderen Klassen modelliert werden → nicht das selbe!

- **n-äre Assoziationen:**
  - Beziehungen zwischen mehr als zwei Klassen
  - keine Angaben über Navigationsrichtungen mehr
  - Multiplizitäten: „Ein Spieler spielt in einer Saison bei wievielen Vereinen?“ etc.
  - Multiplizitäten: zwei Klassen zusammennehmen und die dritte Klasse erfragen
  - kann i.A. nicht durch mehrere binäre Assoziationen modelliert werden → Constraints könnten verloren gehen

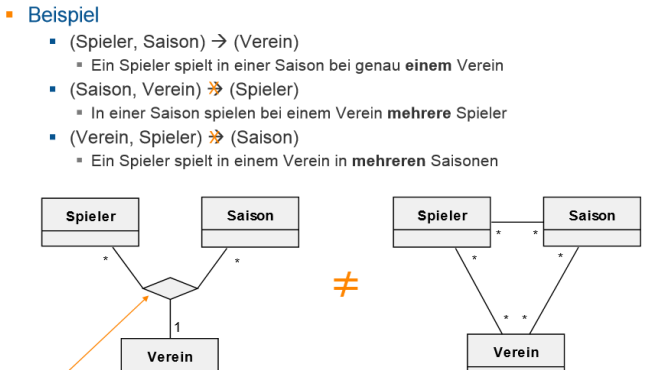


Abbildung 5: ternäre Assoziation

(hier: ternär)

- **Ordnung und Eindeutigkeit von Assoziationen:**
  - **ordered**: es besteht eine Ordnung der Elemente
  - **unordered**: default, keine Rangfolge wird angenommen
  - **unique**: default, z.B. ein Queueitem darf in einer Queue nur ein Mal vorkommen
  - **nonunique**: z.B. Queueitem darf öfters in der Queue vorkommen
  - Frage bei unique/nonunique: Darf eine Instanz einer Klasse mehrmals mit einer Instanz der anderen Klasse in Beziehung treten?
    - ja: nonunique
    - nein: unique

Eindeutigkeit	Ordnung	Kombination	Beschreibung
unique	*unordered*	set	Menge (Standardwert)
unique	*ordered*	*orderedSet*	Geordnete Menge
nonunique	*unordered*	*bag*	Multimenge, d.h. Menge mit Duplikaten
nonunique	*ordered*	*sequence*	Geordnete Menge mit Duplikaten (Liste)

Abbildung 5: Ordnung und EIndeutigkeit von Assoziationen

- **Qualifizierte Assoziation:**
  - besteht aus einem Attribut, dessen Werte die Objekte der in Beziehung stehenden Klassen aufteilen
  - reduziert Multiplizitäten
  - laut Bsp.: persNr muss je Person eindeutig sein!

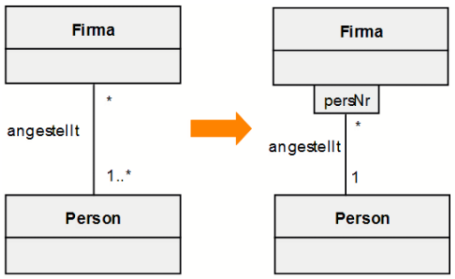


Abbildung 6: Qualifizierte Assoziation

- **Abhängigkeiten:**
  - allgemeine Kopplung zweier Modellelemente
  - → Änderung in einem Element kann Änderung beim Klienten hervorrufen

Aggregation

- spezielle Form der Assoziation mit folgenden beiden Eigenschaften:
  - **Transitivität: A Teil von B, B Teil von C → A Teil von C**
    - z.B. Blatt Teil von Ast, Ast Teil von Baum → Blatt Teil von Baum
  - **Anti-Symmetrie: A Teil von B → B nicht Teil von A**
    - z.B. Buchstabe Teil von Wort, Wort **nicht** Teil von Buchstabe
- **Schwache Aggregation (shared aggregation):**

- „syntactic sugar“
- weiße Raute auf der Seite des Ganzen
- schwache Zugehörigkeit der Teile → Teile sind unabhängig von ihrem Ganzen, können selbstständig bestehen

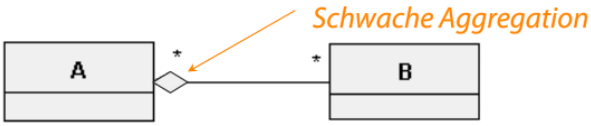


Abbildung 7: Schwache Aggregation

- Multiplizität des aggregierenden Endes der Beziehung (Raute) kann > 1 sein → ein Teil kann in mehreren Ganzen enthalten sein
- eingeschränkte Propagierungssemantik (= Wenn ich das Ganze lösche, werden die Teile **nicht** gelöscht)

- **Starke Aggregation – Komposition (composite aggregation)**
  - schwarze Raute auf der Seite des Ganzen
  - ein bestimmter Teil darf zu einem bestimmten Zeitpunkt in maximal einem zusammengesetzten Objekt enthalten sein
  - Multiplizität des aggregierenden Endes der Beziehung darf (maximal) 1 sein

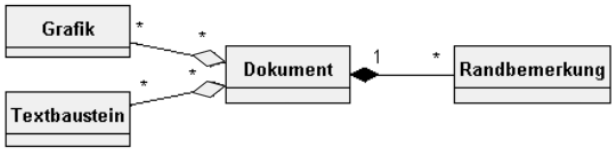


Abbildung 8: Starke und Schwache Aggregation

  - Teile sind vom zusammengesetzten Objekt **abhängig**
  - Propagierungssemantik (= Wenn ich das Ganze lösche, werden die Teile **auch** gelöscht)

Generalisierung

- nicht-ausgefüllte Pfeilspitze bei Superklasse
- **Taxonomische Beziehung** zwischen einer spezialisierten Klasse und einer allgemeineren Klasse
  - spezialisierte Klasse **erbt** die Eigenschaften der allgemeineren Klasse
  - spezialisierte Klasse kann weitere Eigenschafter hinzufügen
  - Instanz der Subklasse kann überall dort verwendet werden, wo eine Instanz der Superklasse erlaubt ist
- stellt Hierarchie von „is-a“ - Beziehungen her (**Transitivität**)
- **Abstrakte Klassen:**
  - {abstract} Klassenname **oder** *Klassenname* (kursiv schreiben)
  - darf keine direkten Instanzen haben!
  - nur in Generalisierungshierarchien mit Instanzen der abstrakten Klasse sinnvoll
  - gemeinsame Merkmale einer Reihe von Subklassen werden hervorgehoben

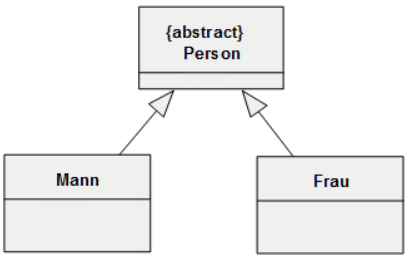


Abbildung 9: Abstrakte Klassen

- **Mehrfachvererbung:**
  - Subklasse kann mehr als eine Superklasse haben
  - nicht immer sinnvoll (Programmiersprache)
- **Eigenschaften der Generalisierungshierarchien:**
  - **unvollständig/vollständig:**
    - vollständig: jede Instanz der Superklasse muss auch eine Instanz der Subklasse sein
    - unvollständig: Obiges gilt nicht
  - **überlappend/disjunkt:**
    - überlappend: ein Objekt kann Instanz von mehr als einer Subklasse sein
    - disjunkt: ein Objekt kann Instanz von einer Subklasse sein, aber nicht von mehr als einer

- **default:** unvollständig, disjunkt

Abbildung 10: Eigenschaften von Generalisierungshierarchien

- **Redefinition von geerbten Merkmalen:**
  - geerbte Merkmale können in Subklasse redefiniert werden
    - {redefines} <feature>
  - redefinierbare Merkmale (= Spezifizieren von Merkmalen):
    - Attribute
    - navigierbare Assoziationsenden
    - Operationen
  - {leaf} Operationsname → Redefinition der Operation wird verhindert
  - Redefiniertes Merkmal muss konsistent zum ursprünglichen Merkmal sein:
    - Typ muss gleich oder ein Subtyp des usprünglichen Typs sein
    - Intervall der Multiplizität muss in jenem des ursprünglichen Attributs enthalten sein
    - Signatur einer Operation muss die gleiche Anzahl an Parametern aufweisen

Datentypen

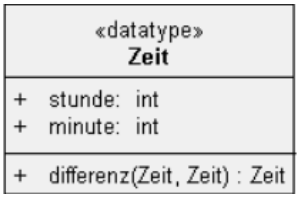


Abbildung 11: Datentyp

- können auch selbst Attribute und Operationen haben
- Instanzen eines Datentyps haben keine Identität
- Werte: Instanzen eines Datentyps
- Schlüsselwort: <<datatype>> Name
- primtive Datentypen:
  - Datentypen ohne innere Struktur
  - primitive Datentypen von UML:
    - boolean
    - String
    - Integer
    - UnlimitedNatural
  - könen auch selbst definiert werden (Schlüsselwort <<primitive>>)
  - bei plattformspezifischer Programmierung die Datentypen der jeweiligen Programmiersprache verwenden
- Aufzählungstypen:
  - Festlegung der Auspärgungselemente per Aufzählung
  - Notation: Schlüsselwort <<enumeration>> Name
  - können auch Attribute oder Operationen haben
  - mögliche Ausprägungen werden durch benutzerdefinierte Literale angegeben

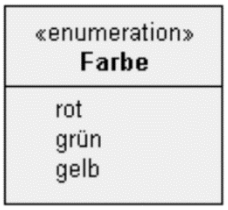


Abbildung 12: Enumeration

OBJEKTDIAGRAMM

- beschreibt den strukturellen Aspekt eines Systems auf Instanzebene
- enthält nur Objekte und Links, keine Klassen!
- Momentaufnahme des Systems = konkretes Szenario
- Ausprägung eines Klassendiagramms
- kann z.B. Fehler in den Multiplizitäten aufzeigen
- kann z.B. aufzeigen, dass weitere Constraints eingeführt werden müssen
- Instanz einer Klasse = Objekt
- Instanz einer Assoziation = Link
- Instanz eines Datentyps = Wert
- Objektdiagramm ist nie vollständig (zu aufwändig), dienst nur zur Veranschaulichung

Notation für Objekte

- Objektname:Klassenname
- Attribute erhalten keine Datentypen, sondern konkrete Ausprägungen und müssen den Datentypen der Klasse entsprechen
- es gibt keine Multiplizitäten mehr

PAKETDIAGRAMM

- in einem großen UML-Diagramm hat man sehr viele Klassen → diese sollen nicht alle in ein **Paket** geworfen werden
- → verschiedene Klassendiagramme in einem Dachdiagramm (Vgl. Namespace)
- z.B. zwei ähnliche Klassen, die sich nur in Details unterscheiden
- ein Paket kann einem eindeutigen Namespace zugeordnet werden (Vgl. Ordnerstruktur)
- Sichtbarkeitsoperatoren wichtig für die Zugrifffreigaben innerhalb und außerhalb eines Namespace
- beliebig tiefe Schachtelung möglich
- Import von Elementen: Sichtbarkeit muss public sein
    - beim Import von Elementen erfolgt auch eine Veränderung des Namespace
        - Namen des importieren Pakets werden in den Namespace des Klienten geladen → Namespace des Klienten wird geändert
        - Qualifizierte Namen sind nicht mehr nötig
    - es gilt **Transitivität!**

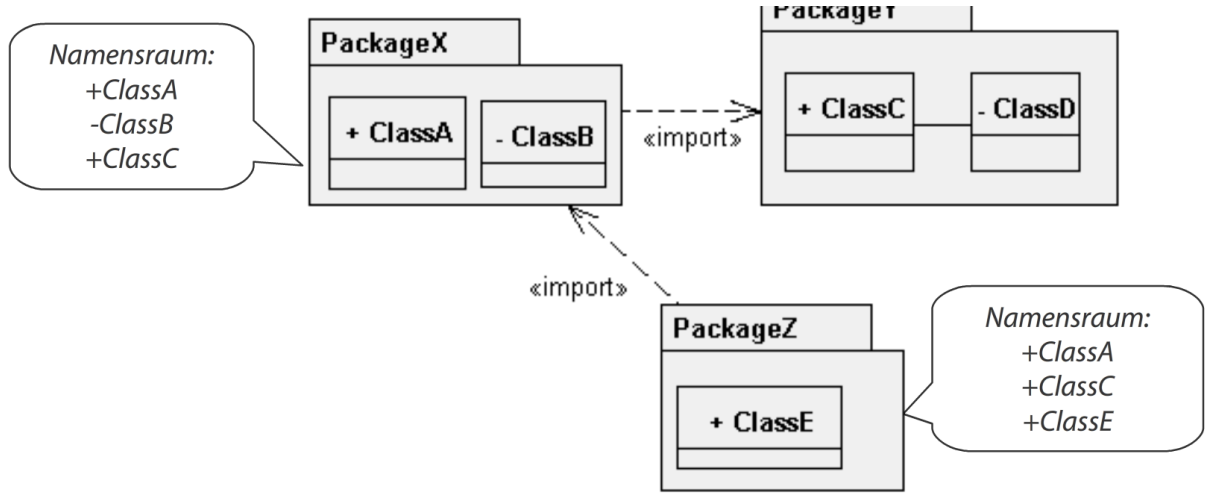


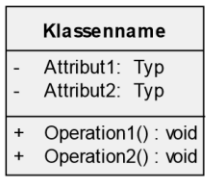
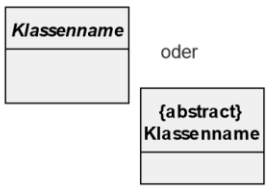
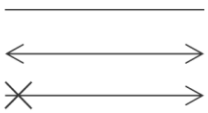
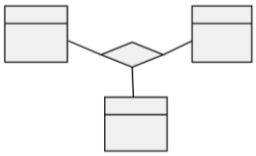
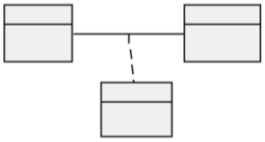
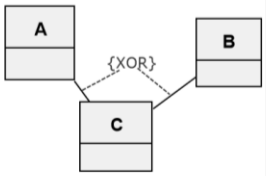
Abbildung 13: Tranistivität bei Paketimport

- Access von Elementen:
    - **nicht transitiv!**
    - Änderung der Sichtbarkeit der importierten Elemente auf private

ÜBERSETZUNG NACH JAVA

- Klassen in UML werden auch nach Klassen in Java übersetzt
- Attribute und Operationen in UML werden in Java als Instanzvariablen und Methoden dargestellt
- Klassenvariablen und -operationen (unterstrichen) werden mit dem Schlüsselwort **static** versehen
- Einfachvererbung wird in Java direkt unterstützt (**extends**)
- Assoziationsklassen können mit Hashtables dargestellt werden
- Assoziationen werden mit Hilfe von Variablen ausgedrückt:
    - 1:1 = es reicht jeweils eine Variable vom Typ der verbundenen Klasse
    - 1:N = Array, ArrayList, o.Ä.
    - Angabe von Navigationsrichtung vereinfacht i.A. die Implementierung
    - Operationen zur Verwaltung der Assoziationen müsen eingefügt werden

ZUSAMMENFASSUNG

Name	Syntax	Beschreibung
Klasse		Beschreibung der Struktur und des Verhaltens einer Menge von Objekten
abstrakte Klasse		Klasse, die nicht instanziert werden kann
Assoziation		Beziehung zwischen Klassen: keine Angabe über Nav.-r.; mit Navigationsrichtung; in eine Richtung nicht navigierbar.
n-äre Assoziation		Beziehung zwischen n Klassen
Assoziations-klasse		nähere Beschreibung einer Assoziation
xor-Beziehung		Entweder steht Klasse A oder Klasse B in Beziehung zu C, nicht aber beide
schwache Aggregation		"Teil-Ganzes"-Beziehung
starke Aggregation = Komposition		exklusive "Teil-Ganzes"-Beziehung
Generalisierung		Vererbungsbeziehung zwischen Klassen

CHECKLISTE

- Sie haben diese Lektion verstanden, wenn Sie wissen …

- was der Unterschied zwischen einer Klasse und einem Objekt ist.
- was der Unterschied zwischen abstrakten und konkreten Klassen ist.
- was Attribute und Operationen einer Klasse sind und welche Eigenschaften diese haben können.
- dass Klassen durch Assoziationen miteinander verbunden werden.
- warum Assoziationen mit einer Multiplizität versehen werden.
- was Generalisierung ist und wann diese eingesetzt wird.
- was der Unterschied zwischen starker und schwacher Aggregation ist.
- wie Sie aus einer textuellen Angabe ein UML-Klassendiagramm erstellen.
- wie Sie die wichtigsten Elemente des Klassendiagramms in Java umsetzen können.
- wozu ein Meta-Modell benötigt wird.
- wie der Zusammenhang zwischen Klassen- und Objektdiagramm ist.
- was Interfaces sind.
- wozu Pakete eingesetzt werden.
- wie Abhängigkeiten in UML dargestellt werden können.