Einführung in die Programmierung für NF

UML

UML

UML

Unified Modeling Language

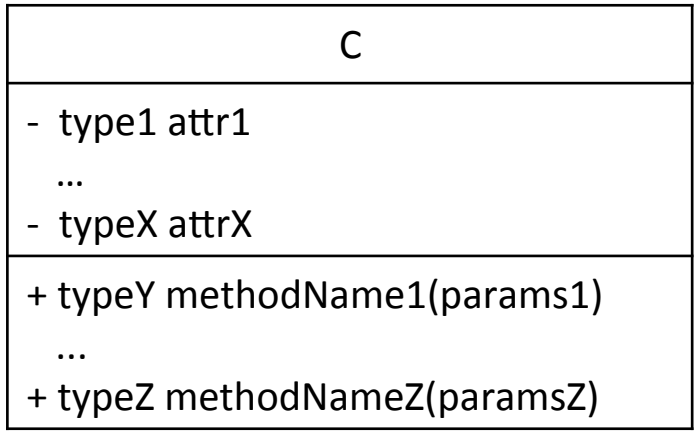
- Modellierungssprache für Objekt-orientierte Programmierung
- Aktueller Standard in Industrie und Forschung
- Wird von OMG (Object Management Group) weiterentwickelt
- UML Diagramme werden zur abstrakten Repräsentation von Klassen verwendet

Einfache Klassen in UML

- Eine Klasse C wird folgendermaßen in UML repräsentiert

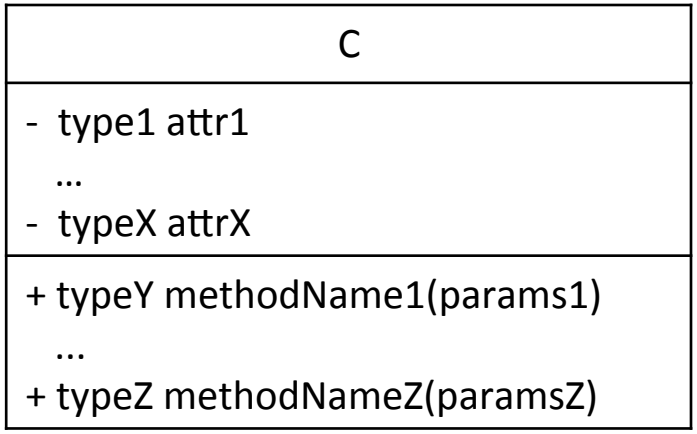
"-" für private

"+" für public



- Die Konstruktoren werden nicht im Klassendiagramm aufgeführt
- Methodenrümpfe erscheinen nicht im UML-Diagramm

Implementierung einer UML-Klasse



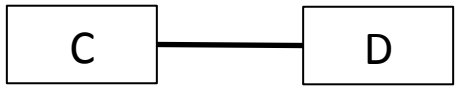
- Für jede Methode einer Klasse muss eine Implementierung (in Java) angegeben werden
- Für alle Methoden müssen Implementierungen durch Methodenrümpfe angegeben werden
- Die Konstruktoren müssen implementiert werden

BEZIEHUNGEN ZWISCHEN KLASSEN

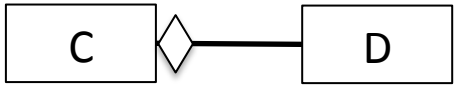
Assoziation und Aggregation

Assoziation ist eine Beziehung zwischen zwei oder mehr Klassen

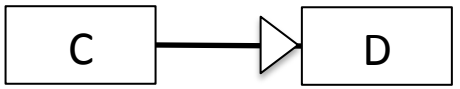
- Assoziation: Die Klassen C und D stehen in Beziehung



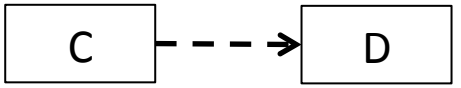
- Aggregation: Jedes Objekt von C enthält Objekte von D



- Vererbung: Die Klasse C ist Erbe der Klasse D

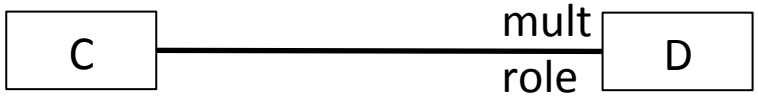


- Abhängigkeit: Die Klasse C verwendet Elemente der Klasse D (i.a. Methoden)



Assoziation

- Die Assoziation ist die allgemeinste Art einer Beziehung

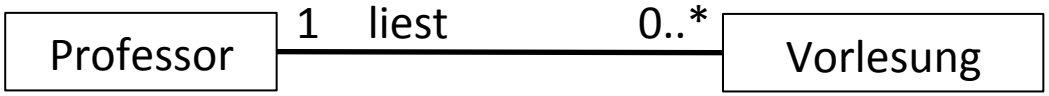


- Das Diagramm drückt aus, dass jedes Objekt O von C mit "mult"-vielen Objekten von D in Beziehung steht, die die Rolle "role" für O spielen
- "role" ist ein Name
- "mult" ist eine natürliche Zahl, ein Stern "*" für beliebig viele Objekte oder ein Intervall der Form a..b

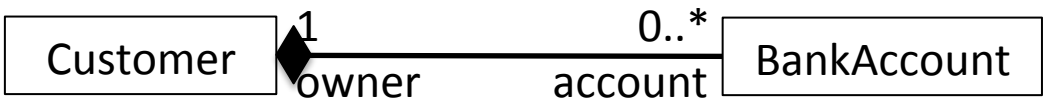
Assoziation und Aggregation

Beispiele

- Assoziation



- Aggregation
  - Starke Aggregation (Komposition)



    - Jeder Bankkunde besitzt 0 oder mehrere Konten, mit der Rolle "account"
    - Jedes Konto hat genau einen Besitzer ("owner")
    - Die Lebensdauer eines Bankkontos ist durch die des Besitzers beschränkt
  - Schwache Aggregation

    - Eine "Teile-Ganzes" Relation, die Lebensdauer der Teile hängt aber **nicht** von der Lebensdauer des Ganzen ab
    - Es entsteht ein gerichteter azyklischer Graph (Wenn B Teil von A, ist A nicht Teil von B

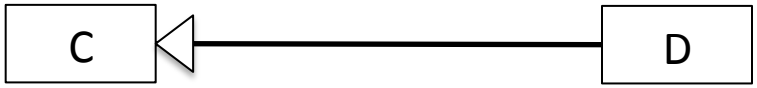
Implementierung

- Assoziation
  - Assoziationen werden durch Attribute repräsentiert
  - Jede Rolle "role" vom Typ D wird als Attribut von C implementiert
  - Ist die Multiplizität 0,1 oder 0..1 erhält man ein Attribut D "role"
  - Ist die Multiplizität >1 oder * verwendet man ein Array oder eine ArrayList von D-Objekten

- Aggregation
  - Die Komponenten werden durch den Konstruktor des Aggregats (C) erzeugt

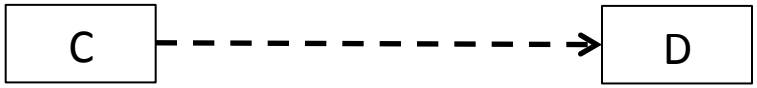
    Z.B.
```
class Circle{
      private double radius;
      private Point center;

      public Circle(double rad, double x, double y){
           radius = rad;
           center = new Point(x,y);
      } ...
```

Vererbung



- Jedes Attribut von C ist automatisch Attribut von D - kann aber evtl. nicht direkt zugreifbar sein (private)
- Jede Methode von C ist automatisch eine Methode von D - eine Methode von D kann jedoch nicht von C aufgerufen werden
- Auf die Methode der Superklasse (hier C) wird mit "super" zugegriffen
- In der Subklasse D können Methoden redefiniert werden. Diese müssen im UML-Diagramm von D explizit genannt werden

Abhängigkeit



- D verwendet Methoden der Klasse C
- Schnittstellen (Interfaces) werden mit dieser Methode in UML beschrieben

Vielen Dank für Ihre Aufmerksamkeit