Universität
Bremen

Universität Bremen
AG Softwaretechnik

**Fachbereich 03**
Mathematik und Informatik

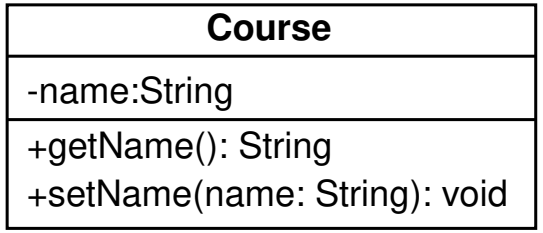
# Software-Projekt

## Datenmodell
## Tipps & Hinweise

Karsten Hölscher

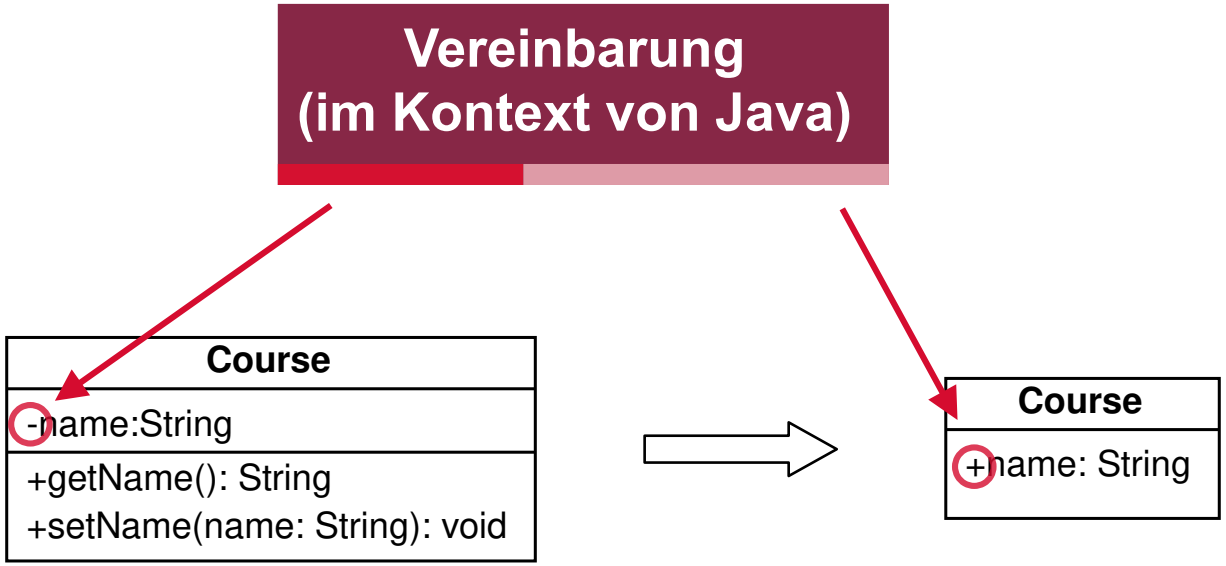
# Datenmodell

- Datenklassen tragen Daten
- Datenklassen enthalten **keine** Methoden
  - Ausnahmen
    - Konstruktor(en)
    - getter-/setter-Methoden
    - equals
    - hashCode



# Java

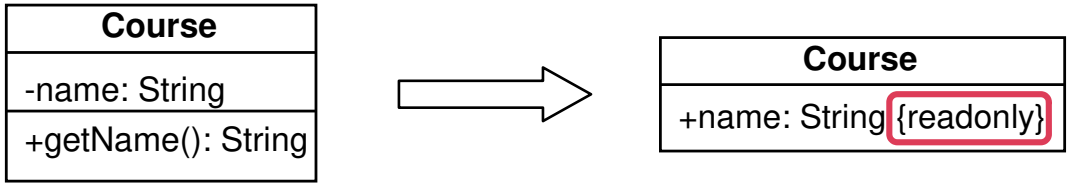
- Attribute immer private
- getter-/setter-Methoden
  - public (meistens)
  - protected
  - package



# Java

- Attribute immer private
- getter-/setter-Methoden
  - public (meistens)
  - protected
  - package

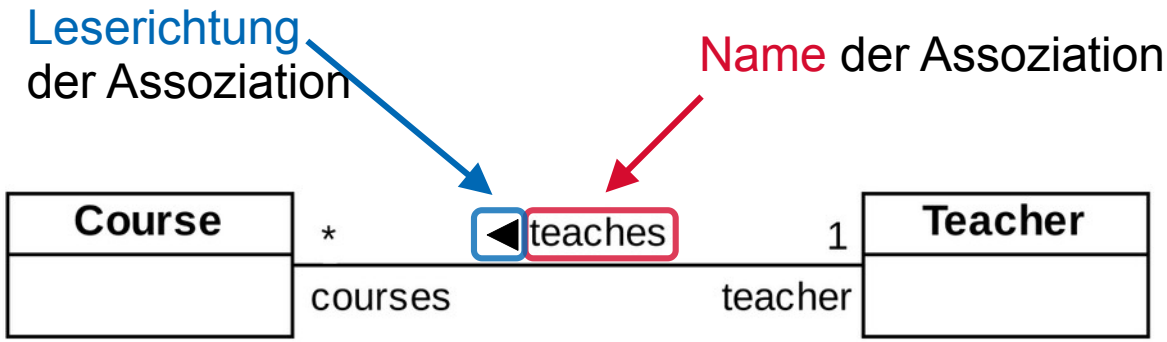
**falls kein setter**



# Assoziationen

**Name und Leserichtung für Implementierung irrelevant**

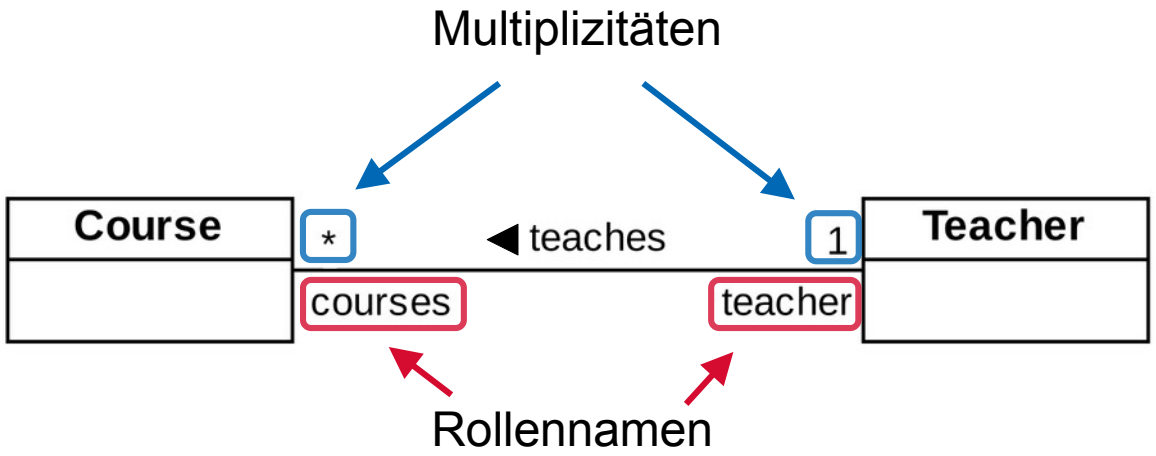
- Name und Leserichtung
  - nur wichtig für „Lesbarkeit“ und „über das Diagramm sprechen“




Lesen der Relation: „Teacher teaches courses“

# Assoziationen

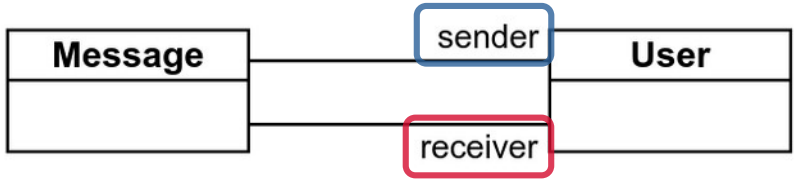
- Rollen und Multiplizitäten




Lesen der Relation: „One teacher teaches any number of courses“

# Assoziationen

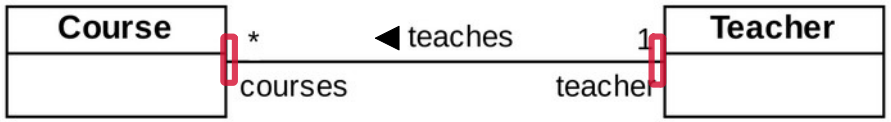
- Rollennamen
  - verwendbar als Attribut-Bezeichner




```
public class Message {

    private User sender;

    private User receiver;

}
```

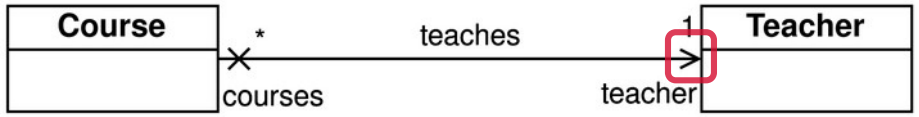
# Assoziationen

- Navigierbarkeit
  - weder Pfeil noch Kreuz → unspezifiziert
    - wird während Implementierung entschieden



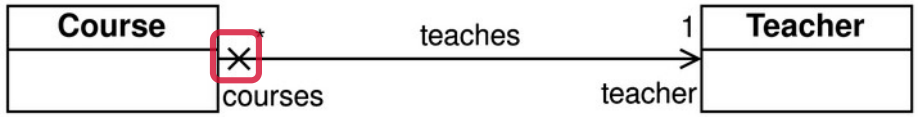
# Assoziationen

- Navigierbarkeit
  - Pfeil → navigierbar
    - Klasse (hier Course) hat ein entsprechendes Attribut



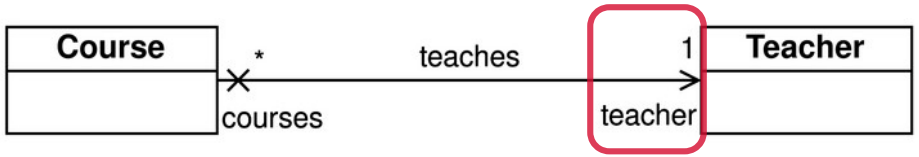
# Assoziationen

- Navigierbarkeit
  - Kreuz → nicht navigierbar
    - Klasse (hier Teacher) hat kein entsprechendes Attribut



# Assoziationen

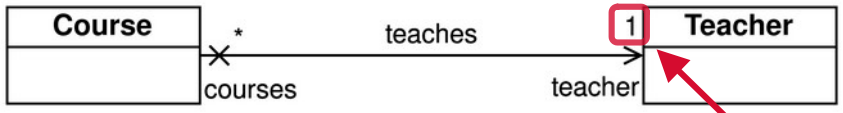
- Navigierbarkeit aus Sicht von Klasse `Course`
  - Pfeil zu `Teacher` → navigierbar
    - Klasse hat ensprechendes Attribut
  - Multiplizität 1
    - Attribut ist eine Objektreferenz vom Typ `Teacher`
  - Rollenname *teacher*
    - Attributname `teacher`




```
public class Course {

    private Teacher teacher;

    ...
```

# Assoziationen

**eine 1 als Multiplizität bedeutet „existenzielle Abhängigkeit“**



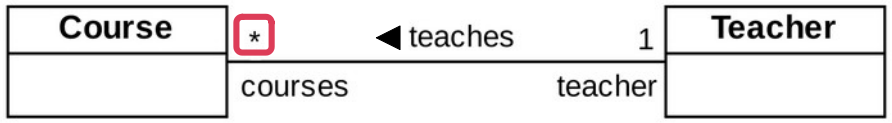

Ein Course-Objekt darf hier wegen der 1 am Teacher-Ende nicht ohne ein zugeordnetes Teacher-Objekt existieren.

Daher ist ein Teacher-Objekt als Parameter im Konstruktor nötig, damit die Multiplizität immer erfüllt ist!

```
public class Course {

    private Teacher teacher;

    public Course(final Teacher pTeacher) {
        // null-Checks etc.
        teacher = pTeacher;
    }

    ...

}
```

# Assoziationen

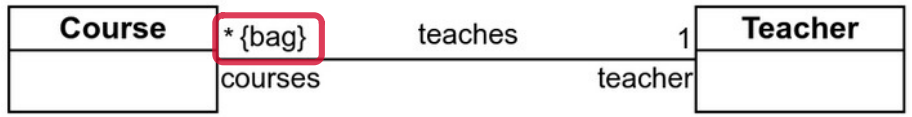
- * bedeutet beliebig viele, aber verschieden → Set




```
import java.util.Set;

public class Teacher {

    private Set<Course> courses;

    ...
```

# Assoziationen

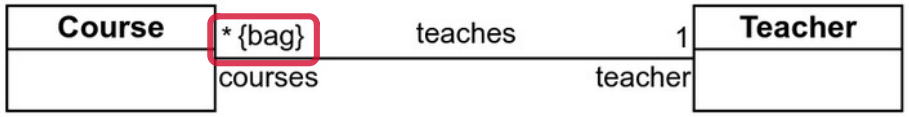
- {bag} → Multimenge → Collection




```
import java.util.Collection;

public class Teacher {

    private Collection<Course> courses;

    ...
```

# Assoziationen

- {ordered} → angeordnet → z. B. Liste




```
import java.util.List;

public class Teacher {

    private List<Course> courses;

    ...
```

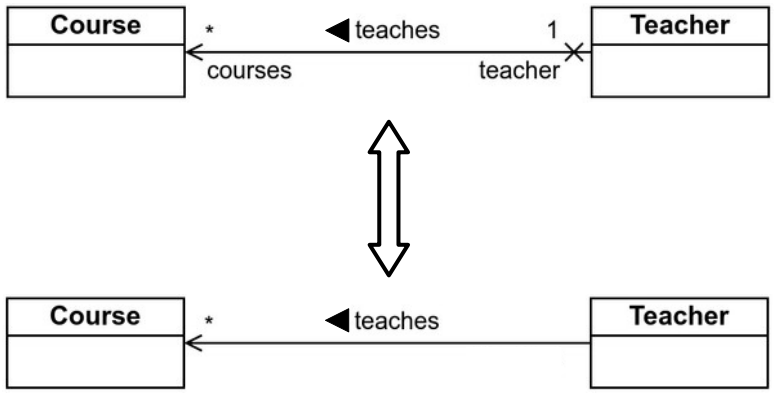
# Beispiele für eigene Konventionen

- erhöhen Lesbar- und Übersichtlichkeit
- Multiplizität
  - wenn keine Angabe → 1
- Rollennamen
  - wenn kein Rollenname → Name der Klasse (kleingeschrieben), ggf. Plural
- Navigierbarkeit
  - wenn kein Pfeil → navigierbar
  - wenn ein Pfeil
    - navigierbar in Pfeilrichtung
    - nicht navigierbar in Gegenrichtung

**Keine allgemeingültigen Konventionen!**
**Daher kurze Beschreibung beim Diagramm.**

# Datenmodell

- ausführlich vs. vereinbarte „Abkürzung“

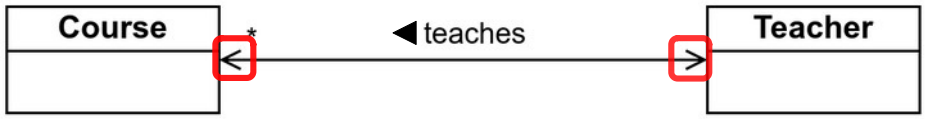



```
public class Course {

    ...

}
```

```
public class Teacher {

    private Set<Course> courses;

    ...
```

# Tipps

- Konsistenz
  - bidirektionale Relationen vermeiden
  - Kreise vermeiden
- idealerweise ist Datenmodell ein Baum




Wenn hier ein Teacher-Objekt zu einem Course hinzugefügt wird, muss die Referenz auf den Teacher im Course gesetzt werden und der Course zum Course-Set des Teachers hinzugefügt werden. Beim Entfernen eines Teachers aus einem Kurs analog. Für einen konsistenten Datenbestand ist es einfacher, möglichst oft unidirektionale Relationen einzusetzen.