universität innsbruck

# UML - Klassendiagramme

Programmiermethodik

Lukas Kaltenbrunner, Simon Priller
Universität Innsbruck

# Modell und Diagramm

- Ein Modell stellt eine Abstraktion eines Realitätsausschnitts dar.
  - Um Informationen verständlicher darzustellen
    - Analog zum Erstellen von Bauplänen von Gebäuden
  - Um essenzielle Systemaspekte aufzuzeigen
  - Zur Kommunikation
    - Mit Projektmitarbeitenden
    - Mit Kundschaft
  - Um komplexe Architekturen darstellen zu können
- Ein Diagramm ist die grafische Repräsentation eines Modells.

# Unified Modeling Language (UML)

- Die Unified Modeling Language (UML) ist eine standardisierte ausdruckstarke Modellierungssprache.
- Mit Hilfe der UML können Softwaresysteme besser entworfen, analysiert und dokumentiert werden.
- Begriff Unified bedeutet
  - Unterstützung des gesamten Entwicklungsprozesses.
  - Unabhängigkeit von Entwicklungswerkzeugen, sowie Programmiersprachen oder auch Anwendungsbereichen.
- Die UML ist aber nicht
  - ein Allheilmittel und vollständig,
  - ein vollständiger Ersatz für eine Textbeschreibung,
  - eine Methode oder Vorgehensmodell.

# UML Diagrammarten

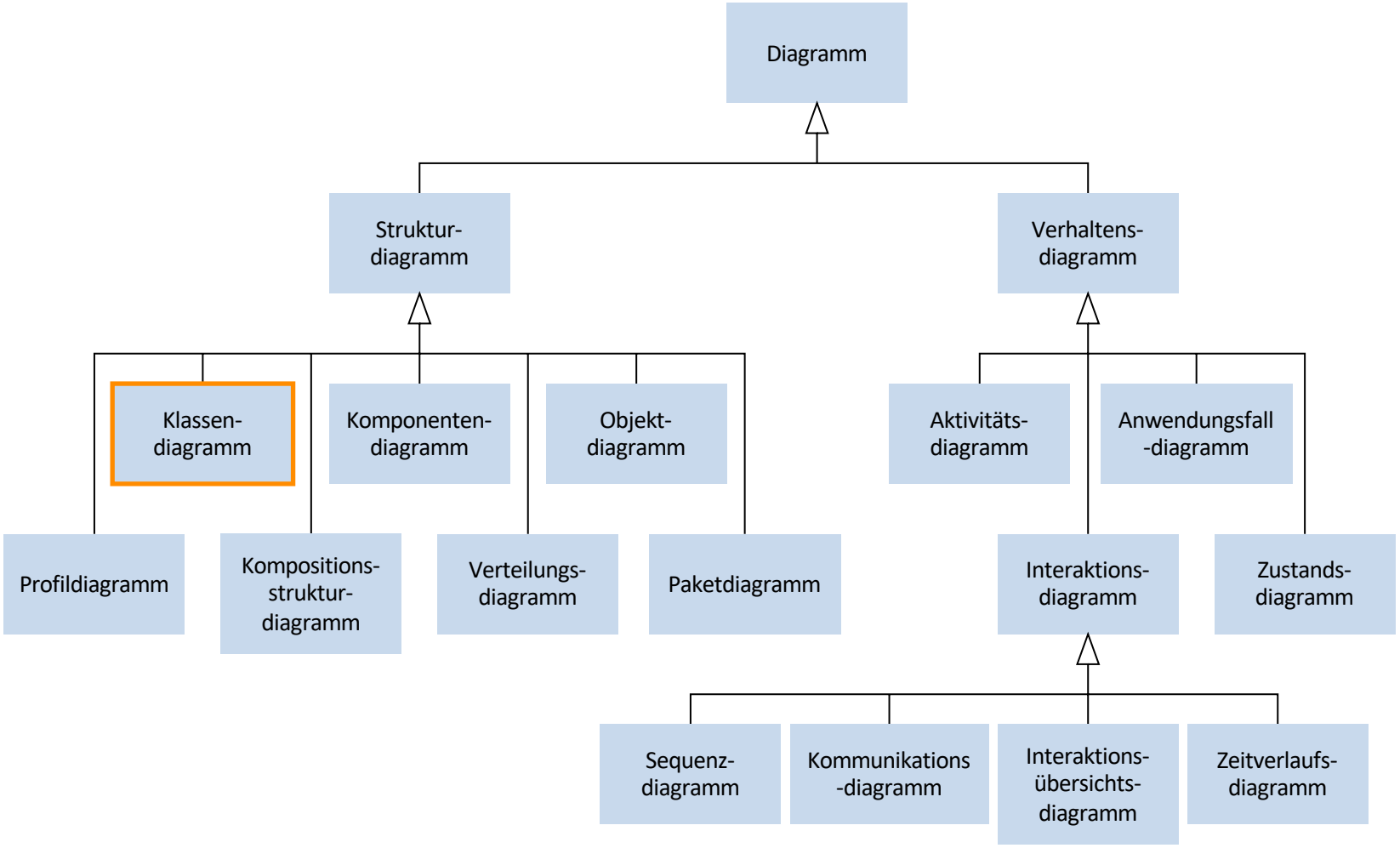


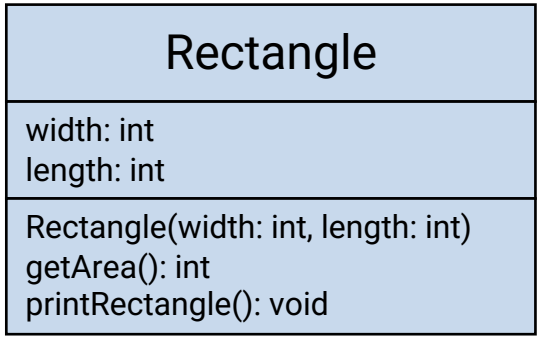

Abbildung angelehnt an „UML @ Classroom“, Seidl et al.

# Notation

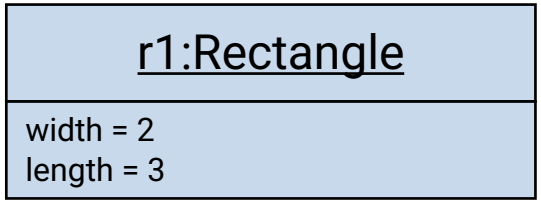
| Sprachkonzept | Notation |
| --- | --- |
| Klasse | Klassenname |
| Klasse mit Abschnitten | Klassenname<br>Attribut 1<br>Attribut 2<br>Operation 1<br>Operation 2 |
| Objekt | objektname<br>objektname:Klassenname |
| Anonymes Objekt | :Klassenname |
| Objekt mit Attributen | objektname<br>Attribut 1 = Wert 1<br>Attribut 2 = Wert 2 |

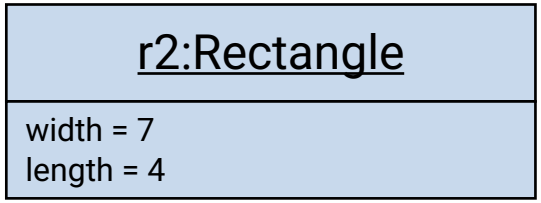
Attribute können auch bei den anderen Notationen für Objekte angegeben werden

# Klassen und Objekte



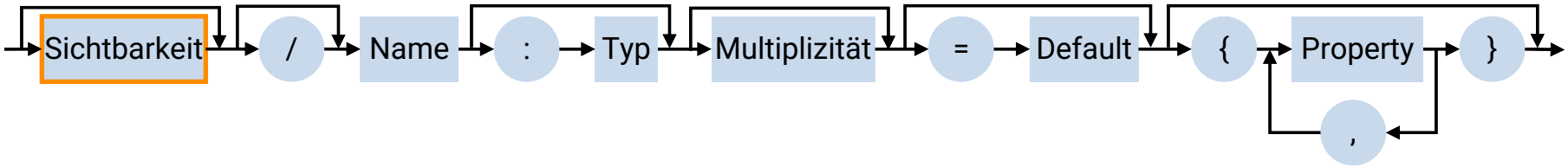

```java
public class Rectangle {

    int width;
    int length;

    Rectangle(int width, int length) {…}
    int getArea() {…}
    void printRectangle() {…}
}
```




```java
…
Rectangle r1 = new Rectangle(2, 3);
Rectangle r2 = new Rectangle(7, 4);
…
```

# Attribute (1)



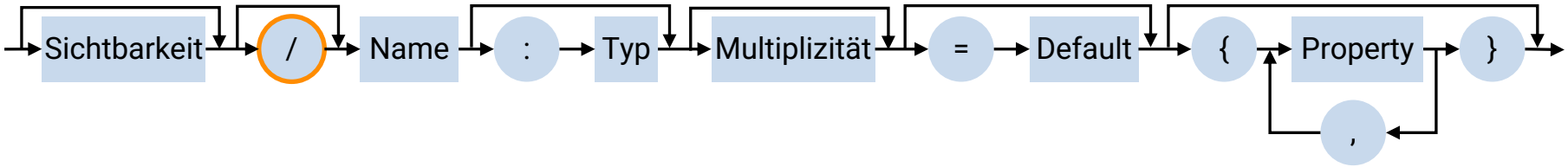

| Name | Symbol | Zugriff auf Attribut |
|---|---|---|
| public | + | Objekte beliebiger Klassen |
| private | - | Nur innerhalb der Klasse |
| protected | # | Objekte derselben Klasse und deren Subklasse |
| package | ~ | Objekte, deren Klassen sich im selben Paket befinden |

**Course**

- name: String {readOnly}
- leader: Person {readOnly}
- participants: Person[1..*] {unique, ordered}
- numberParticipants: int = 1

Abbildung entnommen aus „UML @ Classroom“, Seidl et al.

# Attribute (2)

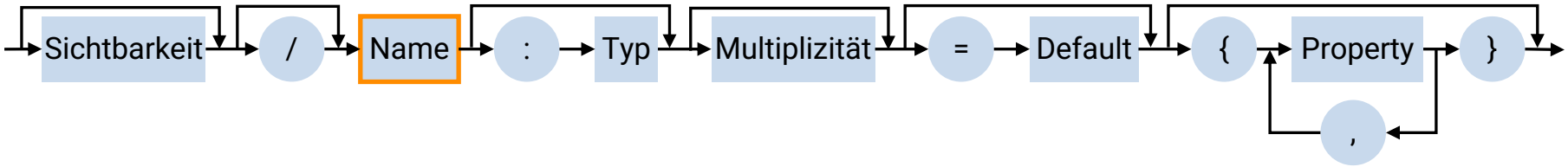



- Kennzeichnung eines abgeleiteten Attributs
- Abgeleitete Attribute
  - Werden aus Werten anderer Attribute berechnet.
  - Benötigen in der Regel keinen dedizierten Speicher im Objekt.
  - Beispiel: Berechnung des Alters anhand des Geburtsdatums.

| Course |
| --- |
| - name: String {readOnly}<br>- leader: Person {readOnly}<br>- participants: Person[1..*] {unique, ordered}<br>- numberParticipants: int = 1 |

Abbildung entnommen aus „UML @ Classroom“, Seidl et al.

# Attribute (3)

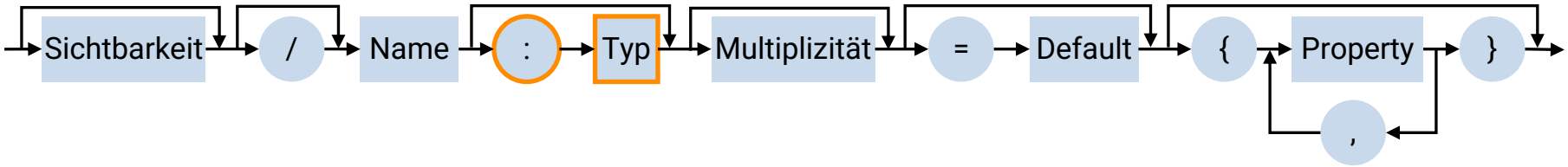



- Bezeichnung des Attributs

| Course |
| --- |
| - name: String {readOnly}<br>- leader: Person {readOnly}<br>- participants: Person[1..*] {unique, ordered}<br>- numberParticipants: int = 1 |

Abbildung entnommen aus „UML @ Classroom“, Seidl et al.

# Attribute (4)

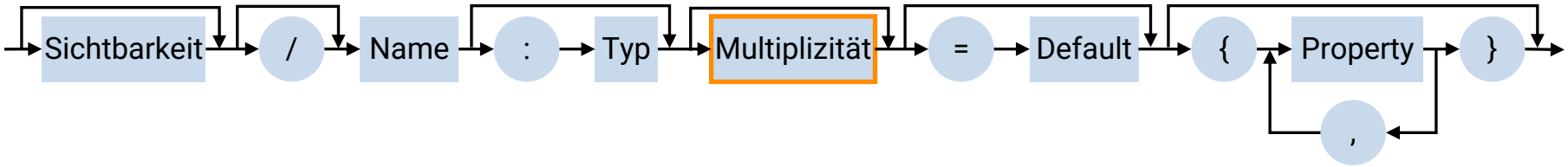



- Angabe des Datentyps

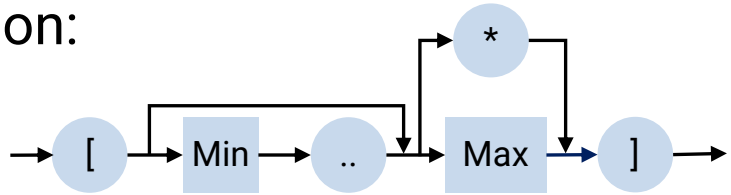
| Course |
| --- |
| - name: String {readOnly}<br>- leader: Person {readOnly}<br>- participants: Person[1..*] {unique, ordered}<br>- numberParticipants: int = 1 |

Abbildung entnommen aus „UML @ Classroom“, Seidl et al.

# Attribute (5)




- Spezifiziert wie viele Werte ein Attribut aufnehmen kann.
- Default Anzahl: 1
- Notation:

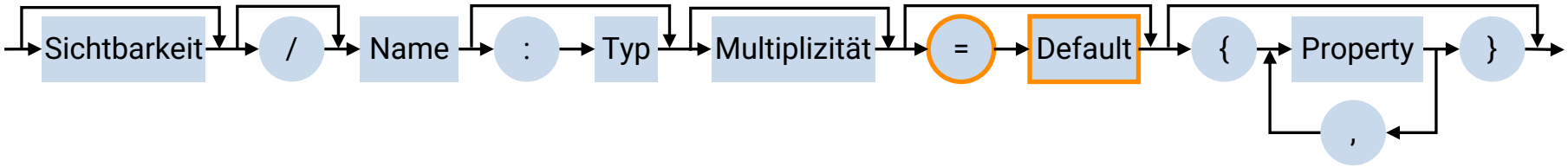



- Beispiele
  - `[0..1]` null oder eins
  - `[0..*]` beliebig viele (entspricht `[*]`)
  - `[1..*]` beliebig viele aber zumindest eins
  - `[5]` genau fünf

| Course |
| --- |
| - name: String {readOnly}<br>- leader: Person {readOnly}<br>- participants: Person[1..*] {unique, ordered}<br>- numberParticipants: int = 1 |

Abbildungen entnommen aus „UML @ Classroom“, Seidl et al.

# Attribute (6)

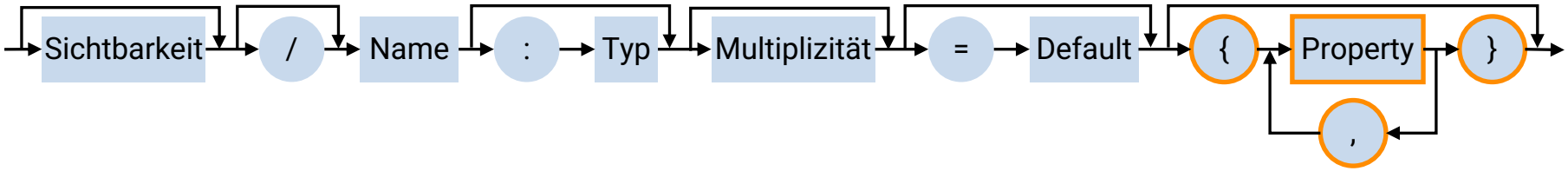



- Standardwert, der verwendet wird, wenn bei der Erzeugung nicht explizit ein anderer Wert angegeben wird.

| Course |
| --- |
| - name: String {readOnly}<br>- leader: Person {readOnly}<br>- participants: Person[1..*] {unique, ordered}<br>- numberParticipants: int = 1 |

Abbildung entnommen aus „UML @ Classroom“, Seidl et al.

# Attribute (7)




- Gibt zusätzliche Eigenschaften von Attributen an.
- Beispiele:

| Property | Beschreibung |
|---|---|
| {readOnly} | Keine Änderung nach der Initialisierung |
| {unique} | Kann keine Duplikate enthalten |
| {non-unique} | kann Duplikate enthalten |
| {ordered} | fixe Reihenfolge |
| {unordered} | keine fixe Reihenfolge |

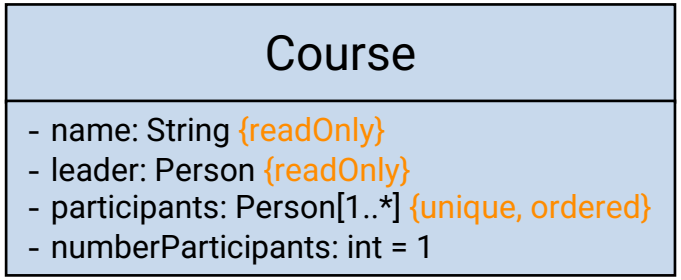
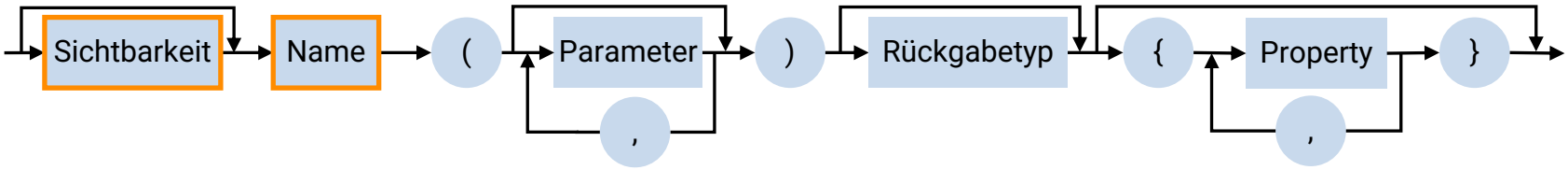



Abbildung entnommen aus „UML @ Classroom“, Seidl et al.

# Operationen (1)

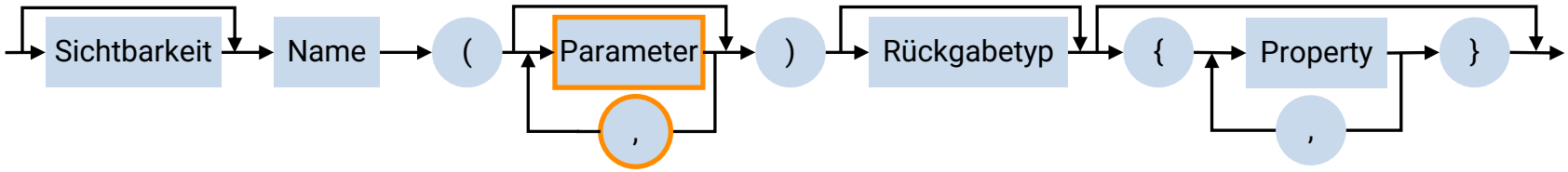



- Sichtbarkeit und Name wie bei Attributen

| Course |
| --- |
| ... |
| + Course(name: String, leader: Person, capacity: int)<br>+ register(person: Person): boolean<br>+ getEmailAddresses(): String[1..*] {unique, ordered}<br>+ getName(): Person<br>+ getLeader(): Person |

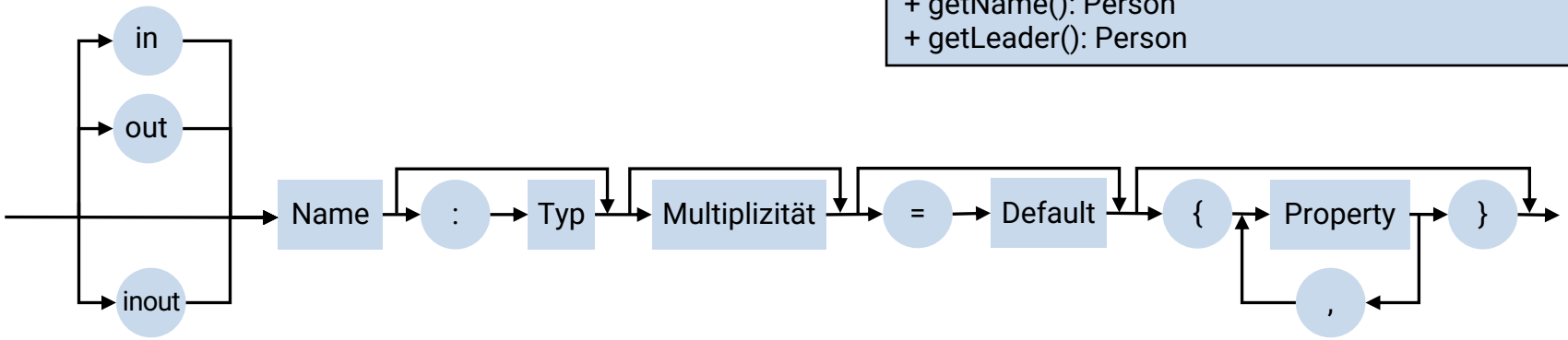
Abbildung angelehnt an UML „UML @ Classroom“, Seidl et al.

# Operationen (2)



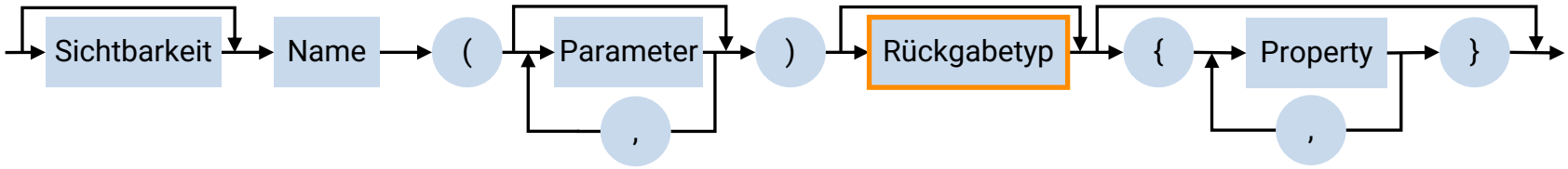

- Parameter
  - Für Parameter gilt die folgende, an Attribute angelehnte, Notation.

| Course |
| --- |
| ... |
| + Course(name: String, leader: Person, capacity: int)<br>+ register(person: Person): boolean<br>+ getEmailAddresses(): String[1..*] {unique, ordered}<br>+ getName(): Person<br>+ getLeader(): Person |

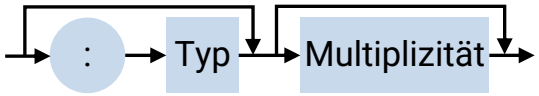



Abbildungen entnommen aus „UML @ Classroom“, Seidl et al.

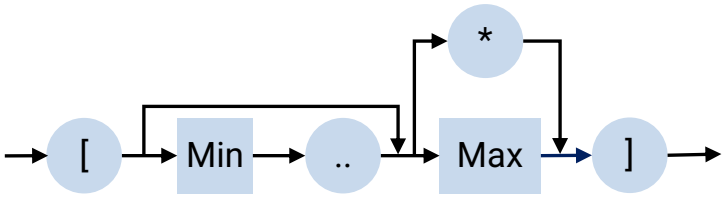
# Operationen (3)




- Rückgabetyp



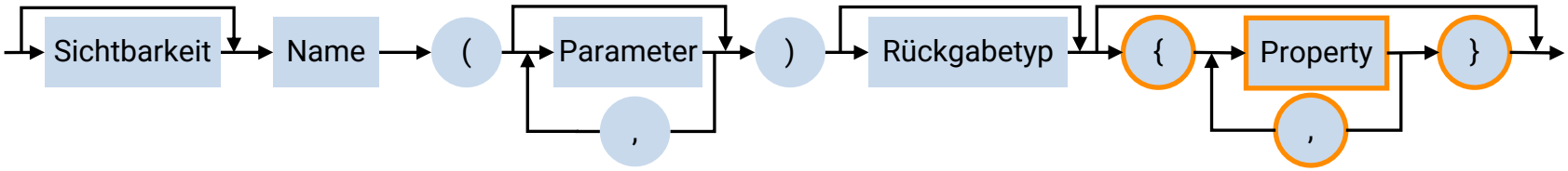

  - Typ gibt den Datentyp des Rückgabewerts an.
  - Multiplizität wie bei Attributen:




| Course |
| --- |
| ... |
| + Course(name: String, leader: Person, capacity: int)<br>+ register(person: Person): boolean<br>+ getEmailAddresses(): String[1..*] {unique, ordered}<br>+ getName(): Person<br>+ getLeader(): Person |

Abbildung angelehnt an UML @ Classroom, Martina Seidl et al., dpunkt.verlag, 2012.

# Operationen (4)




- Gibt zusätzliche Eigenschaften von Operationen an.
- Beispiele:

| Property | Beschreibung |
|---|---|
| {abstract} | Die Operation ist abstrakt. Alternative: die Operation kursiv schreiben |
| {leaf} | Die Operation darf in Unterklassen nicht überschrieben werden. |
| {query} | Die Operation verändert den Objektzustand nicht. |
| {sequential} {guarded} {concurrent} | Angaben zum Kontrollfluss über das Objekt. |

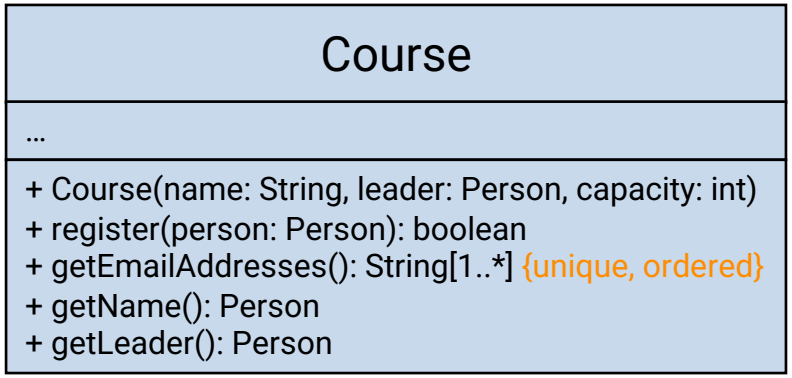
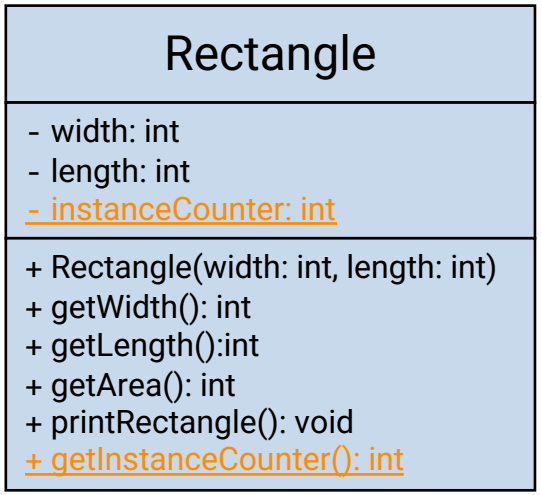



Abbildung entnommen aus „UML @ Classroom“, Seidl et al.

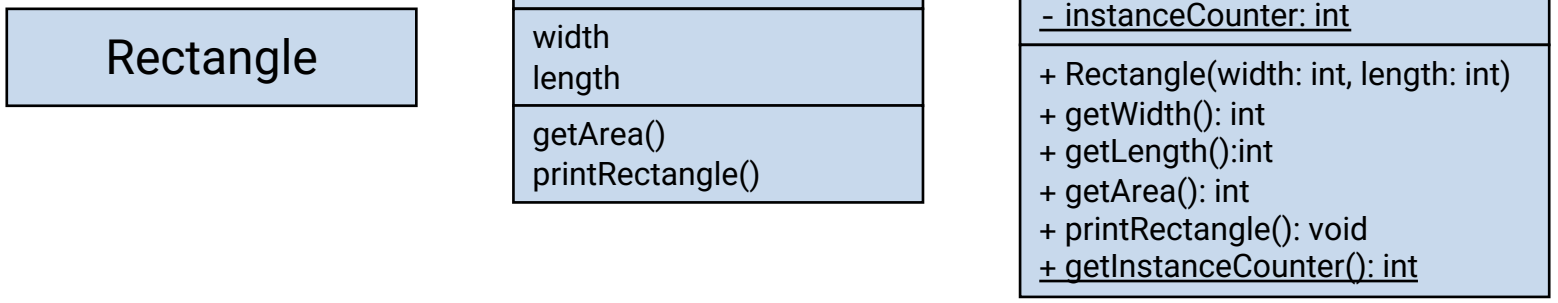
# Klassenvariable und Klassenoperation

- Merkmale sind standardmäßig auf Exemplarebene definiert.
  - Wird von Attribut oder Operation gesprochen sind im allgemeinen Exemplarattribut und Exemplaroperation gemeint.
- Unterstreichen kennzeichnet Klassenattribute bzw. Klassenvariablen
  - Sie können auch statische Attribute genannt werden
- Das selbe gilt auch für Klassenoperationen bzw. statische Operationen.



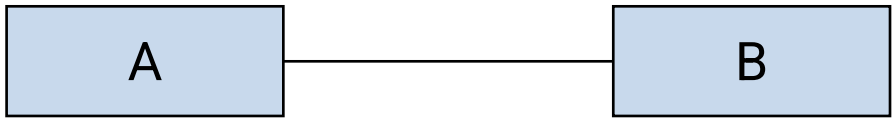
# Detailierungsgrad

grobgranular ⟷ feingranular

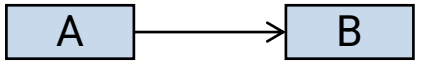


# Assoziation

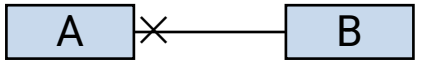
- Grundform (binäre Assoziation)



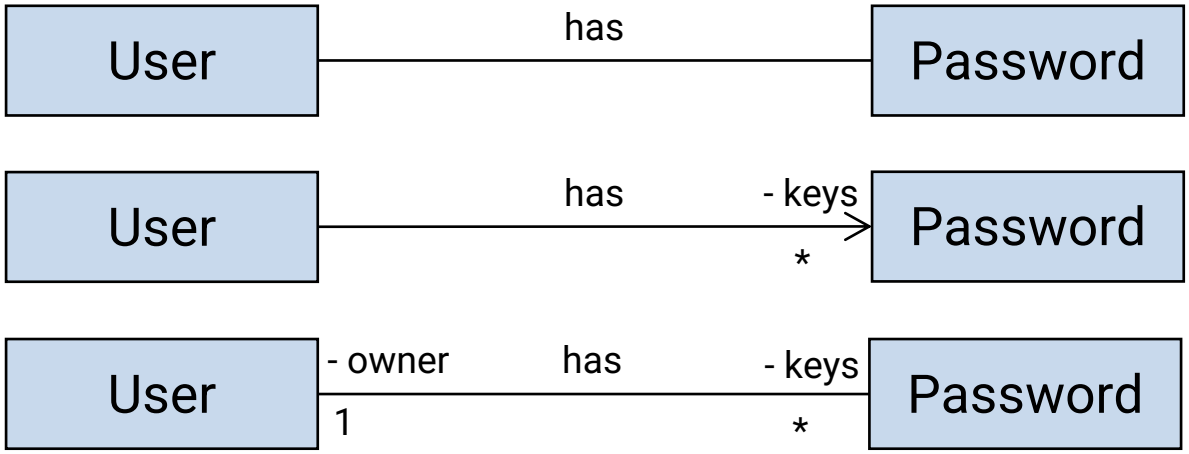

- Zusätzliche optionale Angaben
  - Benennung in der Mitte der Kante
    - Anzeigen der Leserichtung ist durch ► bzw. ◄ möglich
  - Rollen an den Enden der Assoziation
  - Multiplizitäten an den Enden, wie bei Attributen
  - Angabe von Eigenschaften
    - `{ordered}`
    - `{unique}`
    - …
  - Navigationsangaben
    - Zulässige Navigationsrichtung (durch Pfeil ⟶)

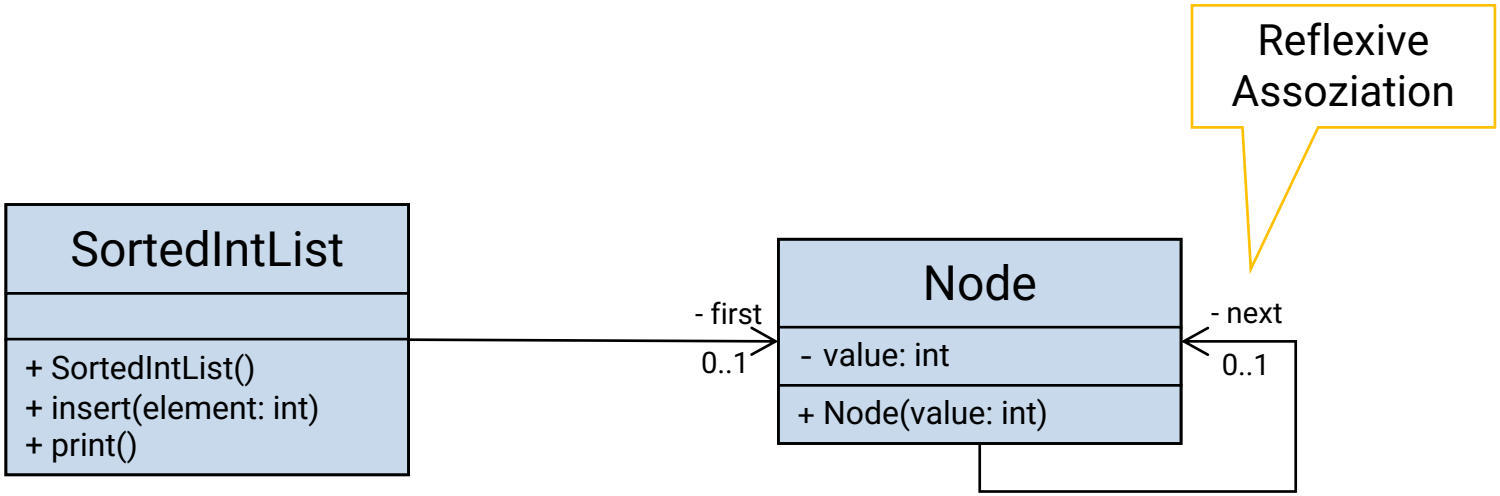



    - Keine Navigation erlaubt (durch ✕ ⟝ )

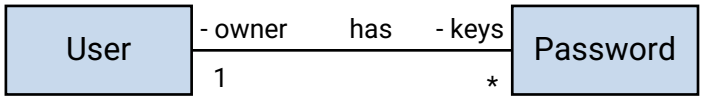


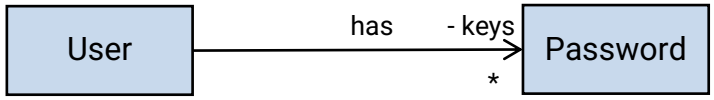
# Beispiele


User
has
Password
User
has
- keys
*
Password
User
- owner
1
has
- keys
*
Password


Reflexive Assoziation
SortedIntList
+ SortedIntList()
+ insert(element: int)
+ print()
- first
0..1
Node
- value: int
+ Node(value: int)
- next
0..1

# Beispiel (Java Umsetzung)




```java
public class User {
    private Password[] keys;
    …
}
```

```java
public class Password {
    private User owner;
    …
}
```

```java
public class User {
    private Password[] keys;
    …
}
```

```java
public class Password {
    …
}
```

# Darstellungsmöglichkeiten für Assoziationen

- Beide Repräsentationen sind äquivalent.
  - In der Praxis ist die linke Darstellung zu bevorzugen, da die Assoziation sofort ersichtlich ist.

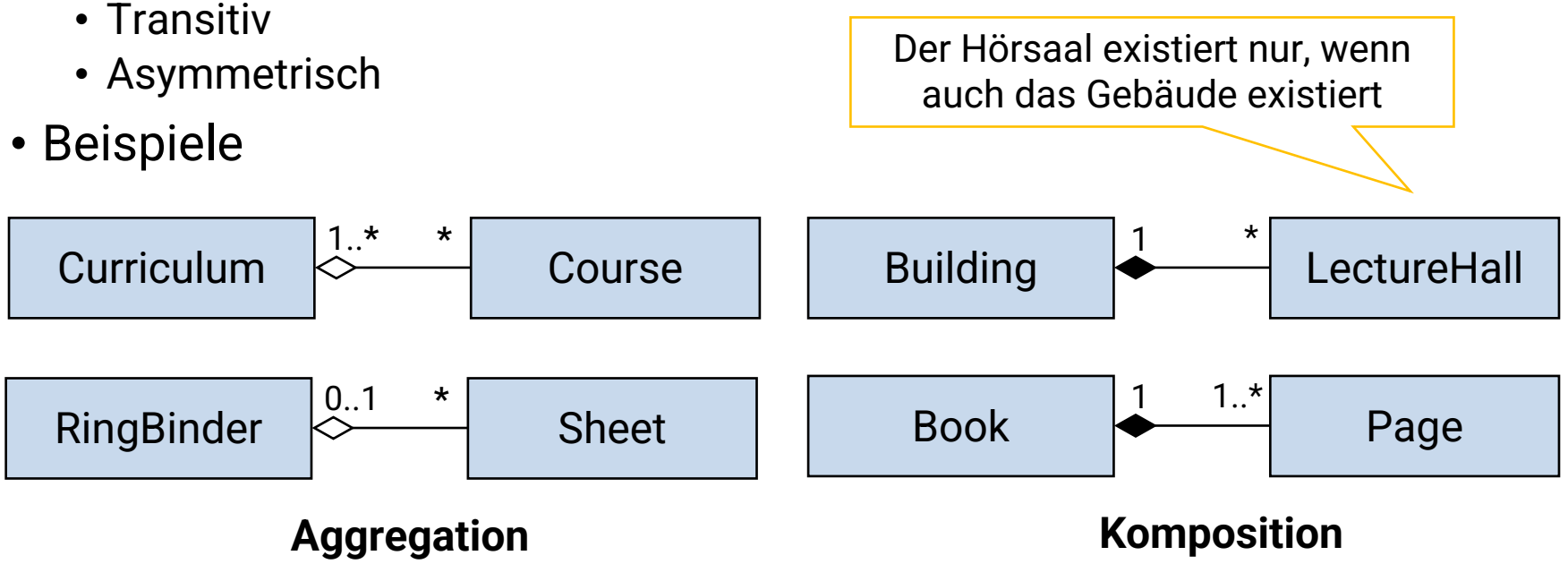

- Äquivalenter Java Code

```
public class Password {
    …
}
```

```
public class User {
    private Password[] keys;
    …
}
```

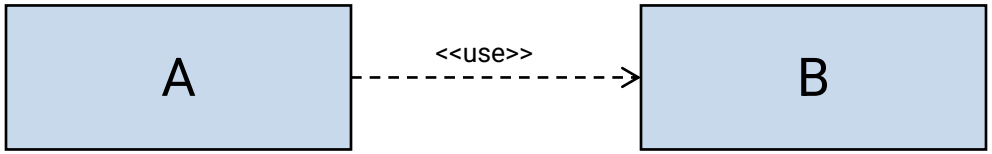
# Aggregation

- Aggregation ist eine spezielle Assoziation (Teile-Ganzes-Beziehung).
- Komposition ist eine speziellere (strengere) Form der Aggregation mit folgenden Einschränkungen.
  - Ein Teil darf Kompositionsteil höchstens eines Ganzen sein.
  - Ein Teil existiert nur so lange wie sein Ganzes.
- Eigenschaften
  - Transitiv
  - Asymmetrisch
- Beispiele



# Abhängigkeitsbeziehungen

- Abhängigkeitsbeziehungen stellen allgemeine, nicht näher spezifizierte Abhängigkeiten dar.
- Notation:




- A hängt von B ab
- Änderungen in B können zu Änderungen in A führen
  - A beinhaltet Attribute/Methodenparameter vom Typ B
  - A erzeugt Objekte vom Typ B
- Beispiel: Client- Server

# Generalisierung

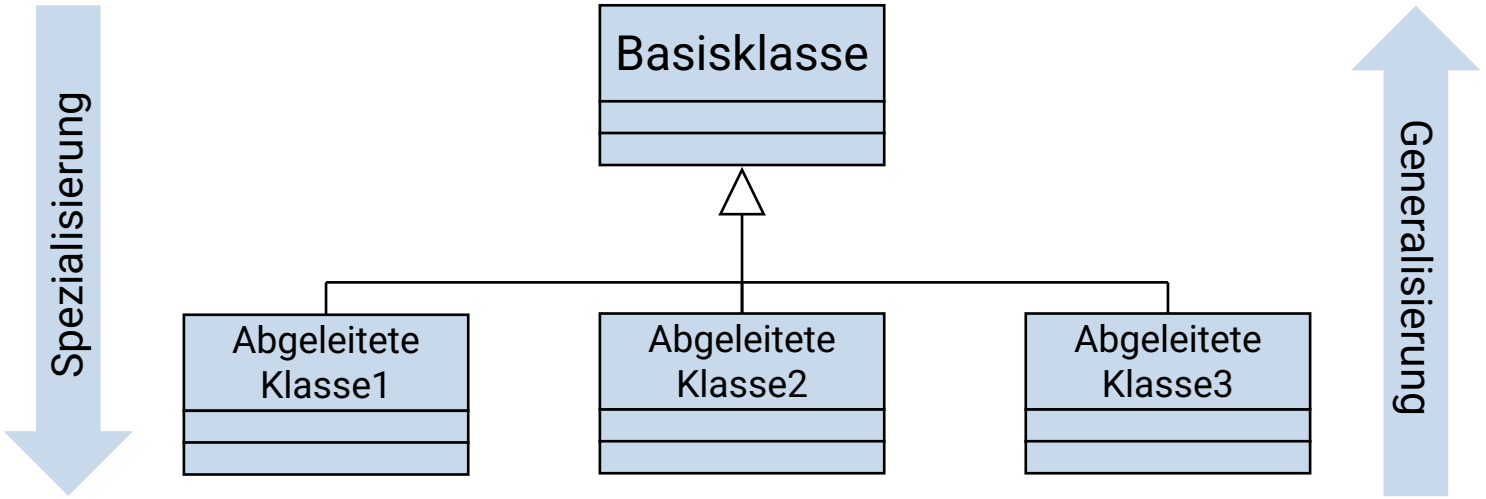



Abbildung angelehnt an „Methoden und Architekturen der Softwaretechnik“, Goll

# Quellen


- Martina Seidl, Marion Brandsteidl, Christian Huemer, Gerti Kappel: **UML @ Classroom: Eine Einführung in die objektorientierte Modellierung**, dpunkt.verlag, 2012
- Joachim Goll: **Methoden und Architekturen der Softwaretechnik**, Vieweg + Teubner Verlag, 2011