Komposition und Aggregation
"hat ein" - Beziehung

Objekte, die Teil eines anderen Objektes sind

Quelle: Online Tutorial, Kap. 35
(

Ausgangspunkt

Es spielt keine Rolle wo ich ein Objekt verwende!

Voraussetzung:

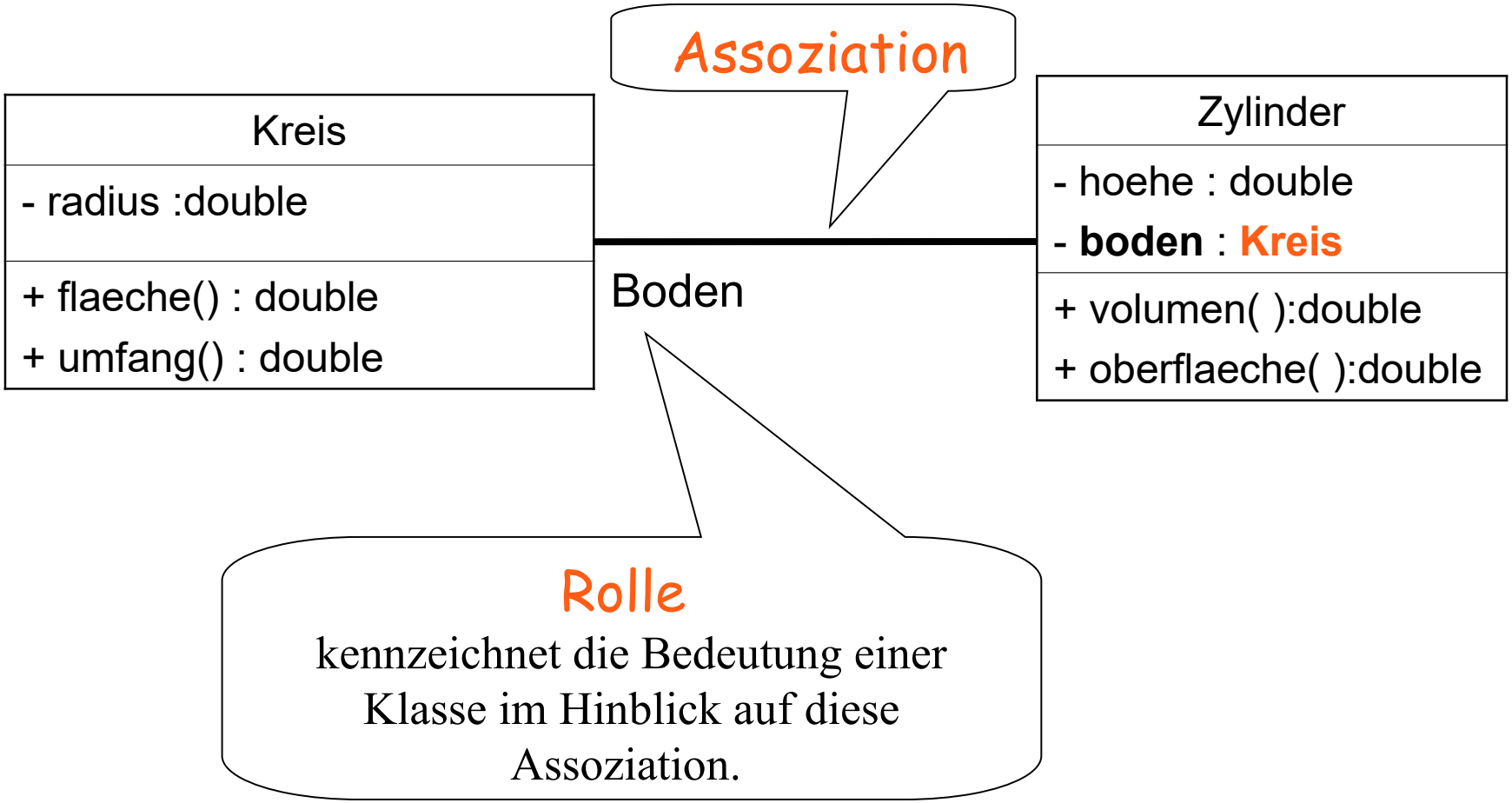
1. Es wurde mit **new** erzeugt.
   Bsp: **Kreis k = new Kreis(2.3);**
2. Eine **Referenz** auf das Objekt ist bekannt.
   - in **main**:
     Kreis k = new Kreis(2.7); *//Referenz und new zusammen*
   - als **Eigenschaft**:
     private Kreis k; *//wird z.B. im Konstruktor mit new erzeugt*
   - als **Parameter**
     public static void methode (**Kreis k**){ //…
     //wurde in main oder irgendwo anders mit new erzeugt

➔ **Referenz bekannt? Objekt existiert?**
Alle public Methoden der Klasse nutzbar!!!

```
k.setRadius(4.3);
double r = k.getRadius();
double f = k.flaeche();
double u = k.umfang();
```

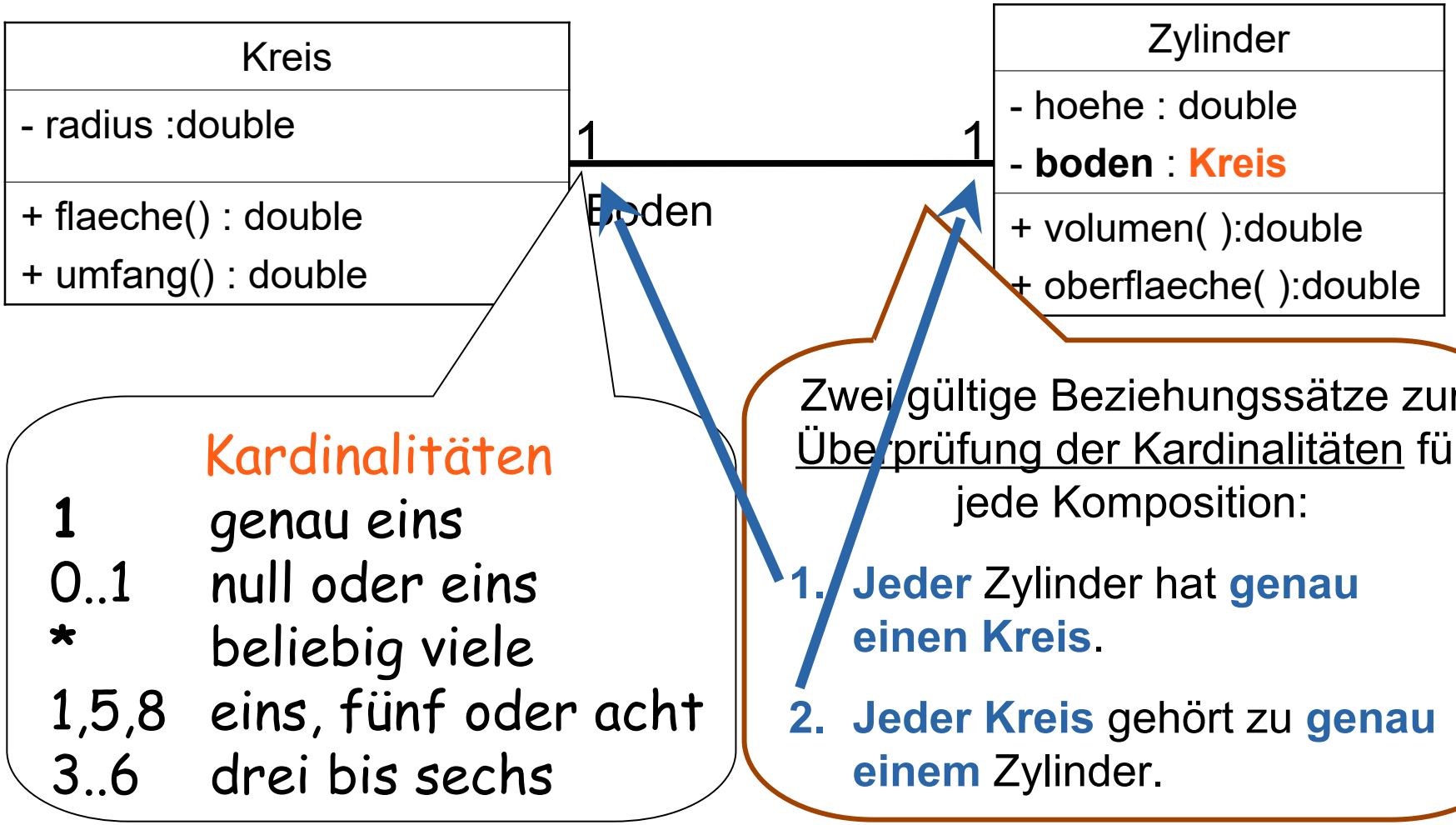
Eine Assoziation in der OOA

Ein Zylinder hat einen Kreis als Boden



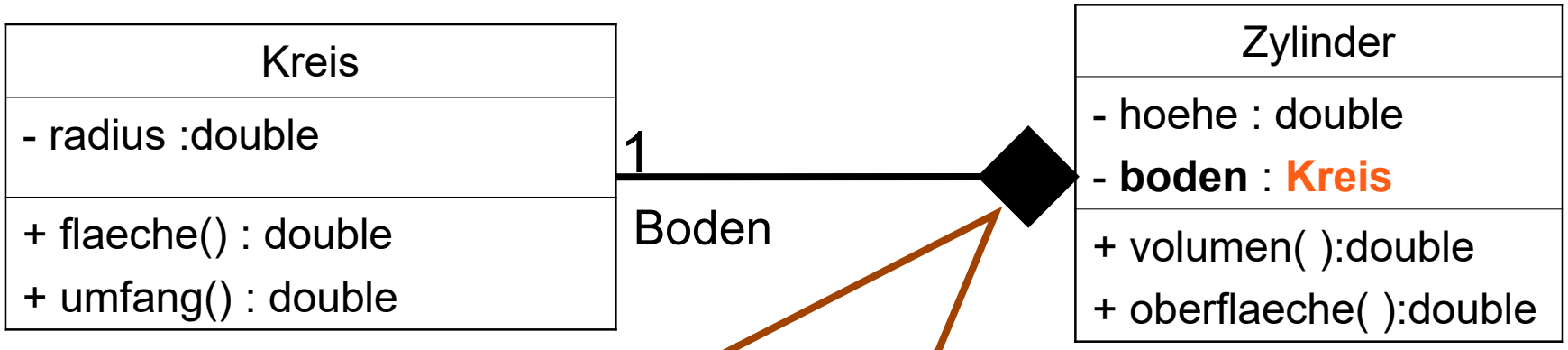
Eine Assoziation in der OOA

Ein Zylinder hat einen Kreis als Boden



Komposition

Ein Zylinder hat einen Kreis als Boden

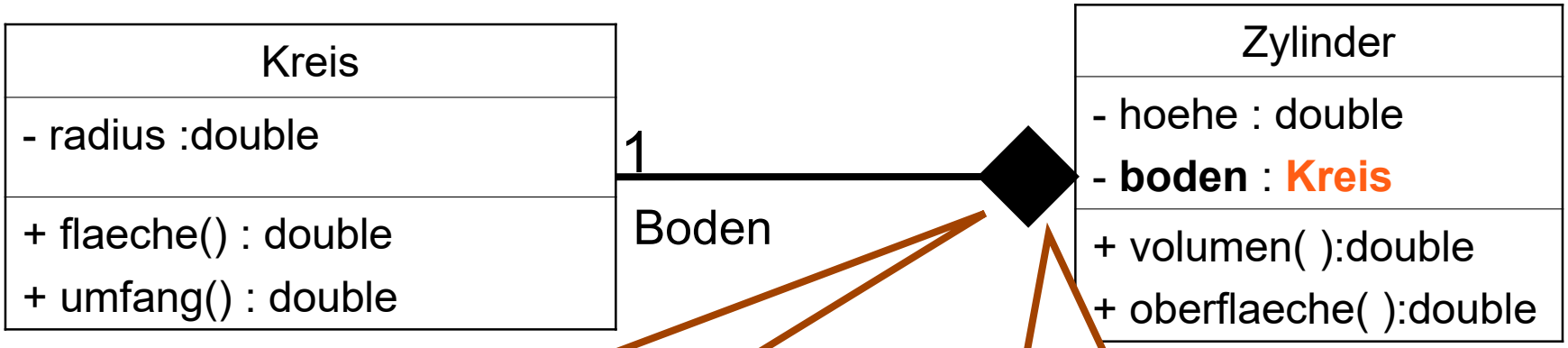


Komposition

Eine Klasse enthält eine **Eigenschaft vom Typ der anderen Klasse**. Die Raute gibt an, welche Klasse die Eigenschaft enthält, d.h. das **Ganze** ist. Die andere Klasse ist dann ein **unzertrennlicher Teil** des Ganzen.

Komposition

Ein Zylinder hat einen Kreis als Boden

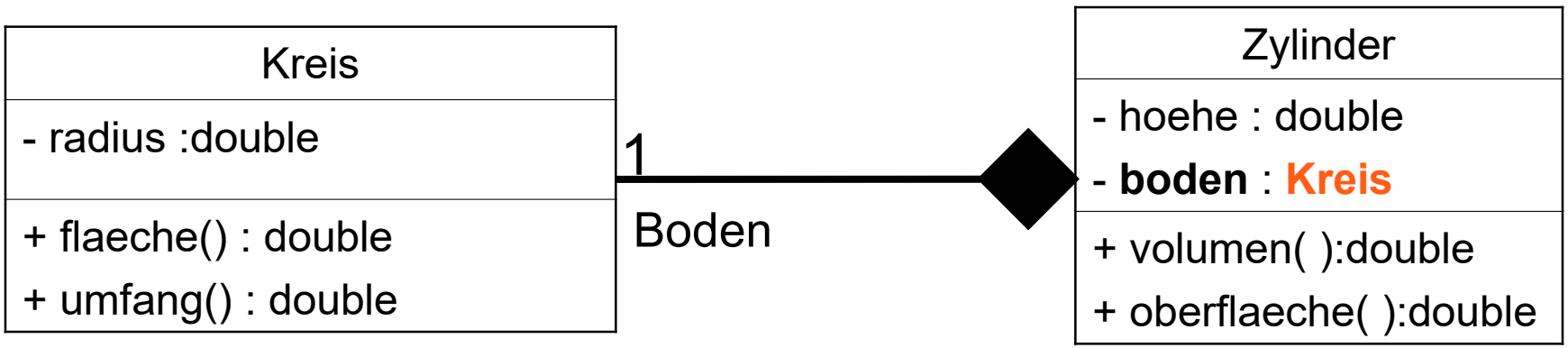


Komposition

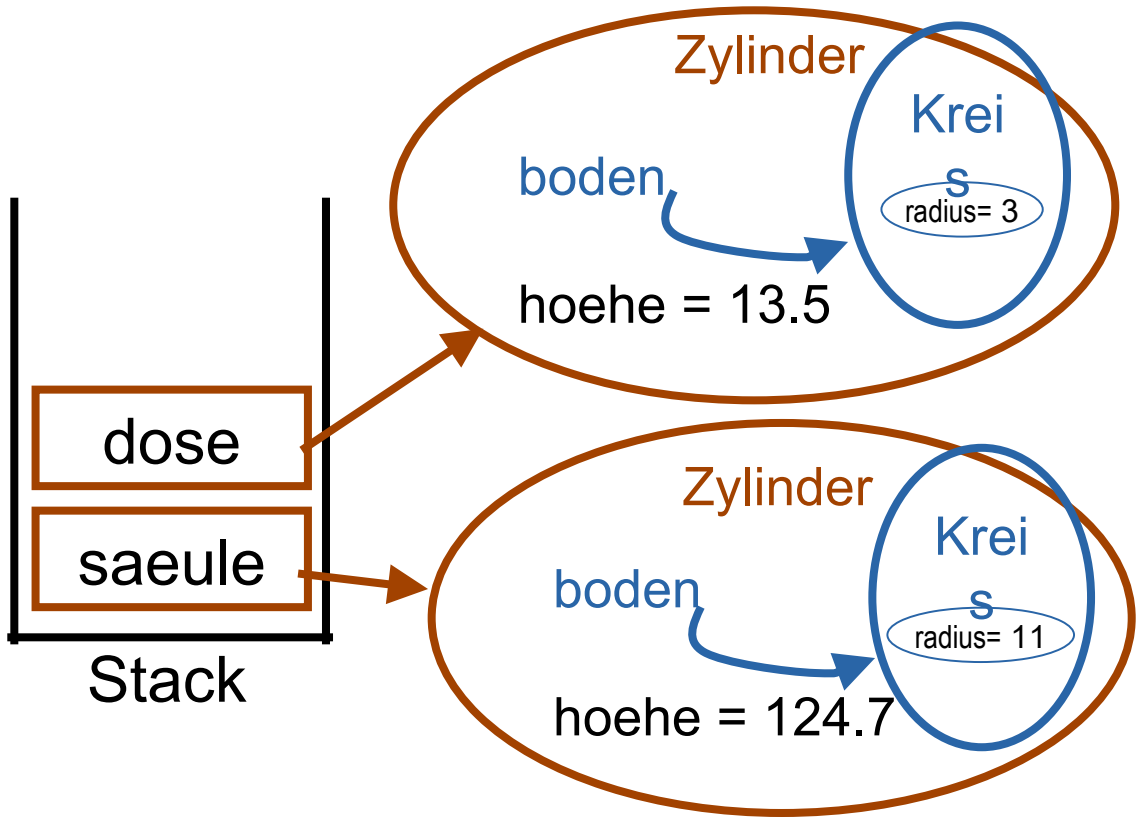
Bei der Komposition **leben und sterben** die Teile mit dem Ganzen.

Die Kardinalität ist immer 1 oder 0..1 und kann daher weggelassen werden.

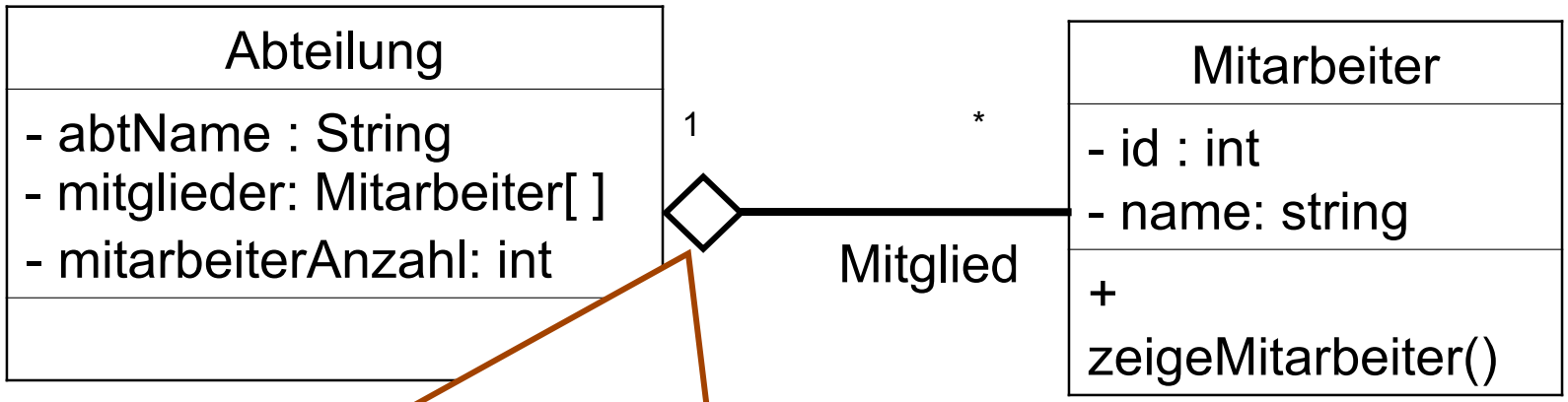
Komposition




```
Zylinder saeule =
    new Zylinder(11, 124.7 );
Zylinder dose =
    new Zylinder(3, 13.5);
```



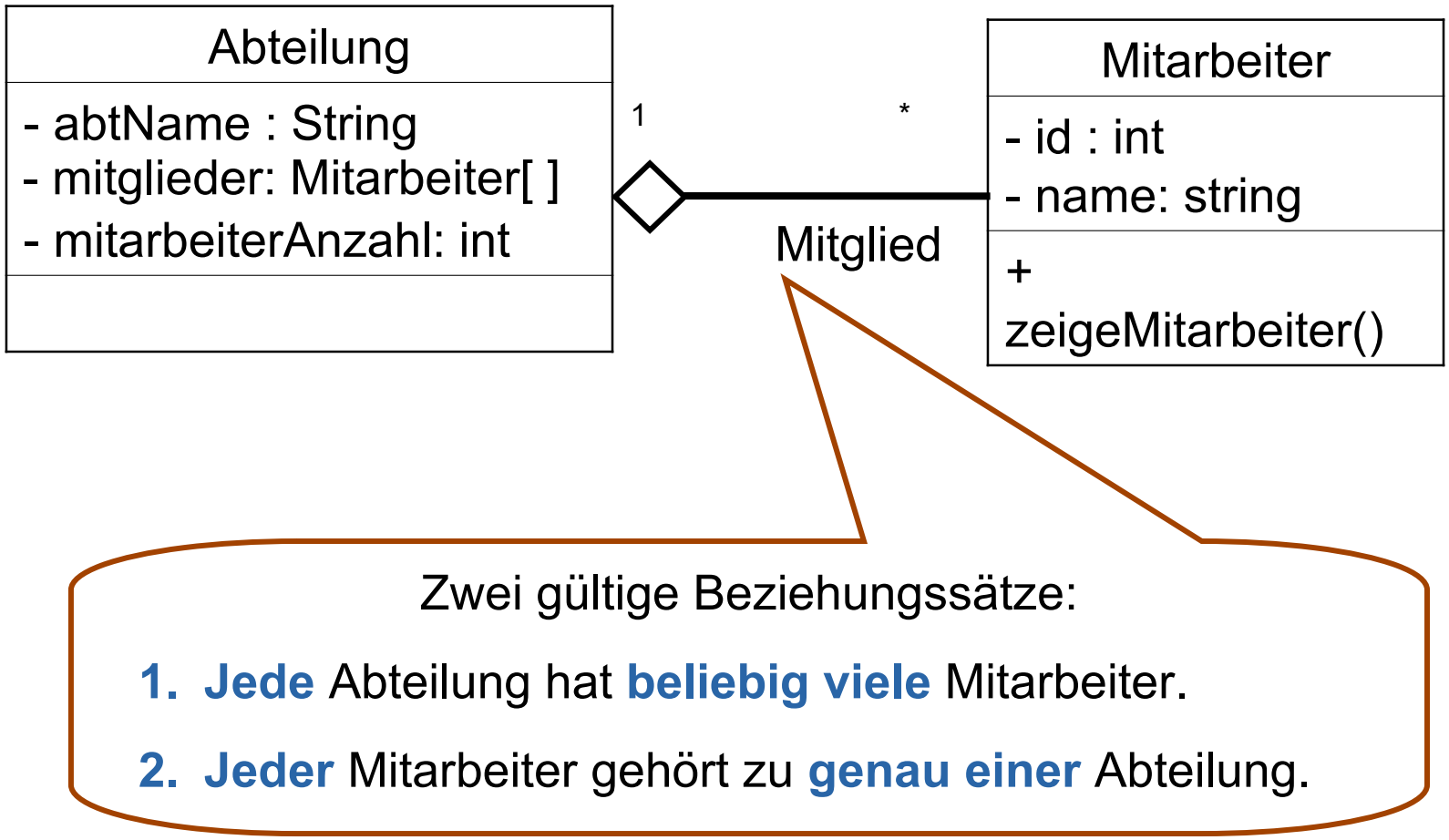
Aggregation



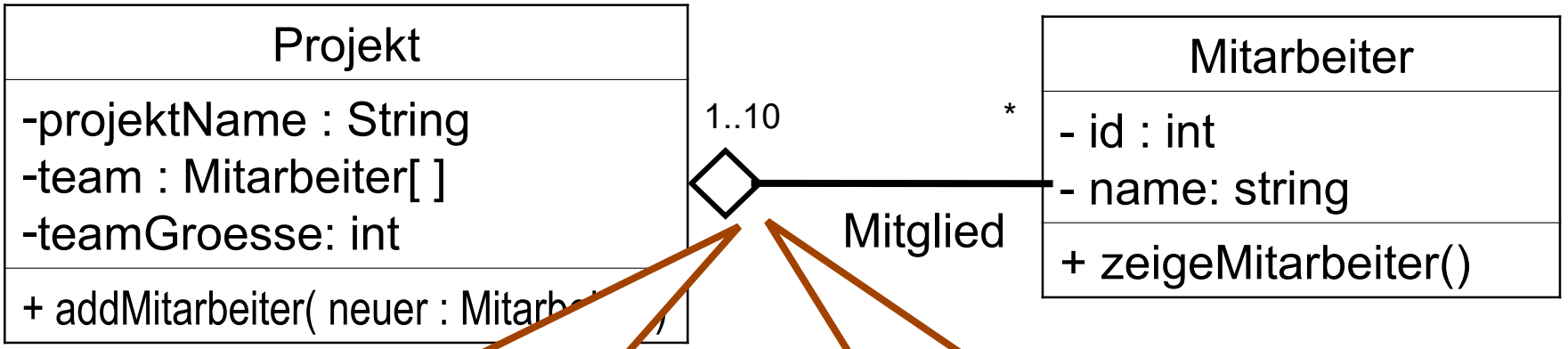
Aggregation

Die Raute gibt an, welche Klasse das "Ganze" ist. Objekte der anderen Klasse **existieren unabhängig** von dem Aggregationsobjekt. Wird die Abteilung aufgelöst, existieren die Mitarbeiter weiter.

Aggregation



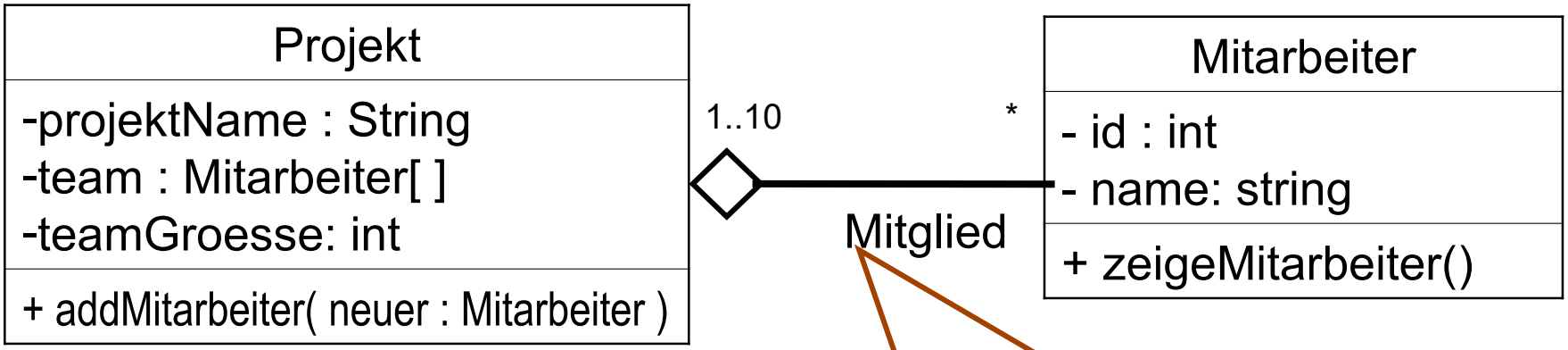
Shared Aggregation



„shared aggregation“
Mitarbeiter können gleichzeitig mehreren Projekten zugeordnet werden.

Die **Kardinalität** an der Raute kann alles sein. Bei Aggregation **darf** sie **nicht weggelassen** werden.

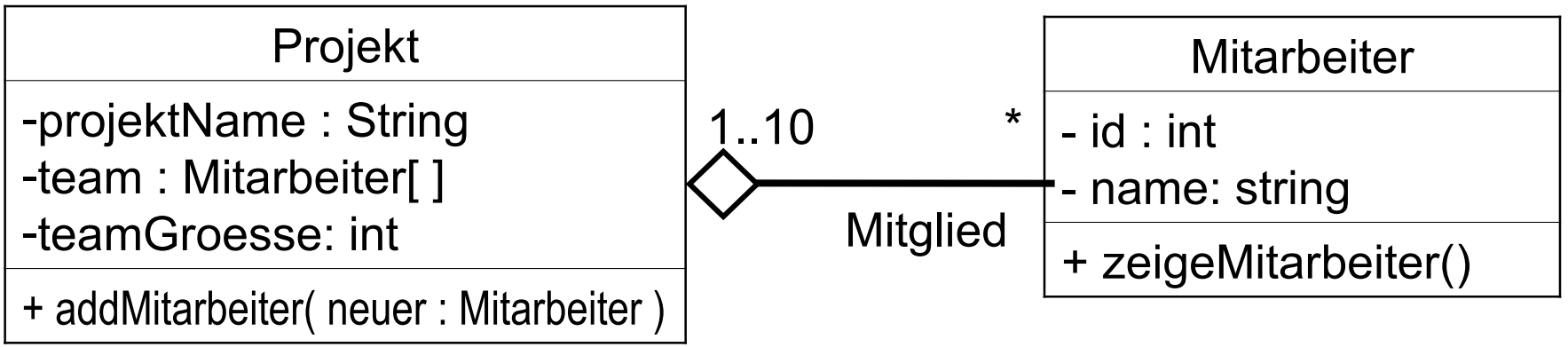
Shared Aggregation

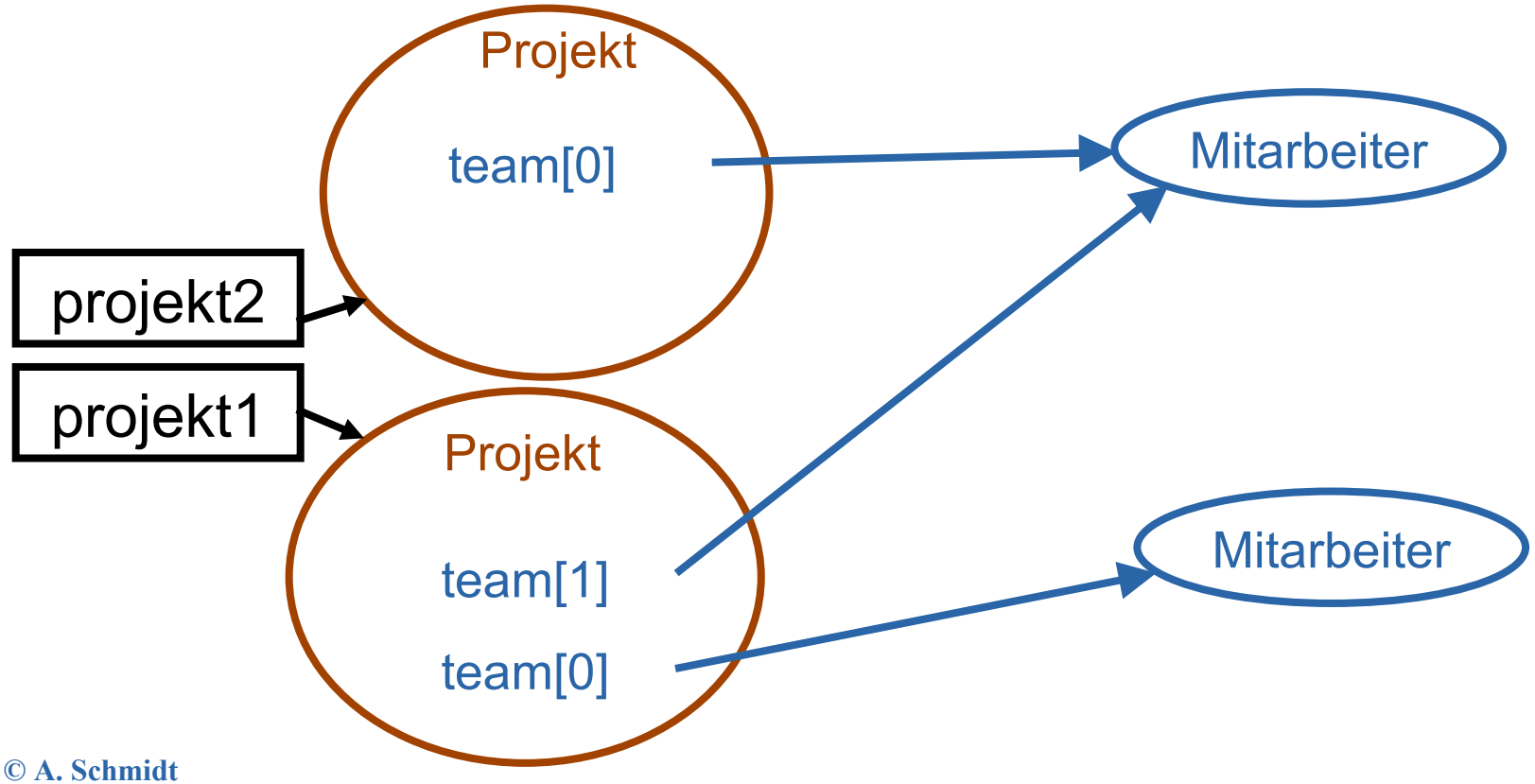


Zwei gültige Beziehungssätze:

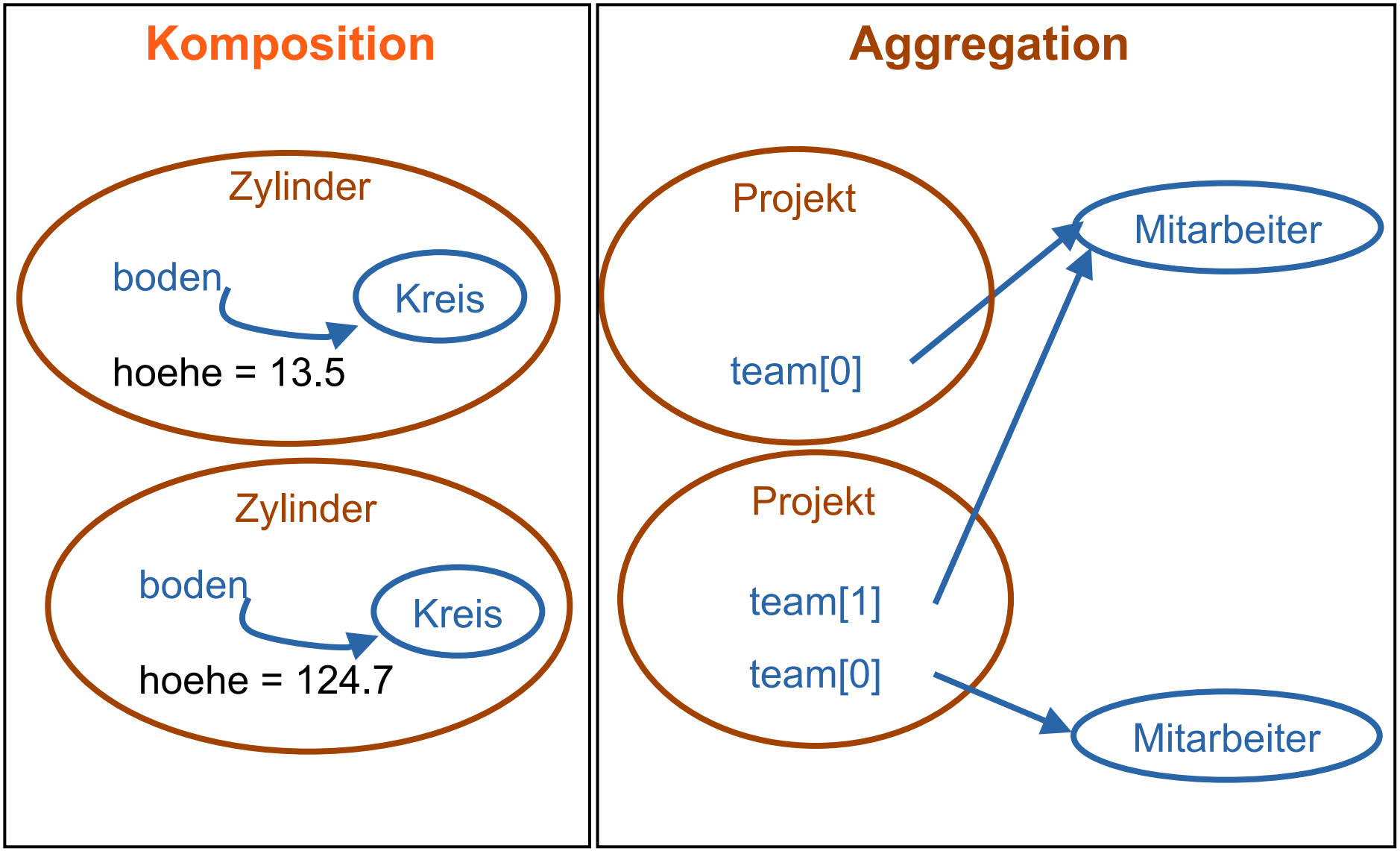
1. **Jedes** Projekt hat **beliebig viele** Mitarbeiter.
2. **Jeder** Mitarbeiter arbeitet an **ein bis zehn** Projekten mit.

Aggregation





Komposition vs. Aggregation



Komposition vs. Aggregation

```
//Komposition
public Zylinder( double hoehe, double radius)
{
    this.setHoehe( hoehe );
    this.boden = new Kreis(radius);
}

//Aggregation
public void addMitarbeiter( Mitarbeiter neuer )
{
    if( neuer != NULL )
    {
        this.team[ teamGroesse ] = neuer;
        this.teamGroesse++;
    }
}
```

Objekterzeugung in Klasse Zylinder

Übergabe eines bestehenden Objektes

besteht Objekt wirklich?

Zuweisung statt Erzeugung

Vergleich Komposition vs. Aggregation

- Beziehung kann durch "besteht aus" oder "ist enthalten in" beschrieben werden (whole part).
- Kardinalität der Kompositionsklasse ist 1 oder 0..1.
- Wird das Ganze gelöscht, dann werden automatisch seine Teile gelöscht (they live and die with it). Ein Teil darf jedoch zuvor explizit entfernt werden. Das Ganze funktioniert aber nicht wenn ein Teil entfernt wird.
- Die dynamische Semantik gilt auch für seine Teile. Beispiel: Kopiert man das Ganze, werden auch die Teile kopiert.

=

- Beziehung kann durch "besteht aus" oder "ist enthalten in" beschrieben werden (whole part).
- Kardinalität der Aggregationsklasse kann beliebig sein.
- Teile existieren unabhängig. Das Ganze funktioniert auch, wenn ein einzelnes Teil entfernt wird.

Konsequenzen für die Programmierung:

- Die Klasse erzeugt das Teil-Objekt - meistens im Konstruktor.
  (ganz neu oder Kopie)
- Der Konstruktor erzeugt kein Objekt. Referenz zeigt auf Außen erzeugtes Objekt.

Komposition oder Aggregation?

Kann das Teil für sich alleine existieren?

Ja

Ist das Ganze noch funktionstüchtig, wenn das Teil entfernt wird?

Ja

Sollen sich **alle** Ganze **verändern**, wenn ein **Teil verändert** wird?

Ja

Trägt das Ganze Verantwortung für seine Teile? D.h. soll es sie erstellen und vernichten?

Nein

Aggregation

Nein

Herz ohne Mensch?

Nein

PKW mit nur 2 Rädern?

Nein

Tank füllen immer bei allen Kfz?

Ja

Liste erstellt Listenelemente selbst

Komposition

Aufgabe 1

Bilden Sie eine Klasse **Zylinder**, welche einen Kreis, der Grundfläche und Deckel definiert, und eine Höhe hat. Es sollen Volumen und Oberfläche errechnet werden.
Volumen = Grundfläche eines Kreises * Höhe
Oberfläche = 2 * Fläche des Grundkreises + Umfang des Kreises * Höhe

J14_Figuren

Aufgabe 2

Bilden Sie eine Klasse **Kegel**, die einen Kreis als Grundfläche und eine Höhe hat. Es sollen Volumen und Oberfläche errechnet werden.

Volumen = 1/3 * Grundfläche * Höhe
Oberfläche = Grundfläche + Mantelfäche des Kegel
Kegel: Mantelfläche Kegel = r * PI * Seitenlänge Mantel
Seitenlänge Mantel = Wurzel aus $Höhe^2 + r^2$

J14_Figuren

Aufgabe 3

Erstellen Sie auch noch die Klassen **Quader** und **Pyramide**. Die Formeln entsprechen denen des Zylinders und des Kegels.

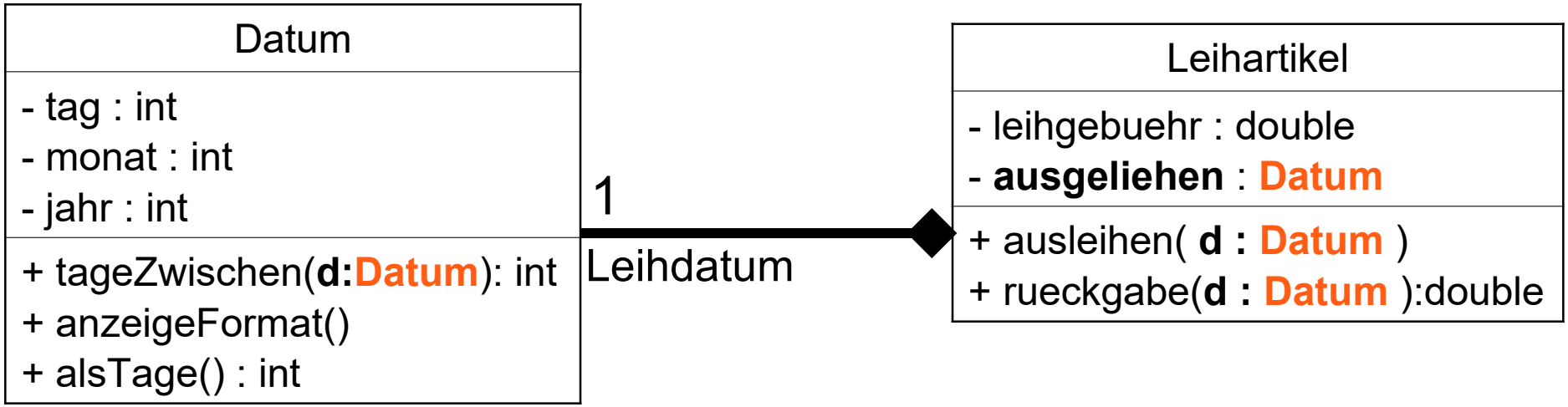
J14_Figuren

Aufgabe 4

Bilden Sie die Klasse Abteilung die einen Abteilungsleiter und beliebig viele Mitarbeiter hat.
(Tip: Implementieren und Testen Sie die Einbettungen schrittweise.)

J14_Abteilung

Aufgabe 5



Erstellen Sie das abgebildete Klassendiagramm. **J14_Artikel**

- Ihre Klasse Datum aus der Präsentation J10 muss noch um die Methode tageZwischen erzgänzt werden. Die Methode tageZwischen() ermittelt die Anzahl der Tage zwischen dem eigenen Datum und dem übergebenen Datum. Sie soll intern die Methode alsTage sowohl mit dem eigenen Datum wie auch mit dem Parameter d benutzen.
- Die Methode rueckgabe() ermittelt mit Hilfe der Methode tageZwischen und der Leihgebühr den zu zahlenden Betrag. Die Leihgebühr ist die zu zahlende Gebühr pro Tag.
- Achtung: Obwohl es eine Komposition ist, erstellt diesmal nicht der Konstruktor, sondern die Methode ausleihen beim ersten Aufruf das Objekt ausgeliehen. Sie muss prüfen, ob ausgeliehen noch NULL enthält, bevor das Objekt angelegt wird.

Aufgabe 6 Modellierung Schachspiel

Zeichnen Sie das UML-Klassendiagramm für die Beziehung zwischen Schachbrett, Feld und Schachfigur.

Jedes **Feld** auf einem Schachbrett hat eine Farbe (schwarz oder weiß) und ist entweder leer, oder hat eine darauf stehende Figur. Neben den üblichen get- und set-Methoden gibt es eine Methode, die abfragt, ob das Feld leer ist.

Ein **Schachbrett** hat 8x8 Felder. Die Methode neuesSpiel() stellt alle Figuren zum Anfang eines neuen Spiels auf. Außerdem muss ein Schachbrett eine Methode besitzen, um eine Figur von einem Feld auf ein anderes zu bewegen (entfernen und setzen).

Eine **Schachfigur** kennt ihre eigene **Position** (Zahl und Buchstabe) und hat eine Farbe (schwarz oder weiß). Nur zum setzen einer neuen Position erhält die Spielfigur Zugriff auf das gesamte Schachbrett.

In dieser ersten Version der Schachbrettaufgabe gibt es nur **Bauern** als Schachfiguren. Die dürfen beim ersten Mal zwei Schritte nach vorne gehen und danach immer nur einen. Sie dürfen immer nur nach vorne gehen, wenn der Weg frei ist.