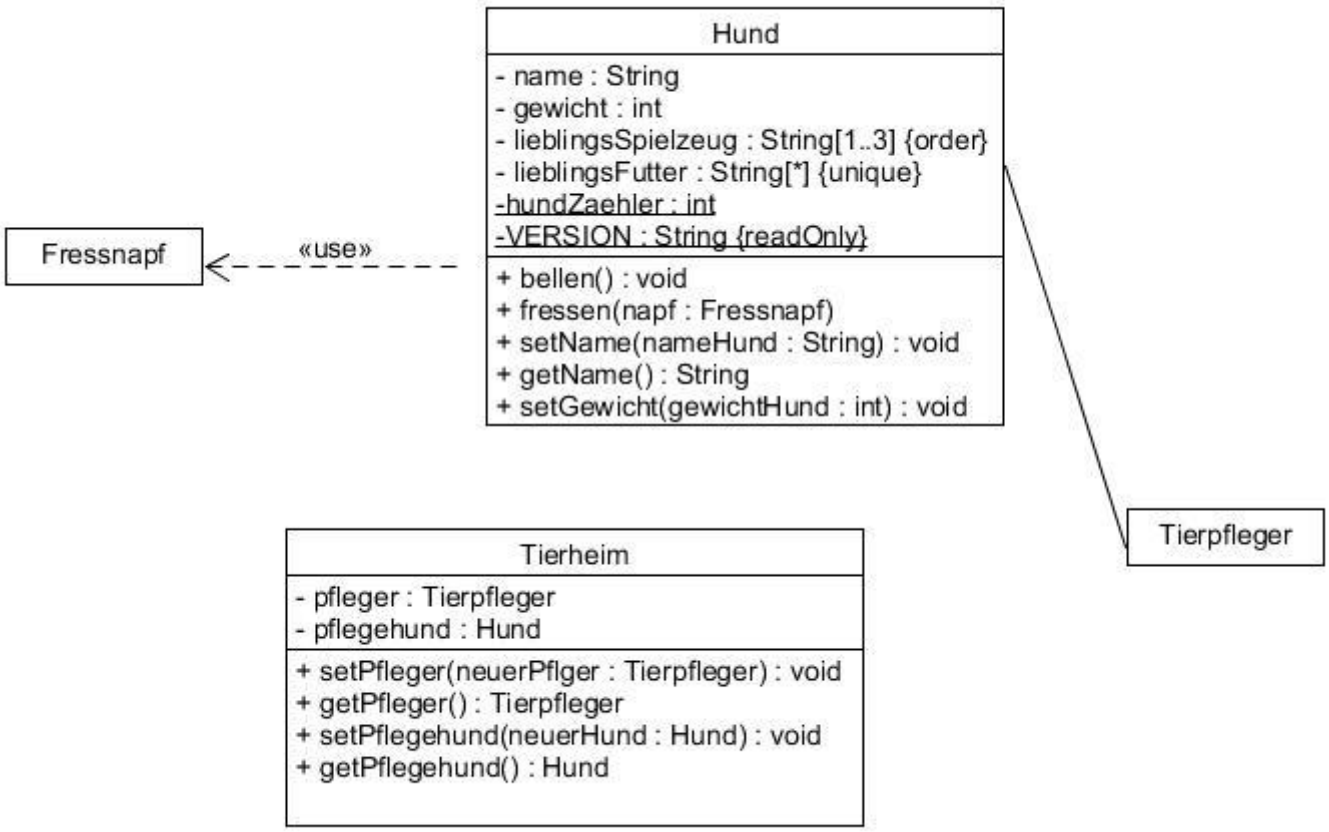
Objektorientierte Analyse / Objektorientiertes Design:

Aggregation vs. Komposition

Ziel: Wir modellieren die Beziehungen zwischen Klassen, um sie anschließend sinnvoll programmieren zu können. Dies haben wir schon bei Assoziationen getan → Wir möchten dies aber noch um zwei Beziehungen erweitern.

(Textquelle: https://www.codeadventurer.de/?p=2428)

Als Start- Beispiel dient das UML Klassen -Diagramm eines Tierheims

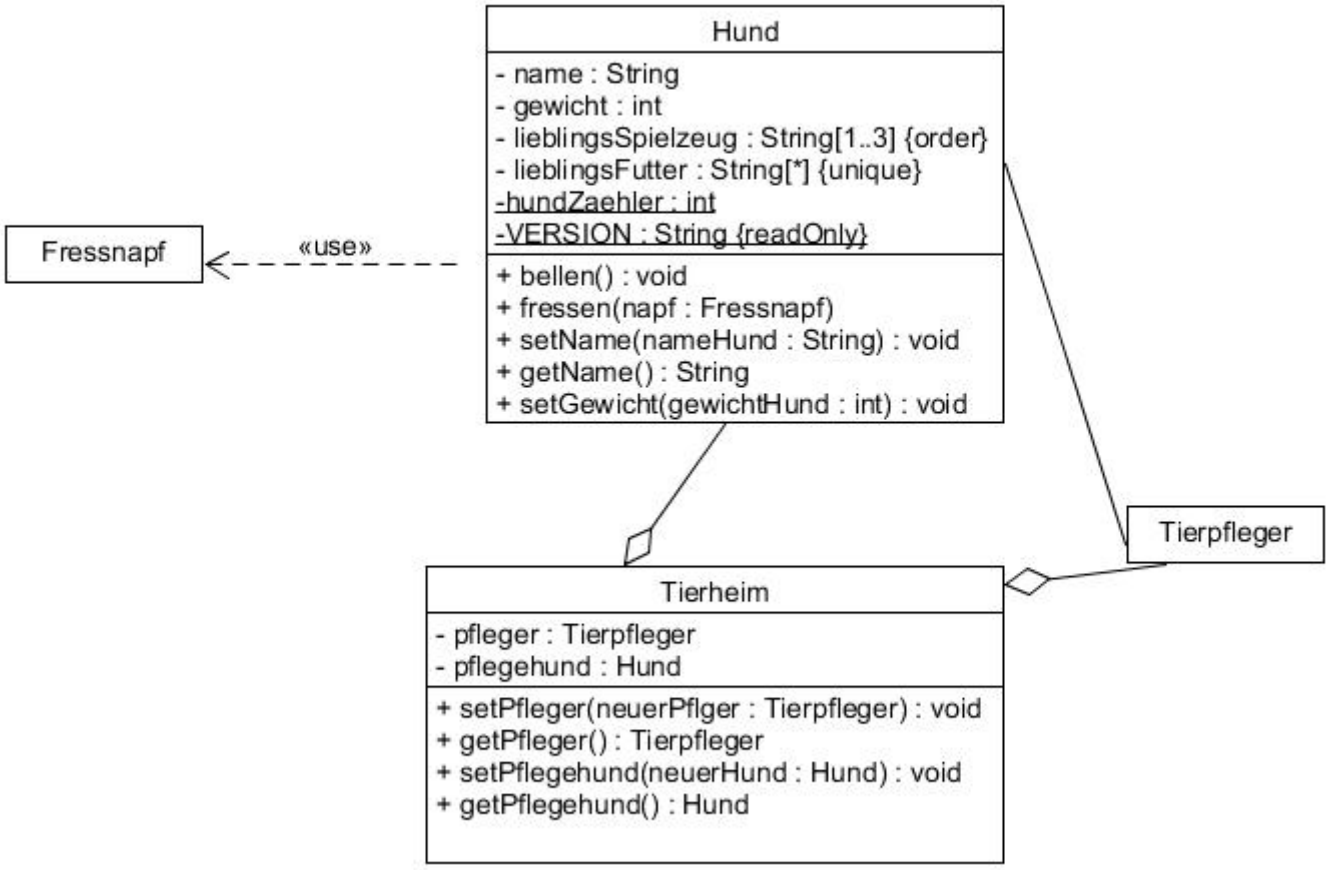


Wir verändern das Klassendiagramm unter folgender Annahme:

- Ein Tierheim besitzt Instanzen eines Tierpflegers und eines Hundes
- Allerdings ist auch hier wieder wichtig,
  dass sowohl der Pflegehund als auch
  der Tierpfleger ohne das Tierheim existieren können

Eine solche Beziehung heißt **Aggregation** und wird mit einem Diamantenzeichen im UML Klassendiagramm gekennzeichnet.

Das veränderte Diagramm sieht jetzt so aus:



Die Komposition!

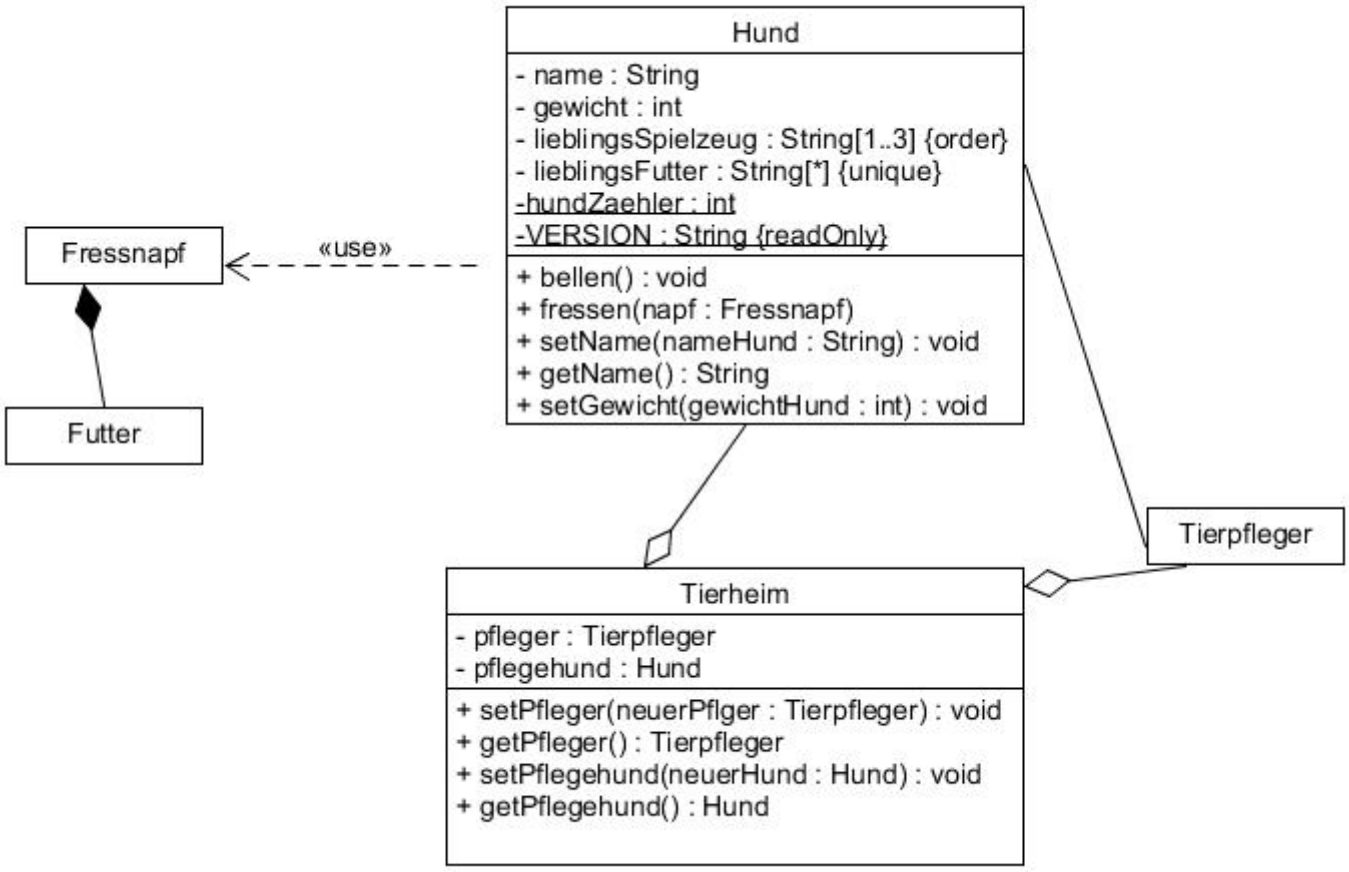
Eine stärkere Assoziation ist die sogenannte Komposition.

Bei diesem Assoziationstyp ist die Beziehung so stark, dass mit dem Löschen des „Behälterobjekts“ auch das integrierte Objekt verschwindet.

Genau das ist bei unserem Fressnapf der Fall!

Denn werfen wir den mit Futter gefüllten Fressnapf weg, verlieren wir auch das darin enthaltene Futter.

Eine Komposition kennzeichnen wir mit einem ausgefüllten Diamantzeichen.



Wie unterscheiden sich Aggregation und Komposition in der Implementierung?

Entscheidender Unterschied zwischen Aggregation und Komposition ist die Stärke der enthält Beziehung.

Schauen wir uns das an einem Beispiel an.

```
Hund bello = new Hund();
Tierheim heim = new Tierheim(bello);
```

Hier erzeugen wir eine Hunde-Instanz `bello`, die wir über den Konstruktor der Klasse `Tierheim` in ein Heim einquartieren.

Was passiert mit `bello`, wenn wir die Tierheim Instanz löschen?

Nix! Denn die Instanz `bello` liegt in einem eigenen Speicherbereich, der unabhängig von dem Bereich, in welchem das Tierheim liegt ist.

Die Tierheim Instanz enthält lediglich eine Referenz auf das Objekt `bello`.

Daher handelt es sich in diesem Fall um eine Aggregation.

Werfen wir einen Blick auf die Komposition.

```
Fressnapf napf = new Fressnapf(new Futter());
```

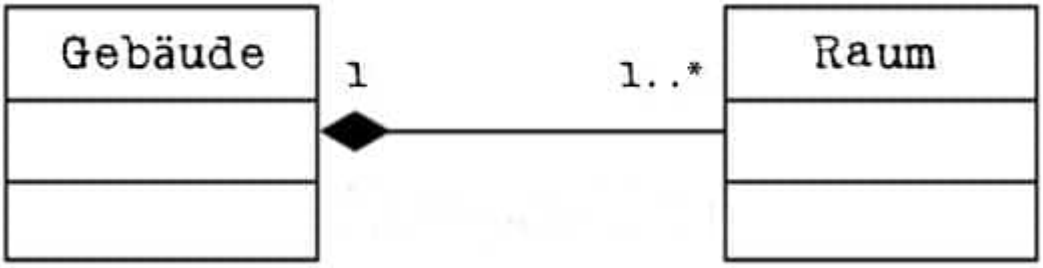
Hier erzeugen wir einen mit Futter gefüllten Fressnapf.

Das Futter erzeugen wir im Argument des Fressnapf Konstruktors, weshalb das Futter in dem für den Fressnapf reservierten Speicherbereich liegt.

Was passiert also mit dem Futter, wenn wir die `napf` Instanz löschen?

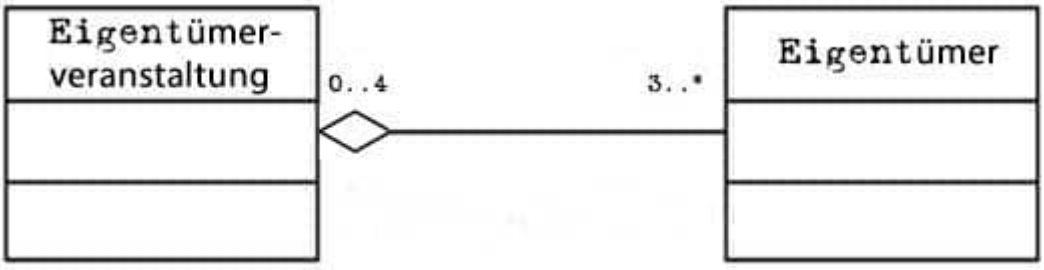
Korrekt! Mit dem Napf verlieren wir auch das Futter, weshalb es sich in diesem Fall um eine Komposition handelt.

Übungen zur Unterscheidung der Komposition und Aggregation



Warum handelt es sich hier um eine Komposition?

Ein Beispiel: Unter der Voraussetzung, dass mindestens 3 Eigentümer an einer Eigentümerversammlung teilnehmen, findet diese maximal 4 Mal pro Jahr statt. Die Eigentümer sind also ein Teil der Eigentümerversammlung. Die Mulitplizität der Eigentümerveranstaltung ist 0..4, die der Eigentümer 3..*.



Warum handelt es sich hier um eine Komposition?