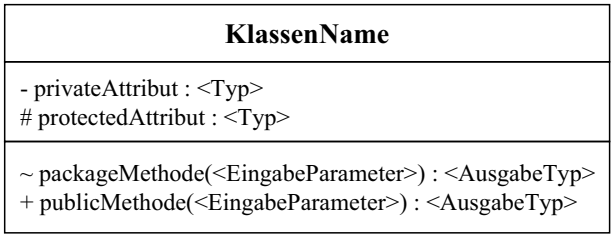
Überblick

Von UML zu Java

- UML wird meist als Entwurfsformalismus genutzt.
- Bei größeren Programmen und Softwaresystemen ist eine systematische Entwurfsphase unabdingbar, in der UML-Entwürfe schrittweise verfeinert werden (*Software-Engineering*).
- Wie kann man aus einem UML-Diagramm Java-Code generieren?

Von UML zu Java: Klassen

Gegeben ein Klassendiagramm:

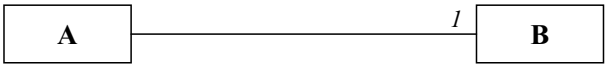


Umsetzung in Java:

```
public class KlassenName
{
  private <Typ> privateAttribut;
  protected <Typ> protectedAttribut;

  <AusgabeTyp> packageMethode(<EingabeParameter>) { ... }
  public <AusgabeTyp> publicMethode(<EingabeParameter>) { ... }
}
```


Von UML zu Java: Assoziationen

- Allgemeine Assoziationen (Verwendungsbeziehungen) werden in Java implizit modelliert:
- Einfachste Form: Objekte der Klasse *A* benutzen jeweils 1 Objekt der Klasse *B*.



- Umsetzung: Die Definition der Klasse *A* enthält ein Attribut vom Typ *B*.
- Aggregation und Komposition sind Spezialfälle der allgemeinen Assoziation.
- Bei einer Komposition muss der Programmierer zusätzlich sicherstellen, dass das entsprechende Attribut nicht `null` ist (z.B. durch geeignete Konstruktoren).

Von UML zu Java: Assoziationen

- Frage: wie kann man Assoziationen mit höheren Multiplizitäten umsetzen?
- Antwort: analog zu vorher werden Attribute in den jeweiligen Klassen eingeführt, die eine *Menge* von Objekten der jeweils anderen Klasse verwaltet.
- Wir benötigen dazu also eine Datenstruktur, die Mengen von Objekten verwaltet.
- Probleme mit Arrays:
  - Wie kann ich bei einer Multiplizität *n..m* dafür sorgen, dass immer mindestens *n* aber nur maximal *m* Objekte im Array vorhanden sind?
  - Analog: wie kann ich das Array allgemein dynamisch halten (bei der Multiplizität * können z.B. beliebig viele Objekte verwaltet werden)?
  - Wie kann ich eine Sortierung effizient sicherstellen ({ordered}-Zusatz)?
  - Wie kann ich sicherstellen, dass in dem Array nur verschiedene Objekte enthalten sind ({unique}-Zusatz)?

Von UML zu Java: Vererbung

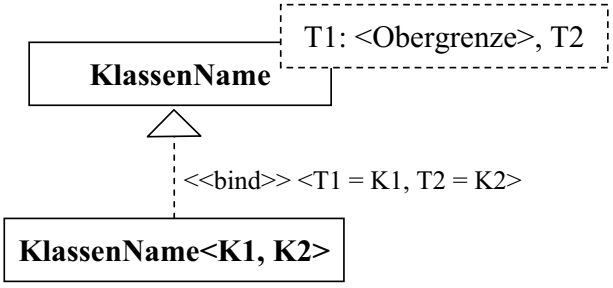
- Die Vererbungsbeziehung zwischen Klasse *A* (Vaterklasse) und Klasse *B* (abgeleitete Klasse) wird in Java mit dem Schlüsselwort **`extends`** spezifiziert.
- Genauer: der Kopf der Klassendefinition von *B* enthält das Statement **`public class`** `B` **`extends`** `A`.
- Natürlich sind überall, wo Objekte der Klasse *A* erlaubt sind (z.B. Variablen oder Parameter vom Typ `A`), auch Objekte der Klasse *B* erlaubt.

Von UML zu Java: Schnittstellen

- Schnittstellendefinitionen sind in Java ähnlich wie Klassendefinitionen, mit dem Unterschied, das anstelle von **`class`** das Schlüsselwort **`interface`** im Deklarationskopf verwendet wird.
- Die Sichtbarkeiten der Methoden sind automatisch **`public`** (d.h. andere Sichtbarkeiten darf man gar nicht angeben) und können daher auch weggelassen werden.
- Interfaces können in Java zusätzlich statische Konstanten definieren (die auch implizit **`public`** sind).

Von UML zu Java: generische Typen

Gegeben:



Umsetzung in Java:

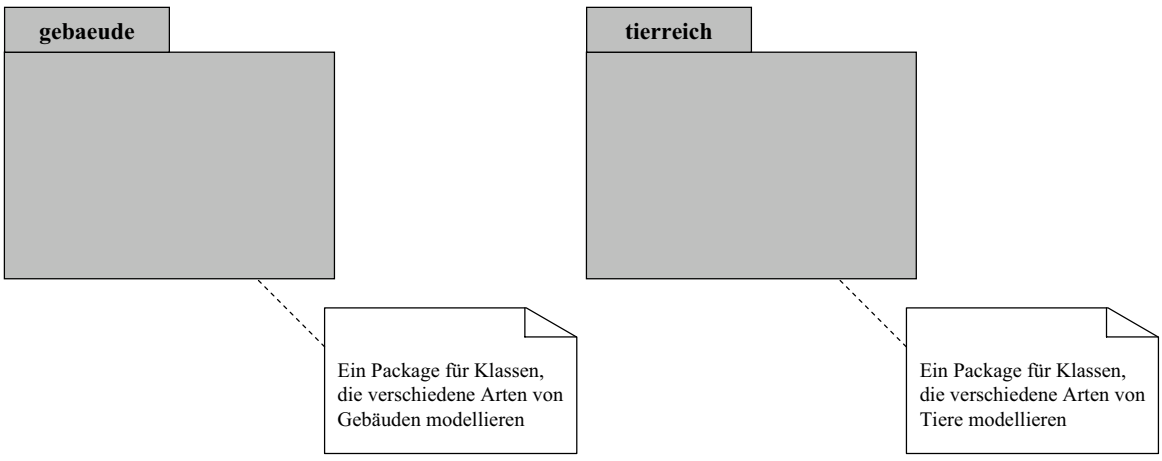
```
public class KlassenName <T1 extends <OberGrenze>, T2>
{
  ...
}

// an einer anderen Stelle:
  KlassenName<K1,K2> ...
```

Von UML zu Java: Pakete

- Alle Klassen und Interfaces, die einem speziellen Paket *P* in UML zugeordnet sind, werden natürlich auch in Java in einem Package zusammengefasst.
- Dazu werden alle betroffenen Klassen- und Interfacedefinitionen mit der Anweisung `package p;` eingeleitet.

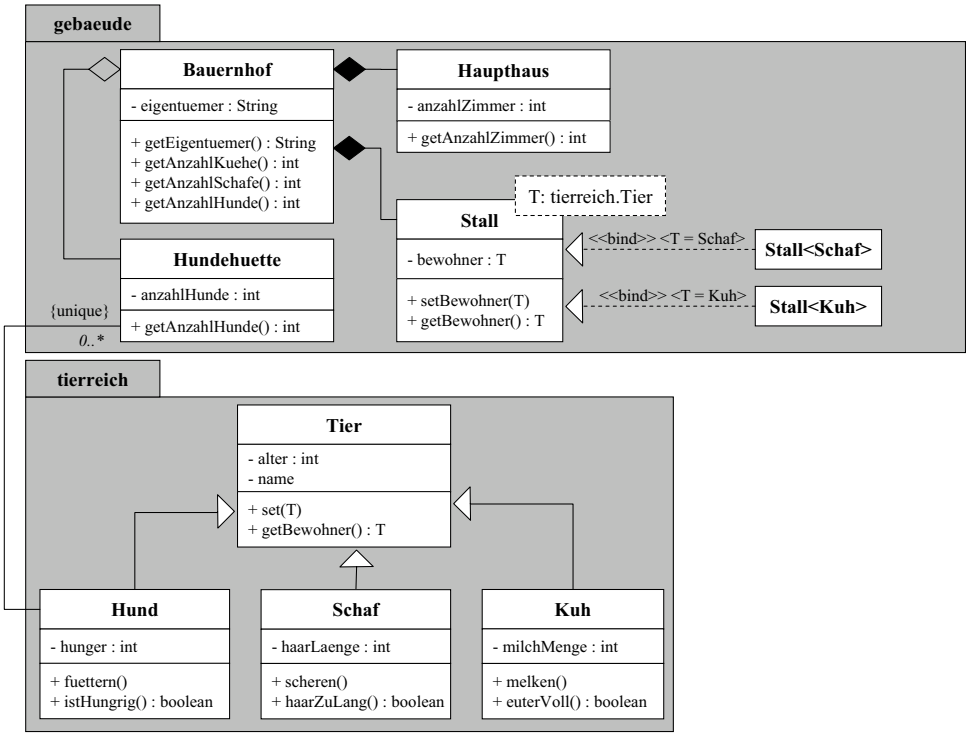
Ein ausführliches Beispiel

Gegeben ist folgendes UML-Diagramm:



Ein ausführliches Beispiel

Etwas konkreter modelliert:



Ein ausführliches Beispiel

Klasse Bauernhof in Java:

```
package gebaeude;
import tierreich.Kuh;
import tierreich.Schaf;

public class Bauernhof
{
    private String eigentuemer;

    private Stall<Kuh> kuhstall;

    private Stall<Schaf> schafstall;

    private Hundehuette hundehuette;

    private Haupthaus haus;

    ...
```

Ein ausführliches Beispiel

Klasse Bauernhof in Java:

```
    ...
    public String getEigentuemer()
    {
        ...
    }
    public int getAnzahlKuehe()
    {
        ...
    }
    public int getAnzahlSchafe()
    {
        ...
    }
    public int getAnzahlHunde()
    {
        ...
    }
}
```


Überblick

Zusammenfassung

Sie kennen jetzt

- Grundzüge von Klassendiagrammen in UML als Formalismus zur oo Modellierung,
- die Umsetzung dieser Grundzüge in Java-Code.

Insbesondere haben wir nocheinmal die Grundideen der oo Modellierung diskutiert:

- Abstraktion
- Kapselung
- Wiederverwendung
- Beziehungen
- Polymorphismus