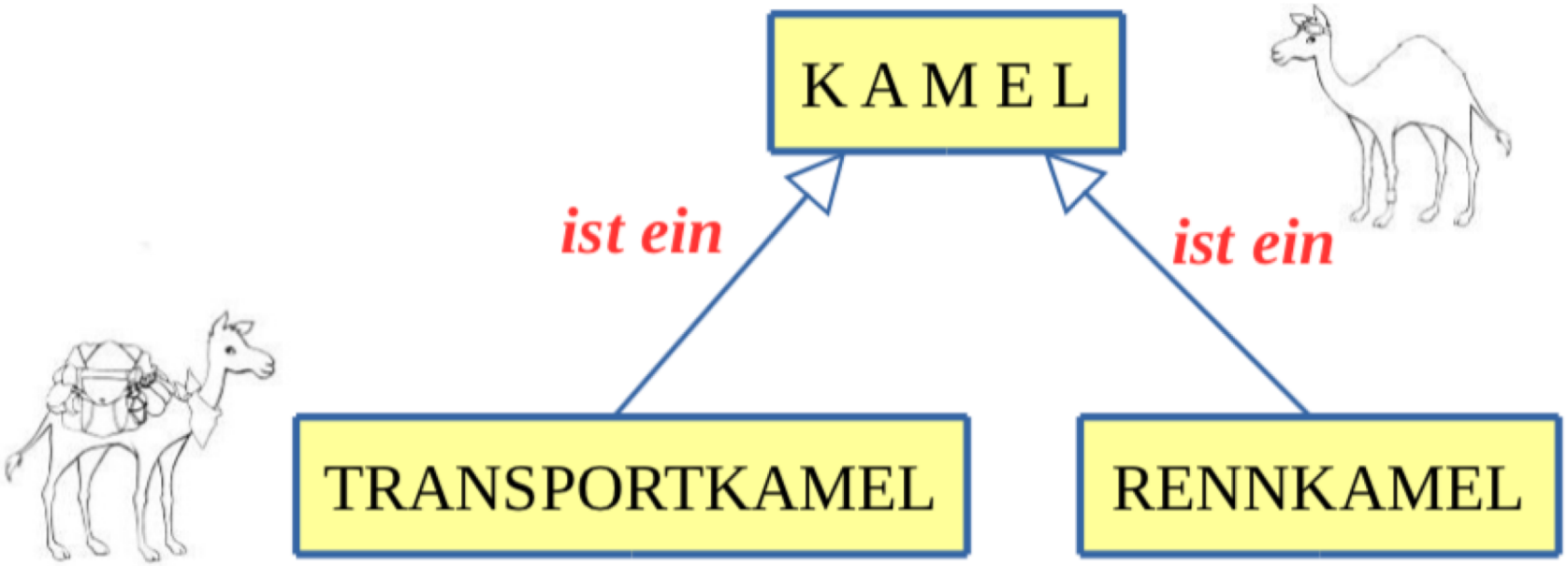
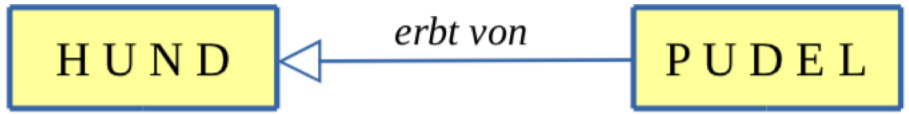
9. Vererbung und Polymorphie

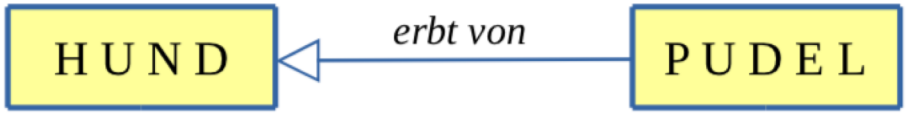


Vererbung bedeutet, dass eine Klasse Attribute und Methoden an eine andere Klasse weitergeben kann.

Im Klassendiagramm stellt man dies durch einen durchgezogenen Pfeil mit nicht ausgefüllter Spitze dar:



Die Klasse HUND wird durch die Klasse PUDEL erweitert.
Die Klasse, von der geerbt wird (hier: HUND) nennt man **Superklasse**, die erbende Klasse (hier: PUDEL) nennt man **Subklasse**.



Man sagt auch: Die Klasse PUDEL ist eine **Spezialisierung** von HUND, die Klasse HUND ist dagegen eine **Generalisierung** von PUDEL.
Deshalb nennt man die Vererbung auch eine **„ist-ein-Beziehung“.**
Ein Pudel *ist ein* Hund.

Die Subklasse **erbt alle Attribute und Methoden der Superklasse**.
Aus der Subklasse heraus kann man aber nur auf öffentliche Attribute und Methoden zugreifen.

Eine Subklasse kann geerbte Methoden **überschreiben**.
Das bedeutet, dass man die Funktion dieser Methode neu gestaltet.

MERKE

Umsetzung in Java:

```
public class BALL extends KREIS {
```

Klasse BALL erbt von Klasse KREIS

```
  String besitzer;
```

Deklaration eines neuen Attributs

```
  public BALL() {
    super();
  }
```

Konstruktor 1 von BALL: Mit super() ruft man den Konstruktor KREIS() der Oberklasse auf.

```
  public BALL(int rNeu) {
     super(rNeu);
     this.besitzer = "Hans";
  }
```

Konstruktor 2 von BALL: Aufruf eines Konstruktors der Oberklasse; Initialisierung des neuen Attributs

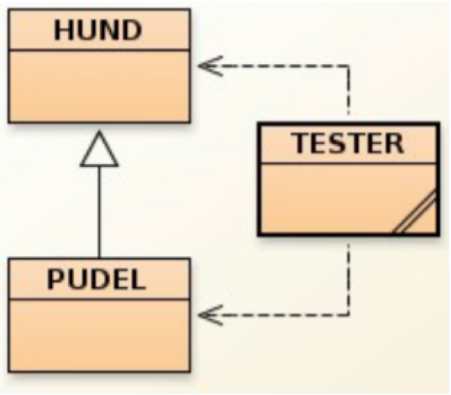
...

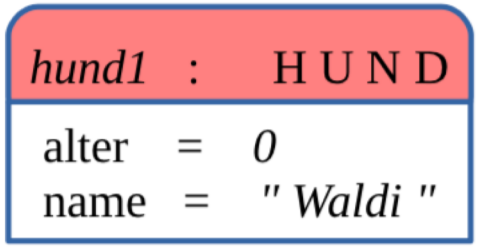
Übung 1

Erstelle ein neues BlueJ-Projekt, nenne es Vererbung_Tiere.

a)
Schreibe die Klasse HUND und
deklariere und initialisiere die Attribute wie in der Objektkarte. Setze alle Attribute auf **private.**

Außerdem soll es eine Methode
public void setzeAlter(int alterNeu) geben.



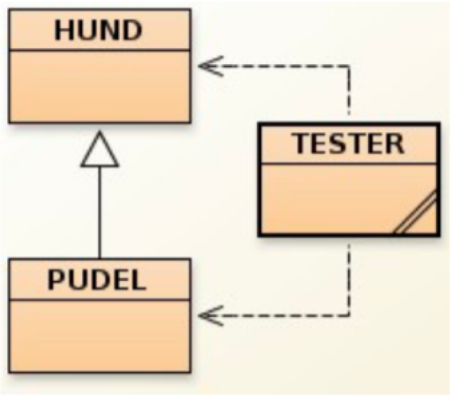


Übung 1

b)
Schreibe nun die Klasse PUDEL, die von HUND erbt.

Erzeuge ein Objekt der Klasse PUDEL und
betrachte die Attributwerte im Objektinspektor.

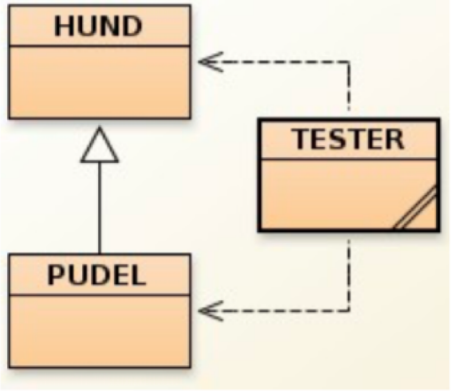
Führe bei offenen Objektinspektor die Methode
setzeAlter(...) aus und prüfe, was geschieht.



Übung 1

c)

Überschreibe nun in der Klasse PUDEL
die Methode setzeAlter(...) wie folgt:



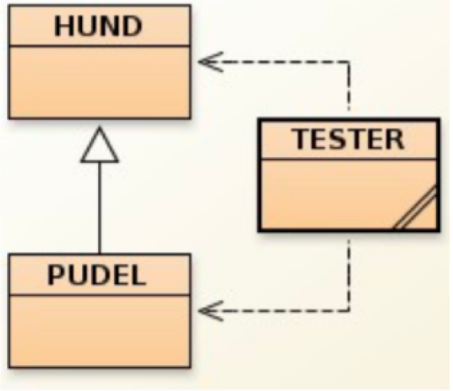

```
@Override
public void setzeAlter(int alterNeu){
    if(alterNeu > alter){
        alter = alterNeu;
    }
}
```

Weshalb erscheint eine Fehlermeldung vom Compiler?

Übung 1

d)

Schreibe in der Klasse HUND eine Methode **public int nenneAlter()**



und ändere in PUDEL die überschriebene Methode **setzeAlter(...)** so ab, dass keine Fehlermeldung mehr erscheint.

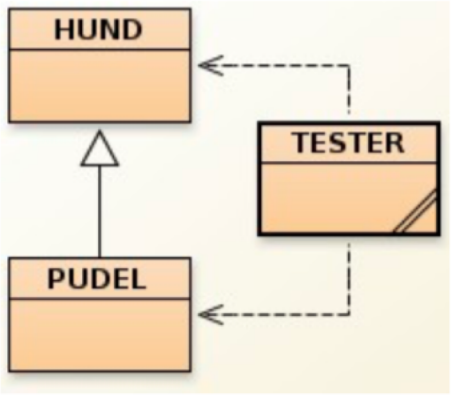
Erzeuge ein Objekt von PUDEL und teste die Methode **setzeAlter(...).**

Übung 1

e)

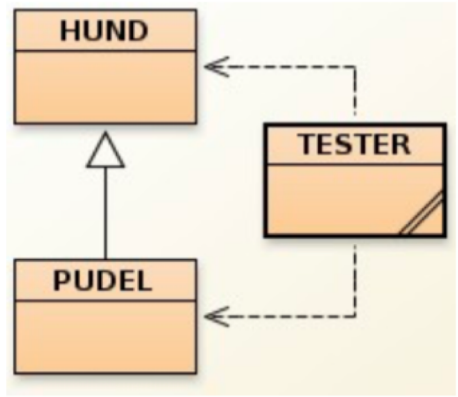
Deklariere in der Klasse PUDEL ein zusätzliches Attribut **dressiert** vom Typ **boolean.**
Setze es auf private.

Schreibe außerdem die zugehörigen setze- und nenne-Methoden für dieses Attribut.



Übung 1

f)
Schreibe nun eine Klasse TESTER, welche über drei Referenzattribute verfügt:




```
public class TESTER{
   private HUND  hund1;
   private PUDEL hund2;
   private HUND  hund3;

   public TESTER(){
      this.hund1 = new HUND();
      this.hund2 = new PUDEL();
      this.hund3 = new PUDEL();
   }
}
```

Sieh dir die dritte Initialisierung genau an und überlege, weshalb diese Konstruktion in dem Konzept der Vererbung möglich ist.

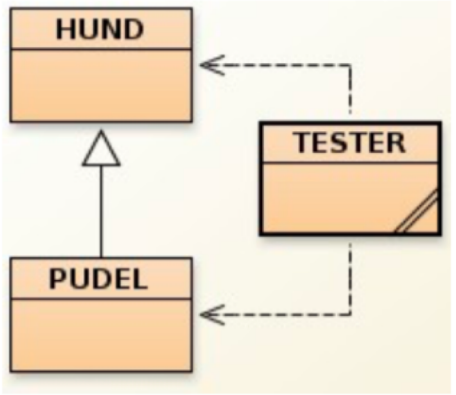
Übung 1

g)
Versuche nun, die folgenden Fragen erst mündlich zu beantworten.

Teste dann deine Vorhersagen, indem du in der Klasse TESTER
eine Methode **test()** mit dem entsprechenden Code schreibst
und die Objekte hund1, hund2 und hund3 genau im Objektinspektor beobachtest.

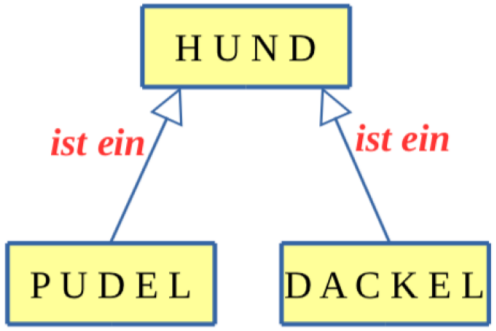
- **Wie reagieren die Objekte hund1, hund2 und hund3 jeweils auf Aufrufe der Methode setzeAlter(...) bei negativen Werten des Übergabeparameters?**

- **Wie reagieren die Objekte hund1, hund2 und hund3 jeweils auf Aufrufe der Methode nenneDressiert() ?**



MERKE

Ein Pudel oder ein Dackel kann im Konzept der Vererbung offensichtlich auch allgemein als Hund betrachtet werden.
Die Übung hat gezeigt, dass einem Referenzattribut des Supertyps (HUND) auch ein Objekt des Subtyps (PUDEL) zugewiesen werden kann.



Unter dem Begriff **Polymorphie** versteht man, dass sich Objekte abhängig von ihrer Verwendung unterschiedlich darstellen können.

Beispiel:
hund1 hat den Typ HUND und führt die Methode setzeAlter(...) in der ursprünglichen Form (ohne if) aus.
hund2 hat den Typ PUDEL und führt die Methode setzeAlter(...) in der spezielleren Form (mit if) aus.
hund3 hat den Typ HUND, wird aber als Objekt von PUDEL erzeugt und führt deshalb die Methode setzeAlter(...) auch in der spezielleren Form (mit if) aus.

MERKE

Gibt es Methoden der Superklasse, die in einer Subklasse überschrieben worden sind, so wird bei einem Objekt immer diejenige Variante ausgeführt, welche in der Vererbung an weitesten unten anzutreffen sind.

Beispiel:

private HUND hund3 = new PUDEL()

setzeAlter(...) wird hier immer in der Variante der Klasse PUDEL ausgeführt.

Diese Tatsache fasst man unter dem Begriff **dynamische Bindung** zusammen und meint damit, dass im Zweifelsfall immer die speziellere Variante einer Methode ausgeführt wird.

MERKE

Verfügt allerdings die Subklasse über eine neue Methode, die in der Superklasse noch nicht enthalten ist, so kann diese Methode nur ausgeführt werden, wenn der Typ dazu passt.

Beispiel:

nenneDressiert()

kann bei hund1 und hund3 nicht ausgeführt werden, da deren Typ HUND und nicht PUDEL ist, auch wenn hund3 sogar direkt als Objekt der Klasse PUDEL erzeugt wurde. Der Typ von hund3 bleibt aber dennoch HUND.

Bei hund3 könnte man jedoch durch einen sogenannten **Typ-Cast** erzwingen, dass Methoden des Typs PUDEL trotzdem angewandt werden können:

((PUDEL)hund3).nenneDressiert()

Übung 1

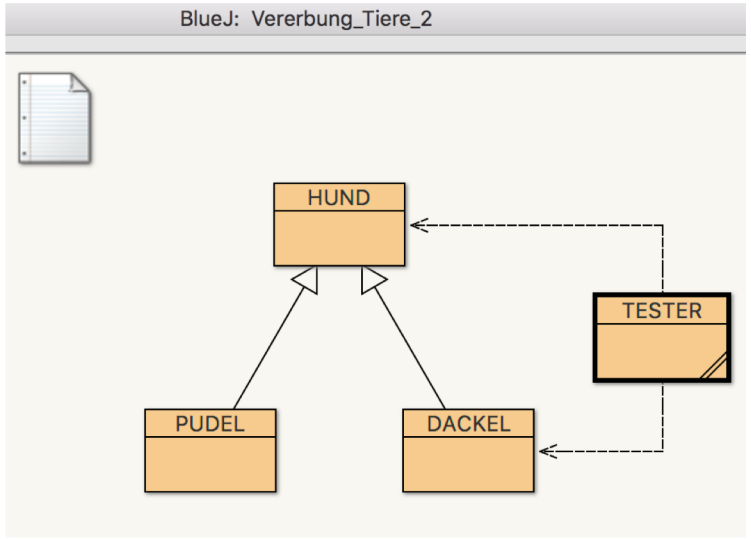
h)
Schreibe in der Klasse HUND zwei Methoden **gibLaut()**:

public void gibLaut() schreibt "wau" in ein Textfenster.

public void gibLaut(int zahl) schreibt z.B. für zahl = 4 "wau wau wau wau" in ein Textfenster.

Beide Methoden dieser Klasse haben denselben Namen.
Der Compiler unterscheidet die Methoden anhand der Anzahl und der Datentypen der Parameter.

Man spricht in diesem Fall vom
Überladen einer Methode.



Übung 1

i)
Schreibe eine Klasse DACKEL, die von HUND erbt.

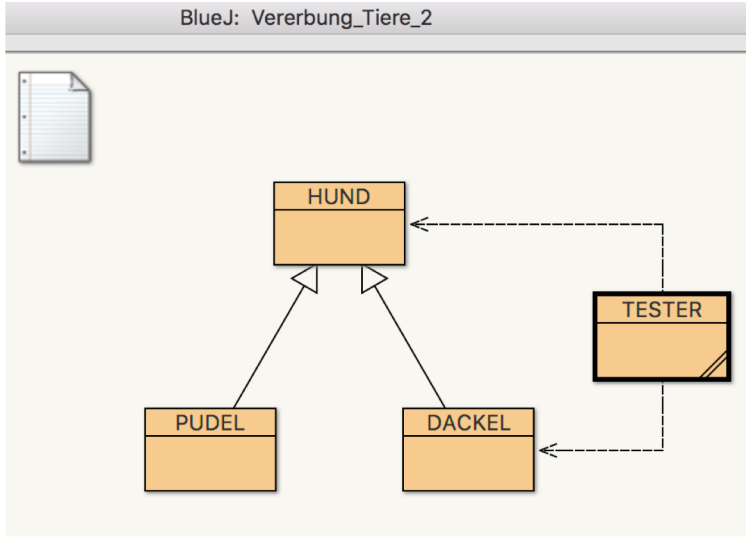
Ein Dackel macht “kläff“ statt “wau“.
Überschreibe die Methode **public void gibLaut() .**

Deklariere in TESTER das Objekt HUND hi.
Initialisiere es als **hi = new DACKEL()**.

Was liefern die folgenden Aufrufe?

hi.gibLaut()

hi.gibLaut(3)



Übung 2

Arbeite das Projekt Kamele mithilfe der Anleitung durch.