7 Vererbung und Klassendiagramme

Bisher verwendeten wir nur die **has-a**-Relation *(Aggregation)*. Aggregation in der Biologie bedeutet *"Zusammenschluss von Teilchen"*. Dementsprechend meinen wir mit Aggregation in der OOP nichts anderes als dass eine Klasse vorsieht Objekte (anderer) Klassen zu beinhalten. Das äussert sich darin, dass in der Klasse Referenzen auf Objekte der aggregierten Klassen stehen.

Listing 1: Mensch.java

```
1 public class Mensch
2 {    Kopf k = new Kopf();
3 }
```

Die Aggregation im nebenstehenden Beispielcode spricht sich als: "Mensch has-a Kopf". Wenn die Klasse `Kopf` z. B. die Methode "`denken`" anbietet, können wir diese in `Mensch` nun durch `k.denken();` verwenden. Man spricht deswegen von **Wiederverwendung** der Klasse `Kopf` mittels Aggregation.

Mit der **Vererbung** kommt nun die **is-a**-Relation *(Generalisierung)* hinzu. Die Vererbung dient dazu, **aufbauend auf existierenden Klassen neue zu schaffen**. Wie bei der Aggregation resultiert dadurch eine **Wiederverwendung** der existierenden Klasse. Neben diesem konstruktiven Aspekt bedeutet Vererbung jedoch **Ähnlichkeiten** zwischen Klassen. Deswegen spricht man von einer **is-a**-Beziehung. So **ist** der Mensch **kein** `Kopf`, aber der `Mensch` **ist ein** `Lebewesen`. Zwischen `Mensch` und `Lebewesen` können wir also die **Vererbung** einsetzen.

Beispiele für **Vererbung** gibt es viele:

a) `Mensch` *"is a"* `Lebewesen`

b) `Eiche` *"is a"* `Baum`

c) `Tanne` *"is a"* `Baum`

d) `Baum` *"is a"* `Pflanze`

e) `Pflanze` *"is a"* `Lebewesen`

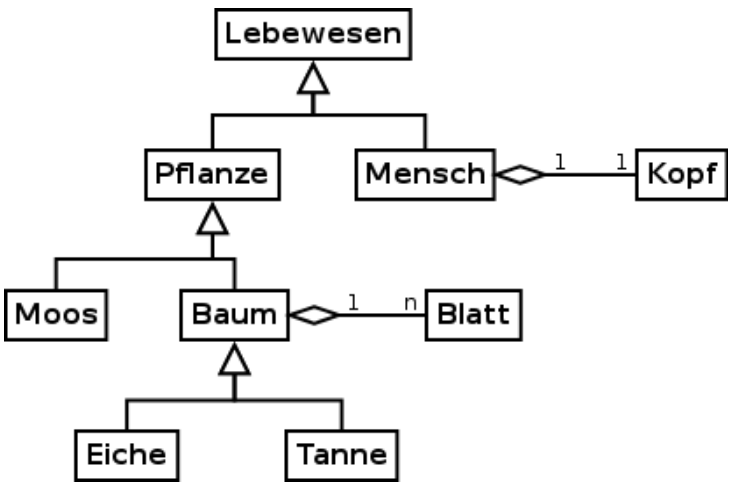
f) `Moos` *"is a"* `Pflanze`

Beispiele für **Aggregation** wären:

g) `Mensch` *"has a"* `Kopf`

h) `Baum` *"has"* `Blätter`
(Tannennadeln sind biologisch auch Blätter)

Die Angaben a) bis h) in ein **Klassendiagramm** gefasst, ergibt folgendes Bild:



Das Design spiegelt sich auch im Code *Listing2*[1]. Zur Vererbung wird das Schlüsselwort "`extends`" verwendet *(sprich, die neue Klasse "erweitert" die Existierende)*.

[1] Öffentliche Klassen müssen stets in einer eigenen Java-Datei stehen.

Listing 2: Projekt Lebewesen

```
1  public class Lebewesen { /* ... */ }
2  public class Pflanze extends Lebewesen { /* ... */ }
3  public class Mensch extends Lebewesen // inherit
4  {   Kopf k = new Kopf(); // aggregate
5      /* ... */
6  }
7  public class Moos extends Pflanze { /* ... */ }
8  public class Baum extends Pflanze
9  {   LinkedList<Blatt> blaetter = new LinkedList<Blatt>();
10     /* ... */
11 }
12 public class Eiche extends Baum { /* ... */ }
13 public class Tanne extends Baum { /* ... */ }
```

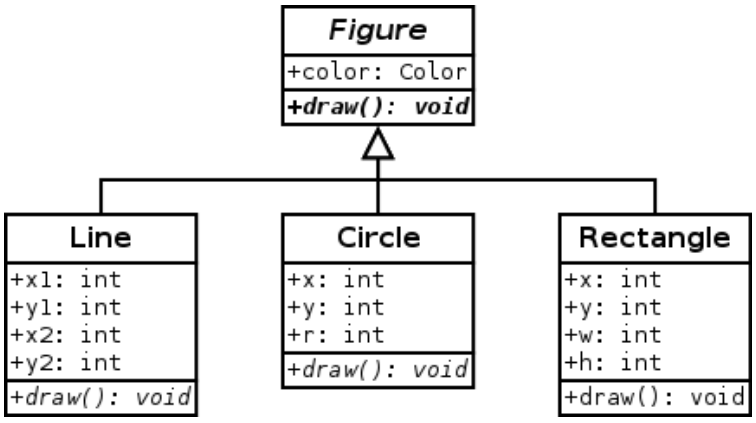
7.1 Fahrzeuge

Zeichnen[2] Sie ein Klassendiagramm analog zum Beispiel mit den Lebewesen mit allen zwölf Elementen aus der nachfolgenden Liste. Verwenden Sie **Vererbung** *(is-a)* und **Aggregation** *(has-a)* möglichst sinnvoll. Bei allen Elementen in der Liste handelt es sich um **Klassen**. Sie müssen keine Methoden oder Felder/Eigenschaften/Variablen notieren.

- Motor
- VW
- Velo
- Airbag
- Pneu
- Töff
- Kraftfahrzeug
- Fahrzeug
- Automobil
- Rad
- Toyota
- Felge

7.2 Geometrische Figuren

Erstellen Sie ein neues Java-Projekt mit den vier Klassen in der dem Bild entsprechenden **Verwandschaft**. Füllen Sie die `draw`-Methoden mit einfachen Textausgaben. Testen Sie Ihre Implementation.



7.3 Figuren Zeichnen

Freiwillig: Wer Textausgaben nicht interessant genug findet, ist **freundlich eingeladen**, die Elemente tatsächlich zu zeichnen. Als Geschenk von uns finden Sie auf der Kurswebsite die Klasse `EasyGraphics`. Bauen Sie sich ein Objekt dieser Klasse und übergeben Sie dieses Objekt stets den `draw`-Methoden als Parameter. Mit dem `EasyGraphics`-Objekt können Sie auf einfachste Weise Pixel ins Grafikfenster zeichnen (z. B. `graph.set(10,10,Color.BLUE.getRGB());`). Am Ende der `draw`-Methoden müssen Sie jeweils das **Zeichnen der Grafik** veranlassen (`graph.repaint();`).

[2] Wenn Sie ihr Diagramm nicht von Hand zeichnen wollen, können Sie z. B. das Gratistool Dia verwenden: http://dia-installer.de/index.html.de. Dia gibt es für Windows und Linux.