Vererbung

Ziel:

- Begriffe aus der Theorie der Vererbung erklären können.
- Sprachelemente von Java für die Umsetzung der Vererbung anwenden können.

Vererbung wieso?

Stellen wir uns ein einfaches Flugzeug vor.

Wir können uns für dieses Flugzeug Attribute und Methoden definieren.

- Flügelspannweite
- Dienstgipfelhöhe
- Maximale Fluggeschwindigkeit
- starten, landen

Wenn wir nun etwas konkreter werden, dann gibt es verschiedene Flugzeugtypen, die die Eigenschaften von Flugzeug übernehmen, dazu aber noch jeweils spezielle Eigenschaften (Attribute und Methoden) haben.

- Kanonen
- schiessen

- Kunstflugprogram
- aufführen

- Passagiere
- boarding

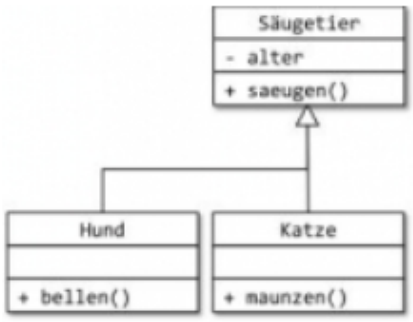
Man spricht hier von der Generalisierung und der Spezialisierung.

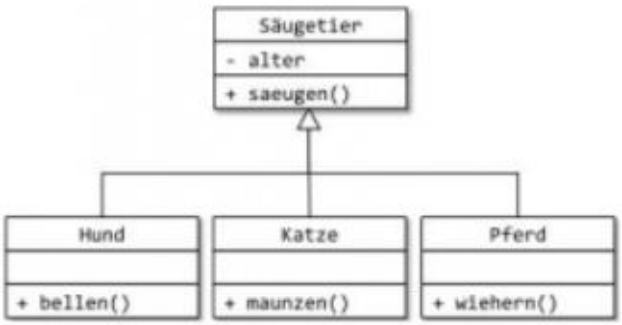
1 Generalisierung versus Spezialisierung

Wo besteht der Unterschied zwischen Generalisierung und Spezialisierung ?

Generalisierung
- vom Besonderen zum Allgemeinen
- Gemeinsamkeiten in Superklasse auslagern
- Subklassen von Superklassen ableiten (**-> Vererbung**)

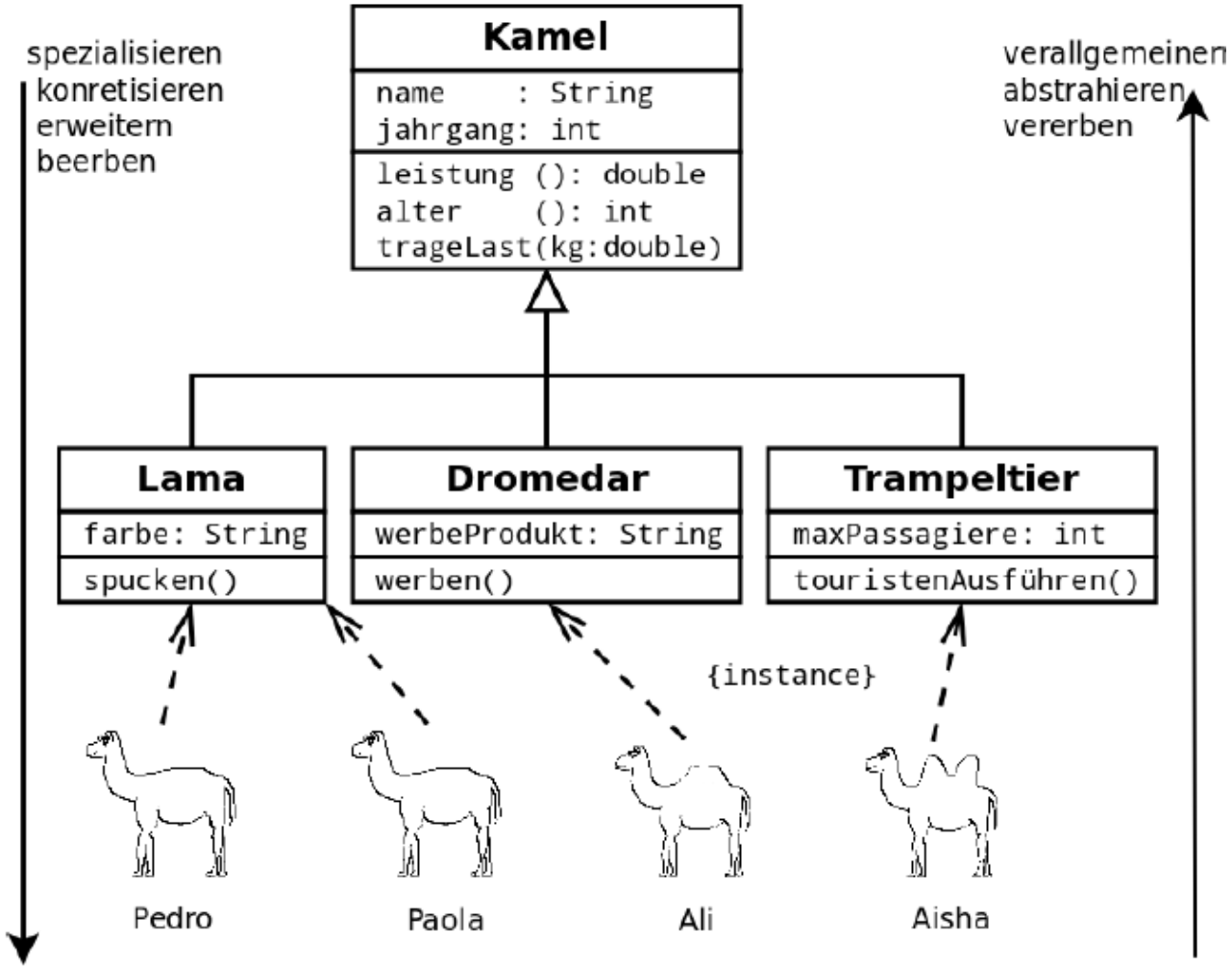
Spezialisierung
vom Allgemeinen zum Besonderen
- neue Subklassen erzeugen
- von Superlasse **erben lassen**





http://www.cobocards.com/pool/de/card/89547948/online-karteikarten-wo-besteht-der-unterschied-zwischen-generalisierung-und-spezialisierung-/

Weiteres Beispiel von Vererbung:



- Vererbung ist eine Hierarchie von Klassen.
- Die es Vererbung erlaubt der Unterklasse alle Eigenschaften (Attribute, Methoden) der Überklasse mitzuverwenden.
- Zudem kann der Unterklasse weitere Attribute oder Methoden hinzugefügt werden.

2 UML Notation

- Klasse 1 erbt von Klasse2
- Klasse 2 ist die Überklasse, Basisklasse
- Klasse 1 ist die Unterklasse, abgeleitete Klasse

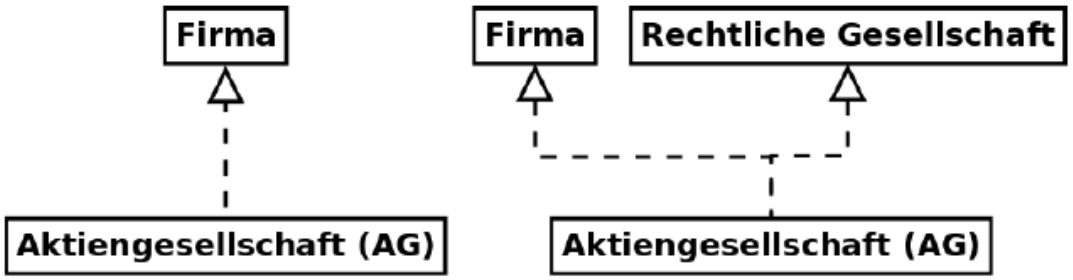
3 Bezeichnungen

Synonyme für übergeordnete Klassen (Substitution):	Vaterklassen, Basisklassen, Superklassen, Oberklassen, parent class, base class, vererbende Klasse, Überklasse, Elternklasse, Verallgemeinerung, Abstraktion
Synonyme für untergeordnete Klassen (Vererbung):	Subklassen, Unterklassen, Kindklasse, abgeleitete Klassen, child class, sub class, derived class, erbende Klasse, Spezialisierung, Konkretisierung

4 Mehrfachvererbung

Wir unterscheiden zwei Arten von Vererbungen:

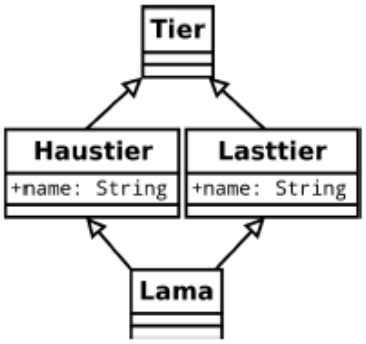
- **Single inheritance**: Die einfache Vererbung bedeutet eine Beziehung zwischen zwei Klassen (Subclass, Superclass).
- **Multiple inheritance**: Die mehrfache Vererbung bedeutet eine Beziehung von mehreren Superklassen zu einer Subklasse.



Einige Programmiersprachen kennen die Mehrfachvererbung (multiple inheritance). Dabei kann eine Klasse von mehreren Klassen erben. Das Lama könnte also erstens ein Lasttier sein, aber auch alle Attribute und Methoden des Haustiers erben.

Dieses Konzept hat aber einen Nachteil. Wenn nämlich eine Methode in beiden Superklassen überschrieben wurde, so ist oft nicht mehr klar, welche Methode denn bei der Kindklasse nun gültig ist. Da die Mehrfachvererbung häufig zu einem unverständlichen Code führte, hat JAVA diese bewusst nicht eingeführt. JAVA kennt einzig die Einfachvererbung (single inheritance).

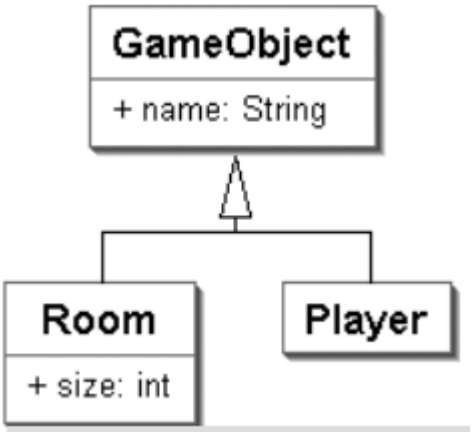
Deadly Diamond of Death



Das Problem mit obiger Klasse `Lama` ist offensichtlich das Attribut `name`, das in beiden Oberklassen vorkommt. Es ist den Programmierern oft nicht klar, welches Attribut denn genau geerbt wurde. Das obige Diagramm macht auch klar, warum die Mehrfachvererbung manchmal auch Deadly Diamond of Death genannt wird.

Quelle: www.programmieraufgaben.ch „Objekte und Klassen" oo.pdf

5 Umsetzung mit Java




```
public class GameObject
{
  public String name;
}

public class Player extends GameObject
{
}

public class Room extends GameObject
{
  public int size;
}
```

Instanziierungen

```
Room clinic = new Room();
clinic.name = "Clinic";                  // Geerbtes Attribut
clinic.size = 120000;                    // Eigenes Attribut

Player theDoc = new Player();
theDoc.name = "Dr. Schuwibscho";         // Geerbtes Attribut
```

Erklärungen

Ein ***Room* ist auch ein *GameObject*** und hat damit **Zugriff das Attribut *name***.

Aber **das Attribut muss/darf nicht nochmals bei *Room* geschrieben** werden!!

Ein *Room* kann also einfach auf die Attribute von **GameObject** und auch die Methoden von *GameObject*, zugreifen.