



# VERERBUNG



Christian Schirmer

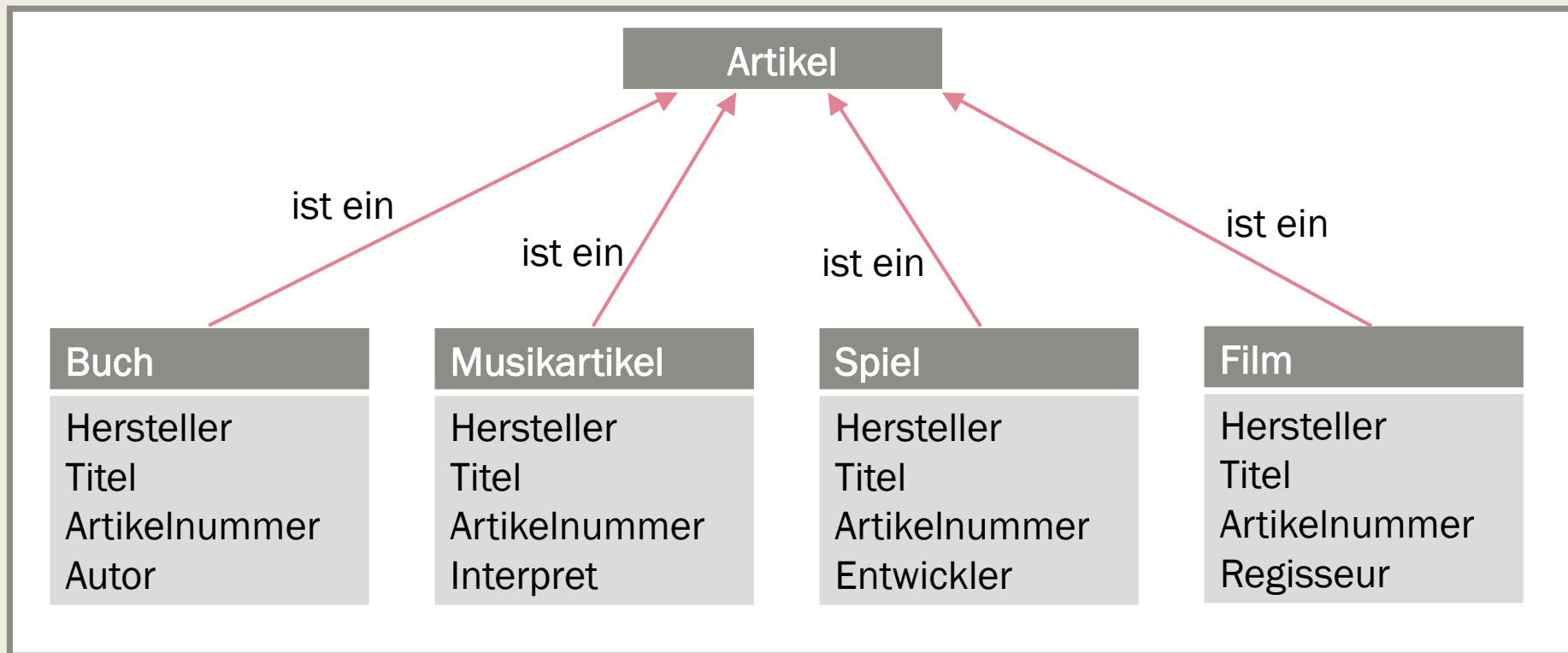
# Vererbung

- **Motivation:** Bei einigen der Klassen gleich lautende Attribute und Methoden

Buch	Musikartikel	Spiel	Film
Hersteller Titel Artikelnummer Autor	Hersteller Titel Artikelnummer Interpret	Hersteller Titel Artikelnummer Entwickler	Hersteller Titel Artikelnummer Regisseur

# Vererbung

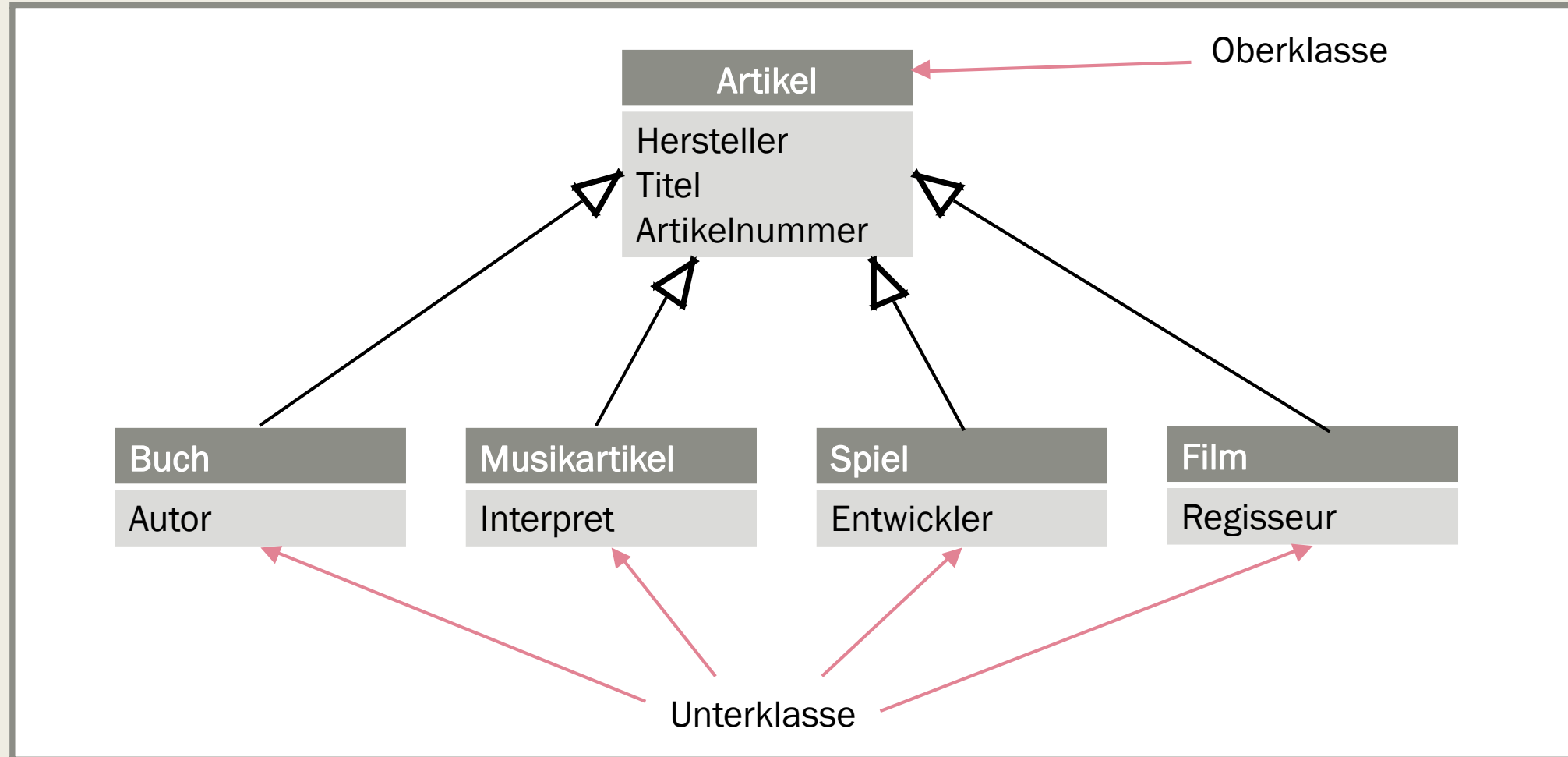
- **Klassenmodell:** Es besteht jeweils ein Beziehung zur Klasse Artikel



# Vererbung

- Die gemeinsamen Attribute werden in einer Oberklasse zusammengefasst
- Attribute werden an die verbundenen Klassen weitergeben (ohne erneute Aufführung)
- In UML Klassendiagrammen wird für die „**ist ein/e**“-Beziehung eine eigene Notation verwendet. Man zieht zwischen zwei Klassen eine Linie und zeichnet an das Ende eine geschlossene nicht ausgefüllte Pfeilspitze

# Vererbung



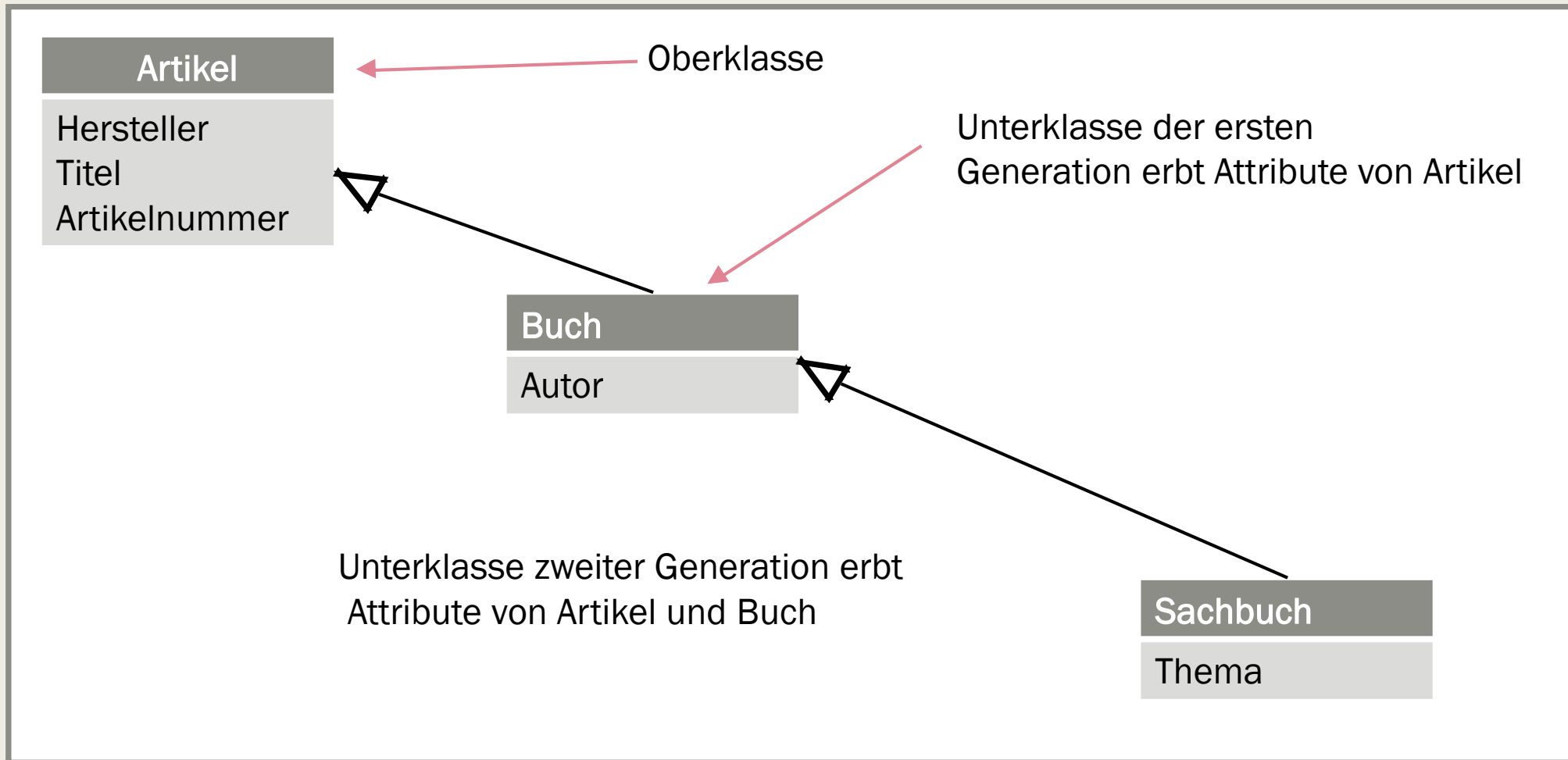
# Vererbung

- Die „ist ein/e“- Beziehung drückt aus, dass eine Klasse eine spezielle Art einer anderen Klasse ist
- Eine Klasse wird als Oberklasse bezeichnet, wenn wir andere Klassen von ihr ableiten
- Eine Unterklasse leitet von einer Oberklasse ab und erbt deren Attribute und Methoden. Eine Unterklasse spezialisiert eine Oberklasse, indem sie mit zusätzlichen Attributen und Methoden weitere Funktionalität implementiert

# Vererbung

- Die Vererbungsbeziehung wird verwendet, um speziellere Klassen einer anderen Klasse zu modellieren
- **Vererbung** ermöglicht das Zusammenfassen von Gemeinsamkeiten und reduziert damit Wiederholungen im Entwurf
- Die Vererbungsbeziehung ist transitiv, sie vererbt sich gewissermaßen mit. Eine weitere Spezialisierung der Klasse „Buch“ ist auch ein „Artikel“, da „Buch“ schon ein „Artikel“ ist

# Vererbung





# Vererbung

- Es werden vererbt:
  - *Attribute der Oberklasse an die Unterklassen.*
    - !Achtung! Eventuelles Shadowing oder private Methoden.
  - *Methoden und Assoziationen der Oberklasse*  
*Es wird nicht nur die Gestalt der Oberklasse, sondern auch ihr Verhalten und ihre Verwendung vererbt*
  - *Mit der Vererbung werden objektorientierte Systeme strukturiert*

# Vererbung

- **Mehrfachvererbung** (Ableiten von mehreren Klassen) ist in Java nicht erlaubt
- Das Fehlen der Mehrfachvererbung schränkt Java nicht ein
- Java erlaubt es mehrere Schnittstellen (Interfaces) zu implementieren und so unterschiedliche Typen (Polymorphie) anzunehmen

# KONSTRUKTOREN IN DER VERERBUNG

Christian Schirmer

# Konstruktoren in der Vererbung

- Konstruktoren haben Ähnlichkeit mit Methoden: sie können überladen werden oder Ausnahmen erzeugen
- Konstruktoren werden nicht vererbt
- Unterklasse muss neue Konstruktoren angeben – mit den Konstruktoren der Oberklasse kann ein Objekt der Unterklasse nicht erzeugt werden
- Java ruft im Konstruktor einer jeden Klasse (ausgenommen `java.lang.Object`) automatisch den Standard-Konstruktor der Oberklasse auf, damit die Oberklasse »ihre« Attribute initialisieren kann
- Aufruf des Standard-Konstruktors der Oberklasse geschieht durch den Aufruf `super()`
- Der Compiler fügt als erste Anweisung automatisch `super()` in den Konstruktor ein (Manuelles Hinschreiben daher nicht notwendig)

# Konstruktoren in der Vererbung

- `super()` darf nur als erste Anweisung im Konstruktor sein
- Beim Aufbau neuer Objekte läuft die Laufzeitumgebung als Erstes die Hierarchie nach `java.lang.Object` ab
- In `java.lang.Object` beginnt von oben nach unten die Initialisierung.
- Ist der eigene Konstruktor an der Reihe, konnten die Konstruktoren der Oberklasse ihre Werte schon initialisieren.
- Nicht nur die Standard-Konstruktoren rufen mit `super()` den Standard-Konstruktor der Oberklasse auf, sondern auch immer die parametrisierten Konstruktoren.
- Für das Aufrufen einer parametrisierten Konstruktor der Oberklasse gibt es `super()` mit Argumenten

# Konstruktoren in der Vererbung

- Gründe für den Aufruf einer parametrisierten Konstruktors der Oberklasse
  1. Parametrisierter Konstruktor der Unterklasse leitet *Argumente an die Oberklasse weiter; der Oberklassen - Konstruktor soll die Attribute annehmen und verarbeiten*
  2. *Oberklasse hat keinen Standard Konstruktor, die Unterklasse muss daher den speziellen, parametrisierten Konstruktor `super( Argument ...)` aufrufen.*

# Konstrukturen in der Vererbung


- Veranschaulichung der Notwendigkeit einer parametrisierten super ()
- Oberklasse Alien erwartet in einem parametrisierten Konstruktor den Planetennamen

```
public class Alien
{
    public String planet;
    public Alien( String planet ) { this.planet = planet; }
}
```

# Konstrukturen in der Vererbung

- Erweitert eine Klasse Grob für eine besondere Art von Außerirdischen die Klasse Alien, kommt es zu einem Compilerfehler:

```
public class Grob extends Alien { }
```

```
//  Compilerfehler
```

- Fehler vom Eclipse-Compiler ist: *"Implicit super constructor Alien() is undefined. Must explicitly invoke another constructor."*



# Konstruktoren in der Vererbung

- Grund: Grob enthält vom Compiler generierten vorgegebenen Konstruktor, der mit `super()` nach einem Standard-Konstruktor in Alien sucht – den gibt es aber nicht.
- 1. In der Oberklasse muss ein Standard-Konstruktor angelegt werden (geht nicht bei nicht modifizierbaren Klassen)
- 2. `super()` muss in Grob so eingesetzt werden, dass es mit einem Argument den parametrisierten Konstruktor der Oberklasse aufruft

# Konstrukturen in der Vererbung

- Aufruf von `super()` - parametrisierter Konstruktor der Oberklasse - mit einem Argument
- Es spielt keine Rolle, ob Grob Standard-Konstruktor oder parametrisierten Konstruktor besitzt

```
public class Grob extends Alien
{
    public Grob()
    {
        super( "Locutus" );    // Alle Grobs leben auf Locutus
    }
}
```

# Konstrukturen in der Vererbung

- Aufruf von `super()` - parametrisierter Konstruktor der Oberklasse - mit einem Argument

```
public class Grob extends Alien
{
    public Grob( String planet )
    {
        super( planet );
    }
}
```

# Konstruktoren

**können** alle Sichtbarkeitsmodifikatoren besitzen. public, protected, paketsichtbar und auch private.

**können nicht** abstract, final, native, static oder synchronized sein.

Konstuktoren besitzen keinen Rückgabewert, auch nicht void

Gleicher Name wie die Klasse. Beginnt mit einen Großbuchstaben

this ist eine Referenz in Objektmethoden und Konstruktoren, die sich auf das aktuelle Exemplar bezieht.

this() bezieht sich auf einen anderen Konstruktor der gleichen Klasse. Wird this() benutzt, muss es in der ersten Zeile stehen

# Konstruktoren

super ist eine Referenz mit dem Namensraum der Oberklasse. Damit lassen sich überschriebene Objektmethoden aufrufen.

super() ruft einen Konstruktor der Oberklasse auf. Wird es benutzt, muss es die erste Anweisung sein.

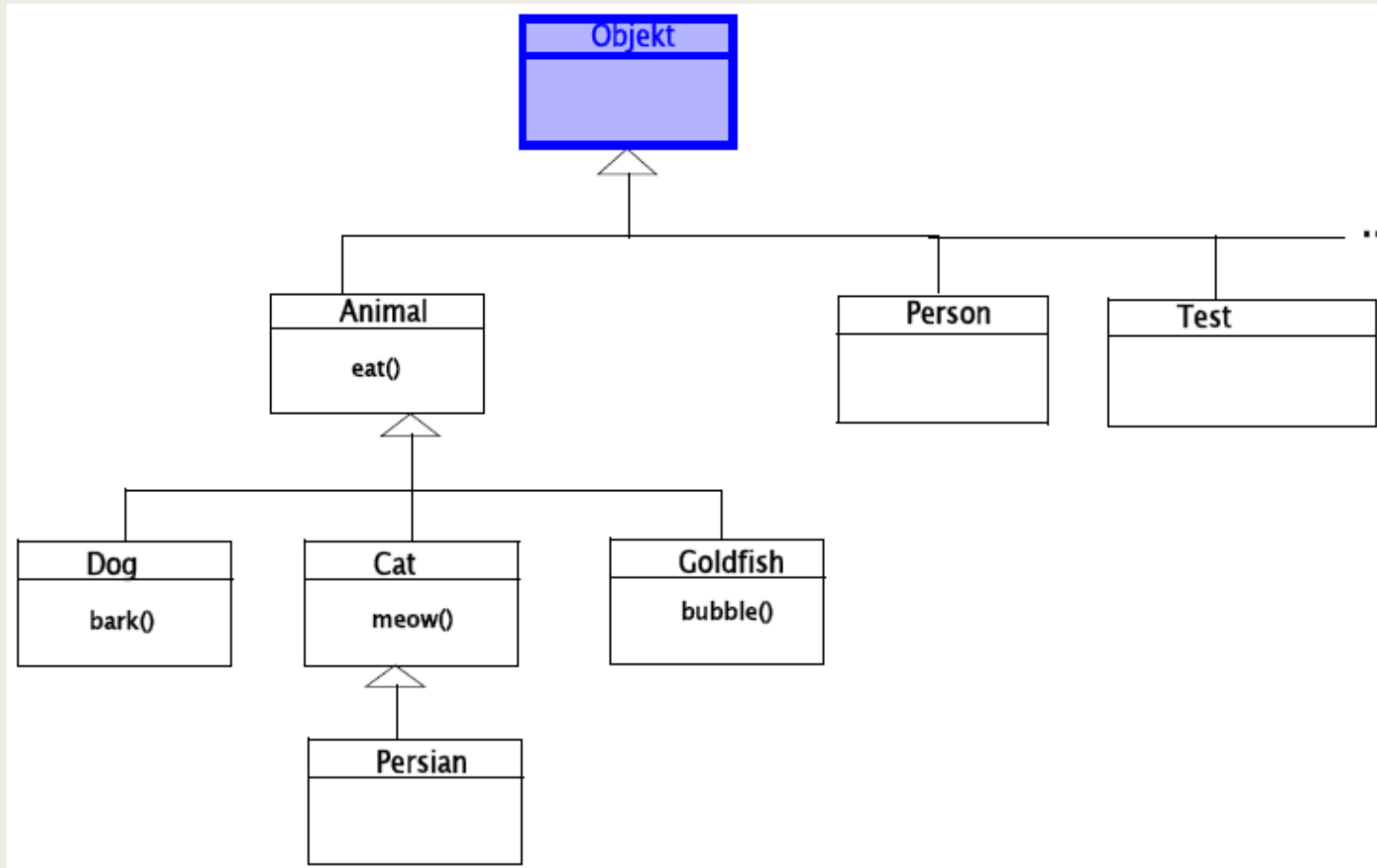
Konstruktoren werden nicht vererbt

# VERERBUNG

Christian Schirmer

# Vererbung

Christian Schirmer



# Vererbung

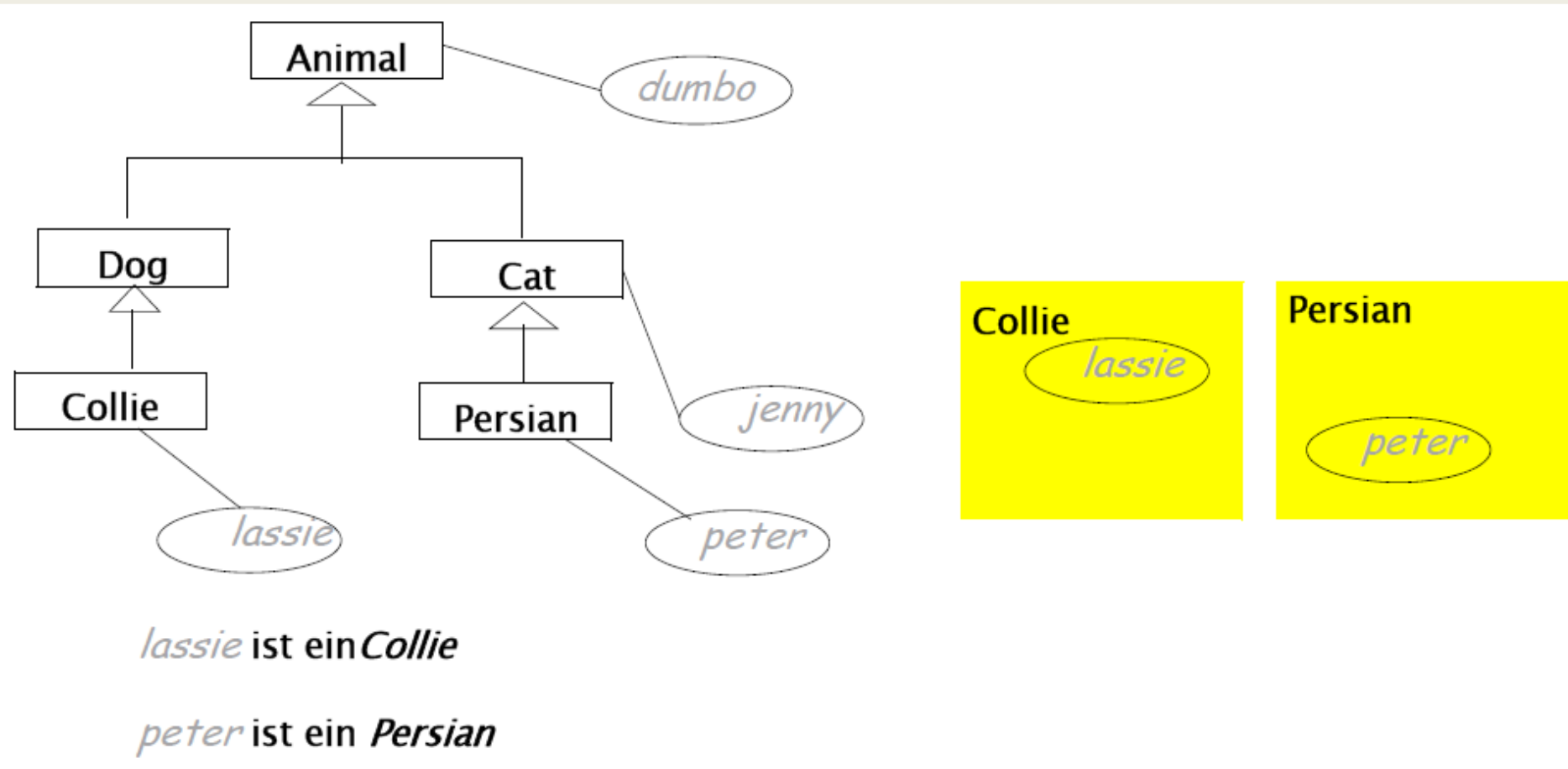
## ■ Methoden der Klasse Object

- `public boolean equals(Object o)`  
liefert true, wenn zwei Objekte gleich sind.
- `public String toString()`  
liefert den Zustand eines Objekts als String.
- `public Class getClass()`  
liefert den Laufzeittyp eines Objekts.



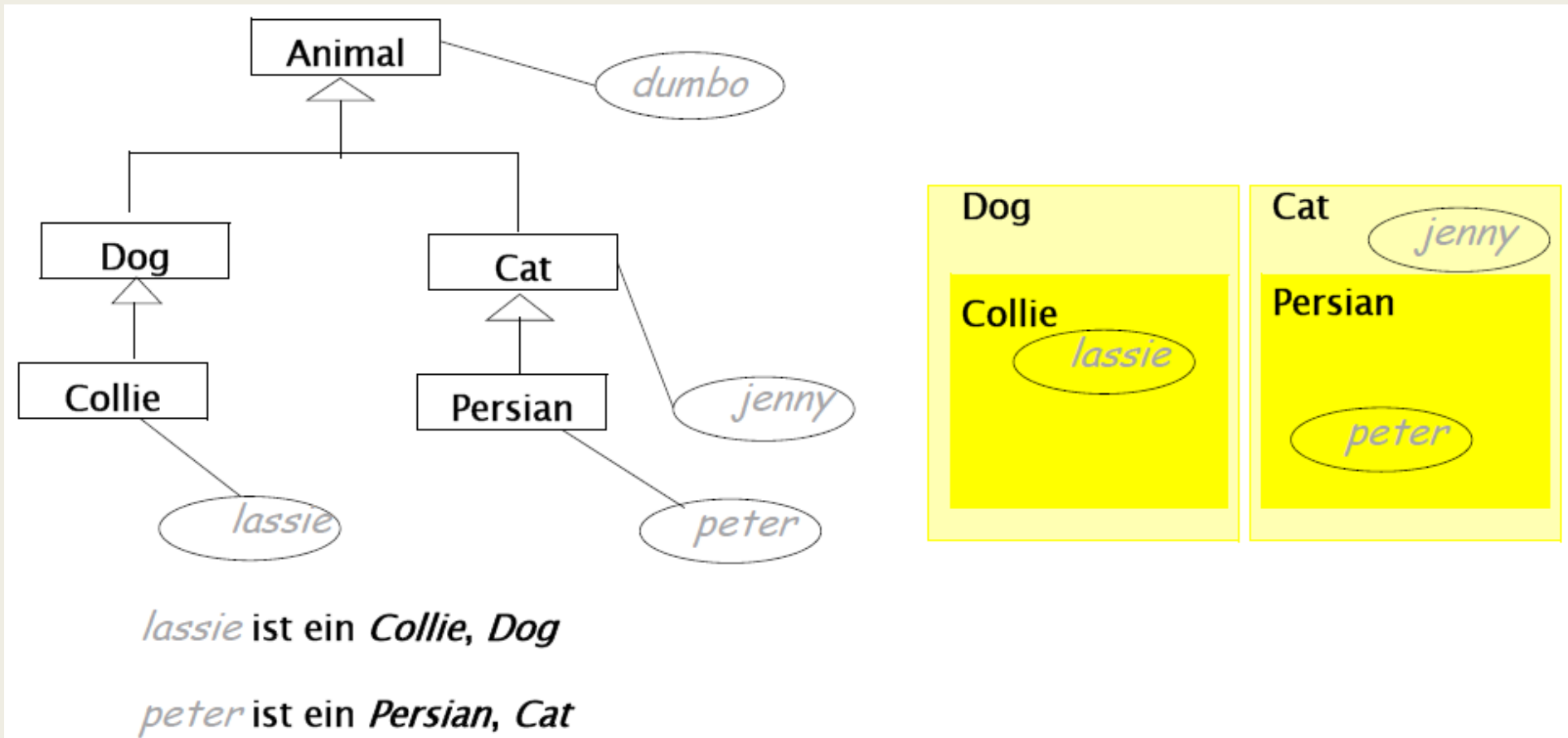
# Vererbung

- Objekt Inklusion (Objekt hat mehrere Typen)



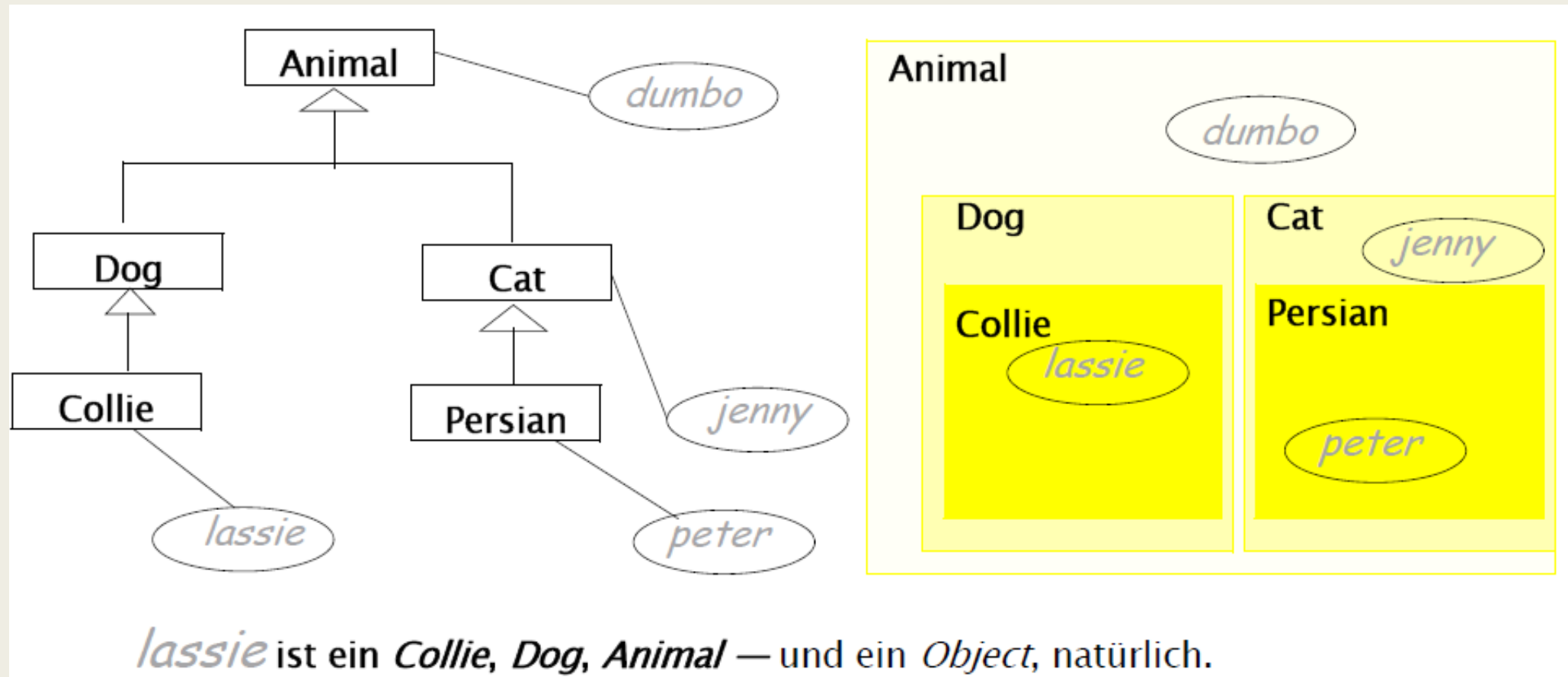
# Vererbung

- Objekte der Unterklasse sind auch Objekte der Oberklasse.



# Vererbung

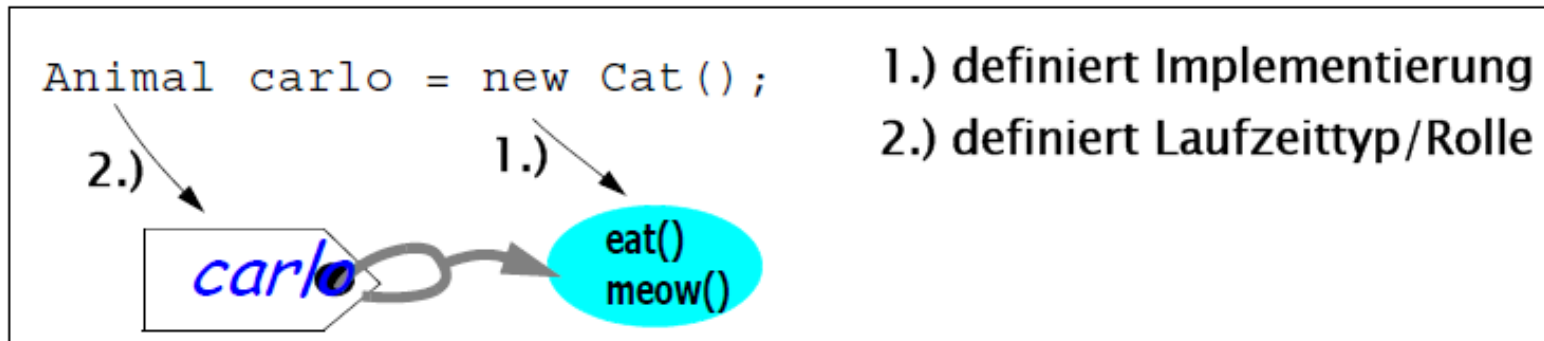
- Objekte der Unterklasse sind auch Objekte der Oberklasse.



# Vererbung

Eine Referenz kann jeden **kompatiblen** Typ annehmen. Kompatibel sind alle möglichen Supertypen und natürlich die erzeugende Klasse selbst:

```
Cat oscar = new Cat();  
Animal carlo = new Cat();  
Object minnie = new Cat();
```



Die Referenz hat einen zur Laufzeit gültigen Typ. Er definiert, welche Eigenschaften gültig sind:

```
() carlo.eat() //gültig ...  
()carlo.meow() //Problem: meow()nicht gültig
```

# Vererbung

- Java unterscheidet die erzeugende Klasse von Laufzeittyp der Referenz.
- Ein Objekt ist immer vom Typ der erzeugenden Klasse und aller ihrer Supertypen.
- Konstruktoren werden nicht vererbt
- Java startet zu Beginn jedes Konstruktors den parameterlosen Konstruktor der Superklasse (also: `super()` Aufruf).
- Mit `this()` wird ein Konstruktor derselben Klasse gestartet.
- Mit `super()` wird ein Konstruktor der Superklasse gestartet.
- `this()` oder `super()` könne nur in der ersten Zeile des Konstruktors stehen, man kann also nicht beide in einem Konstruktor verwenden.