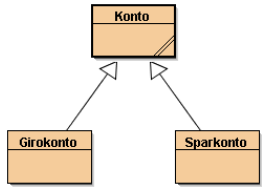
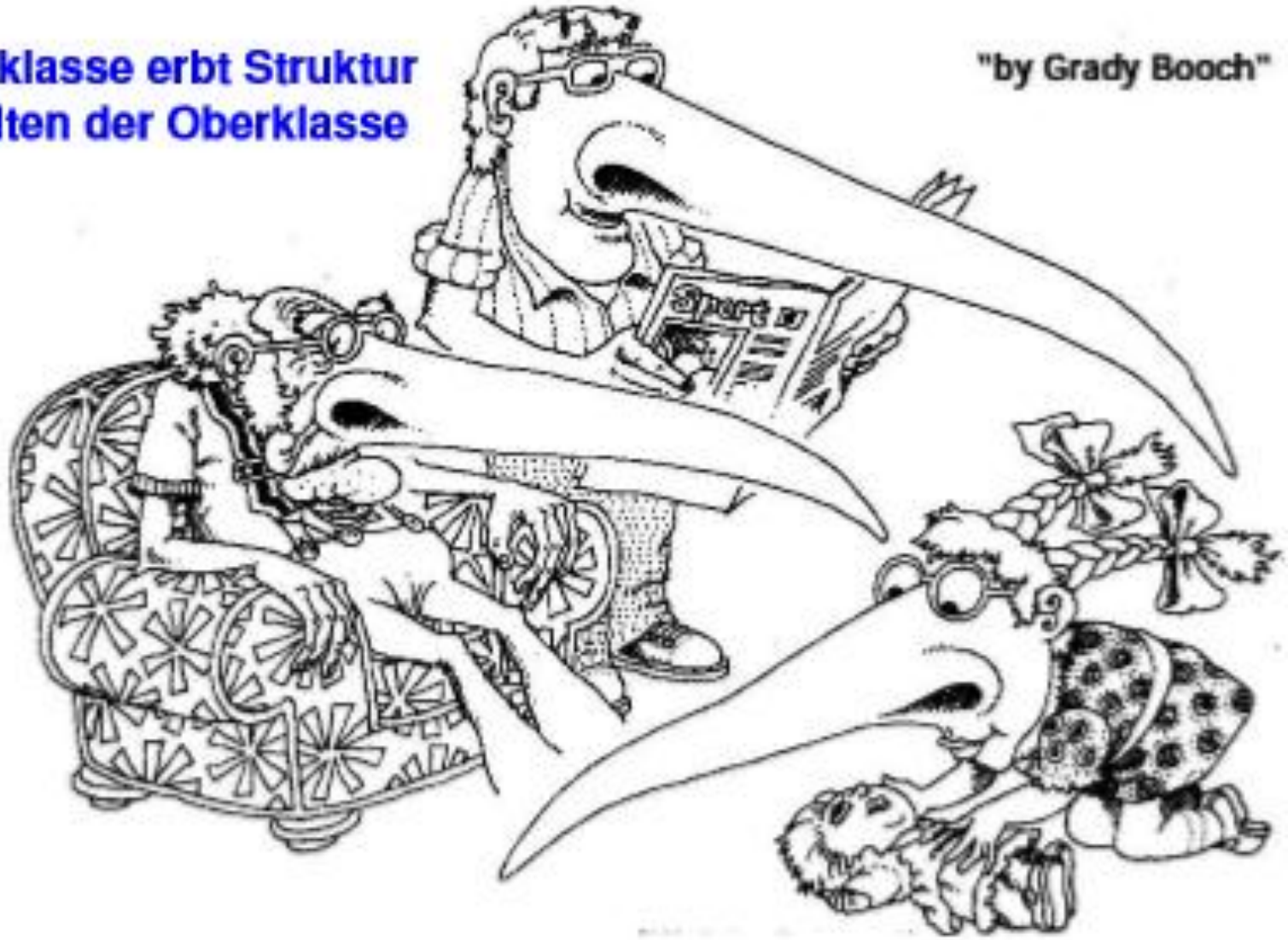
3. Vererbung

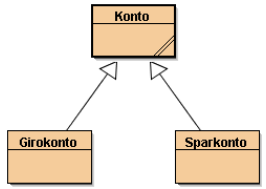


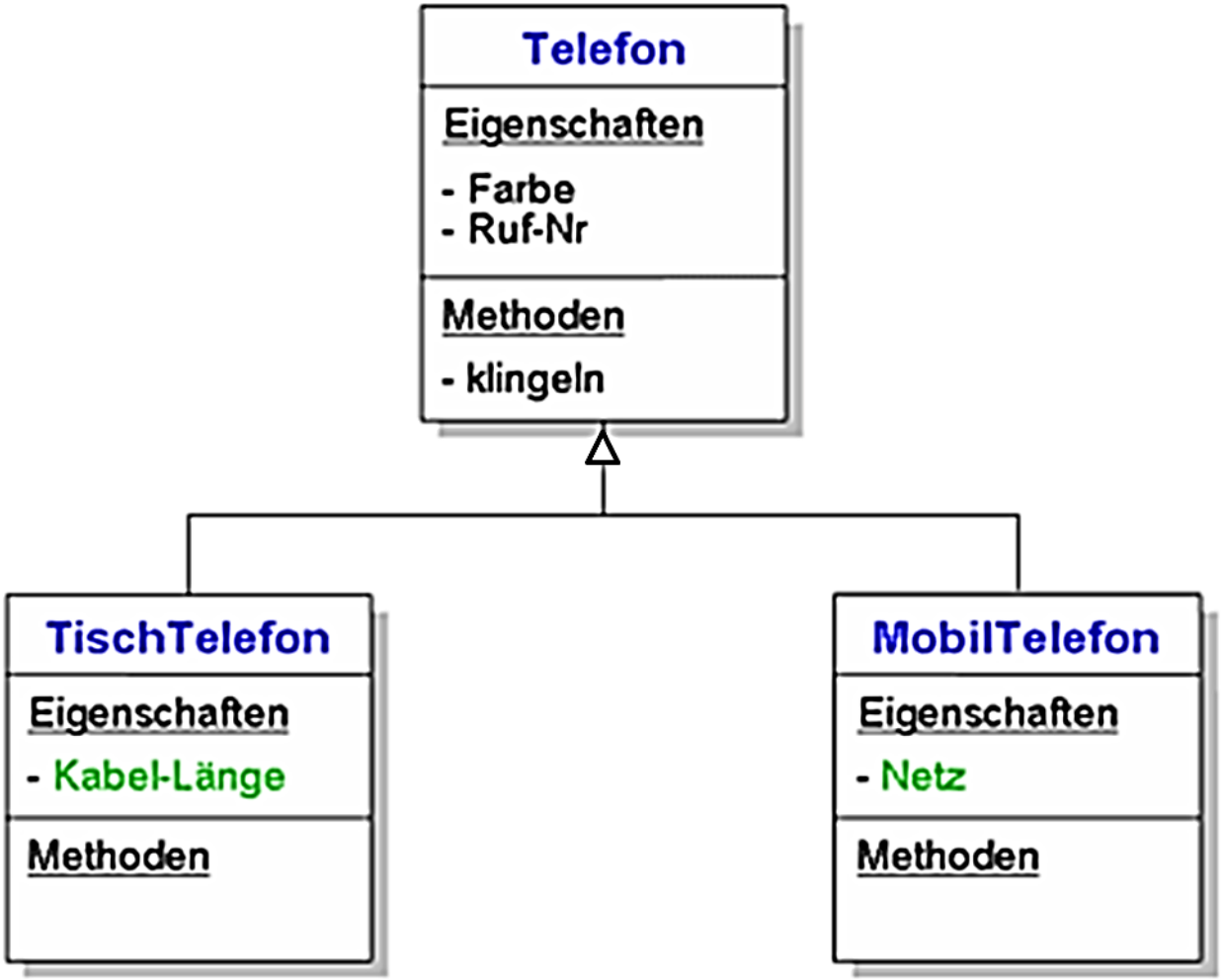
Eine Unterklasse erbt Struktur und Verhalten der Oberklasse

"by Grady Booch"



3. Vererbung



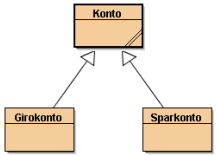


Basis-Klasse

Vererbung

Sub-Klasse

3. Vererbung

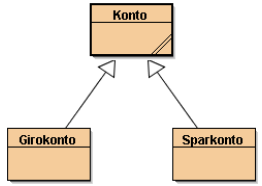


Spezialisierung und Generalisierung

Spezialisierung: Bildung eines Objekttyps, der in einem anderen Objekttyp vollständig enthalten ist. Sie repräsentiert eine ist-ein-Beziehung.
(Bsp: Hase ist ein Säugetier und enthält alle dessen Eigenschaften und Fähigkeiten)

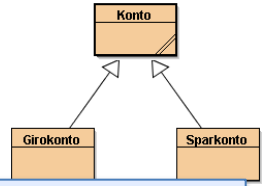
Die gemeinsame Struktur verschiedener Klassen wird durch **Generalisierung** in einer Basisklasse beschrieben. Sie erzeugt ein Objekt, das untergeordnete Objekttypen mit allgemeinen Eigenschaften vollständig enthält.
(Bsp: Person enthält gemeinsame Eigenschaften und Methoden von StudentInnen, DozentInnen, PraktikantInnen, HausmeisterInnen…)

3. Vererbung



ist das Erzeugen einer neuen Klasse als Erweiterung einer bereits bestehenden Klasse.
Die abgeleitete Klasse wird Subklasse genannt.
Die übergeordnete Klasse wird Basis - oder Superklasse genannt.

3. Vererbung



Erbe anfordern 😂 in der Subklasse

public class Kind extends Mutter { ... }

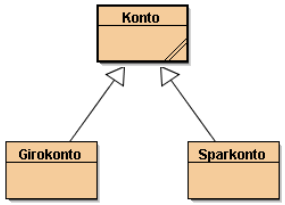
Die Subklasse erbt alle Attribute und Methoden der Basisklasse, die nicht auf *private* *(Information Hiding)* sind.

Benutzen einer Methode aus der Basisklasse

super. mutterMethode (...)

Eine Methode der Basisklasse kann dabei überschrieben (d.h. neu programmiert) werden --> Polymorphie
Sinnvollerweise enthält eine Subklasse s zusätzliche Attribute bzw. Methoden.

3. Vererbung



Vorteile der Vererbung

• **gemeinsame Nutzung** von Programmcode, da er in Unterklassen nicht wiederholt werden muss. (codesharing)

• **übersichtliche** Programmstruktur

• Basisklassen speichern die **allgemeine Programmfunktionalität**, die Subklassen drücken die **speziellere Funktionalität** aus.

• Bei Problemänderungen einfaches **“Unterhängen”** weiterer Subklassen mit neuer speziellerer Funktionalität möglich

3. Vererbung

Codesharing durch Vererbung

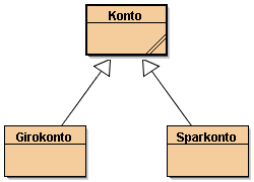
```
public class Konto
{
    String kontonr;
    String inhaber;
    String ort;
    protected int versuche;
```

Klasse, in der die Attribute und Methoden programmiert und vererbt werden.

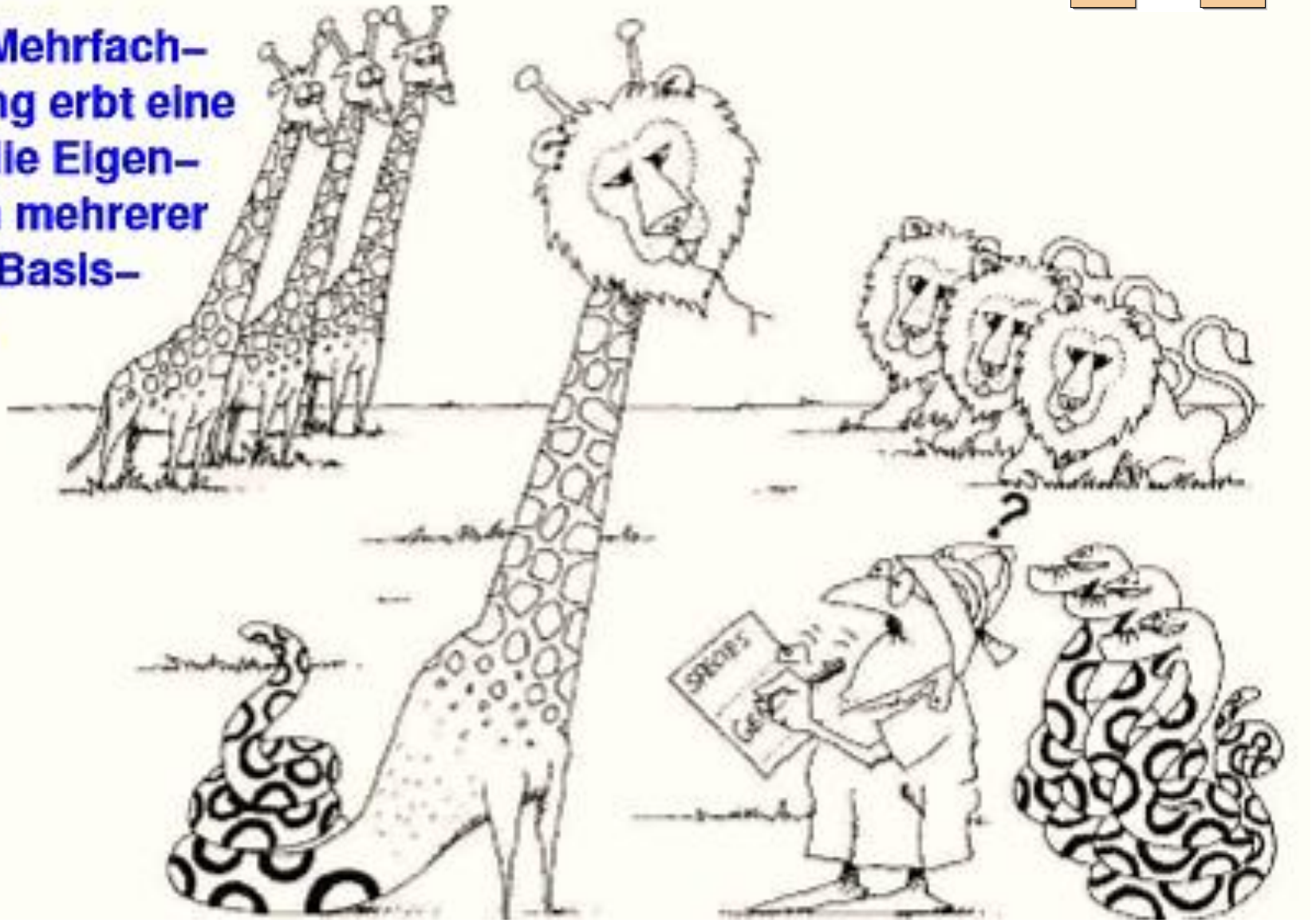
```
public class Geschaeftskonto extends Konto
{
    String firma, geschaeftsfuehrer;
    public Geschaeftskonto()
    {
        firma=InOut.readString("Firmenname: ...");
        geschaeftsfuehrer=InOut.readString("Geschäftsführer: ...");
        super.kontoDatenEingeben();
    }

    public void druckeAuszug()
    {
        System.out.println("Firma:                "+firma);
        System.out.println("Geschäftsführer:      "+geschaeftsfuehrer);
        super.druckeAuszug();
    }
}
```

Klasse, die von der Konto-Klasse erbt.

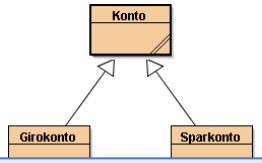
3. Vererbung



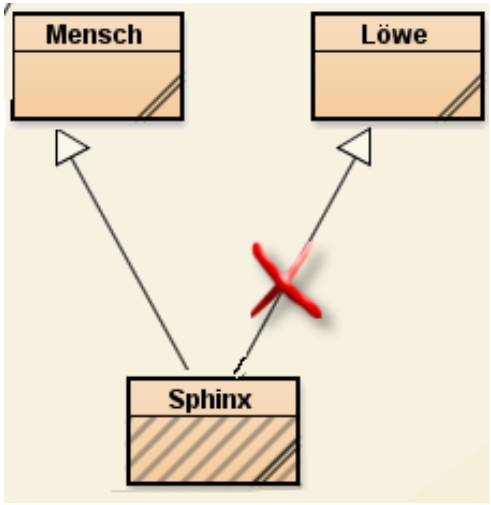
Bei der Mehrfach–vererbung erbt eine Klasse die Eigen–schaften mehrerer direkter Basis–klassen

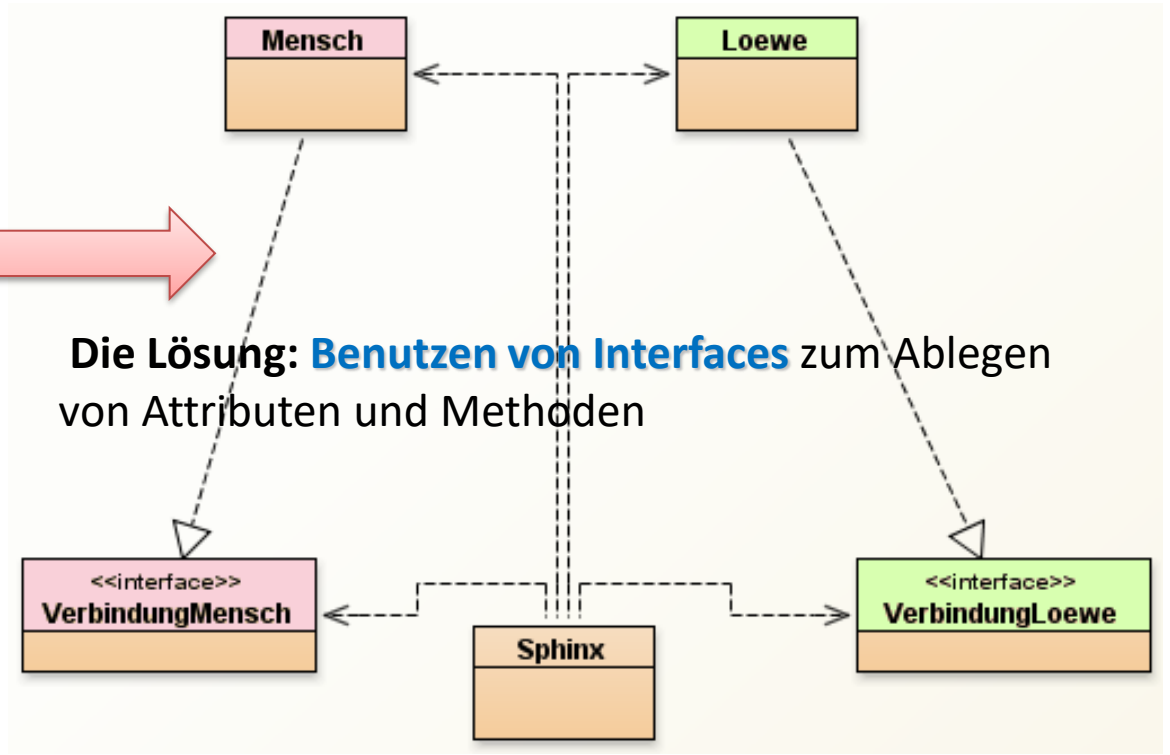


3. Vererbung



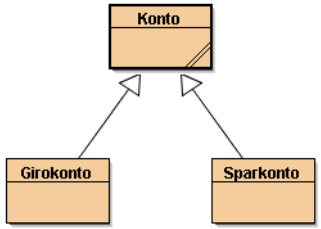
Wichtig!
In Java gibt es (im Gegensatz zu anderen Programmiersprachen)
keine Mehrfachvererbung.





Die Lösung: Benutzen von Interfaces zum Ablegen von Attributen und Methoden

3.1. Polymorphie

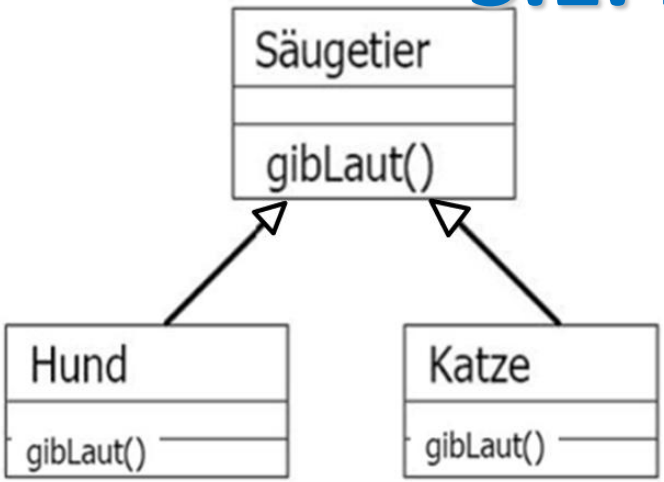


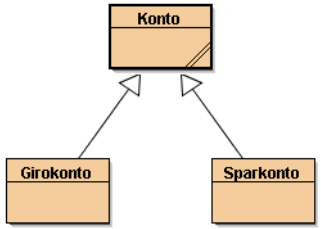
Bei der Vererbung ist es meist so, dass die Basisklassen nicht allen speziellen Anforderungen der Subklassen gerecht werden.

Methoden, die nicht so verwendet werden können, wie sie in der Basisklasse festgelegt sind, werden in den Subklassen neu definiert, d.h. überschrieben.

Auf die originale Basis-Methode zuzugreifen, ist trotzdem möglich. Dies geschieht mit dem Schlüsselwort super.methode().

3.1. Polymorphie



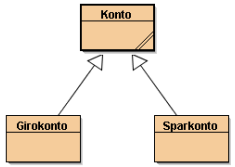


Beide Subklassen erben die Methode gibLaut().

Diese muss aber für Hunde - und Katzenobjekte angepasst werden - die geerbte Methodewird überschrieben und durch eine individuelle Methode ersetzt.

Mit super.gibLaut() könnte die Original-Methode der Basisklasse trotzdem genutzt werden.

3.1. Polymorphie



Der Zugriff der Subklasse auf die Basisklasse erfolgt mit dem Schlüsselwort **super (**…evtl. Parameter…**)**.
Werden Methoden/ Attribute/ Konstruktoren überschrieben, so nennt man dies Polymorphie.
– mit super() erfolgt der Aufruf des Konstruktors der Oberklasse
– super() muss der erste Befehl im Subklassen-Konstruktor sein

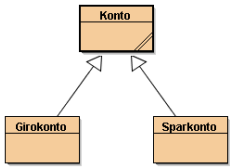
- **super.methodenName** (…evtl. Parameter…**)**

– ruft die Methode der Basisklasse auf
– Aufruf im Konstruktor oder den Methoden der Subklasse

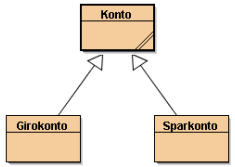
- **super.attributName = 123;**

– greift auf ein Attribut der Basisklasse zu

3.1. Polymorphie

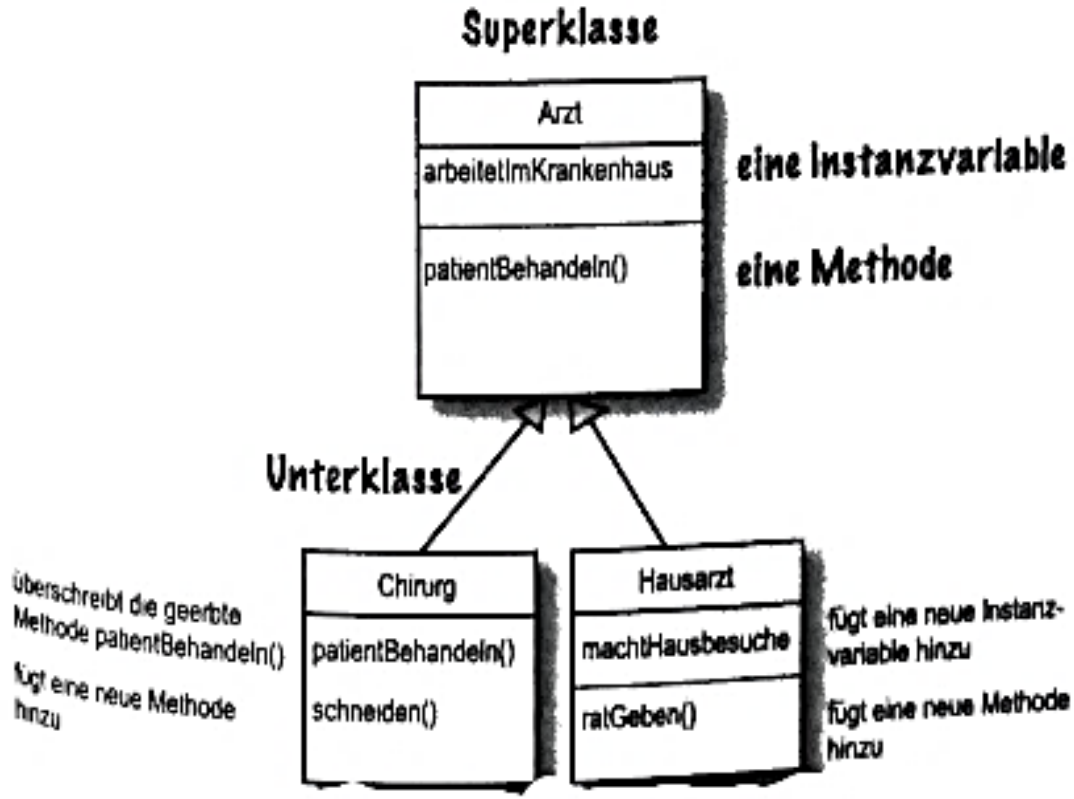


- **super()** muss immer der erste Befehl sein, der im Subklassen-Konstruktor aufgerufen wird.
- Hat der Konstruktor der Basisklasse keine Parameter, wird er beim Erzeugen eines Objektes der Subklasse automatisch aufgerufen. Ein Aufruf mittels **super()** ist überflüssig.
- Gibt es in der Basisklasse einen Konstruktor mit Parametern, dann ist der Aufruf super(...parameter...) im Konstruktor der Subklasse zwingend erforderlich (Fehlermeldung).
- Gibt es in der Basisklasse einen Konstruktor mit Parametern und einen weiteren parameterlosen, hat man in der Subklasse die Wahl.



3. Polymorphie

	Klasse	Package	Unter-klasse	Überall
public	✓	✓	✓	✓
protected*	✓	✓ (komisch)	✓	✗
Keine Angabe: „package private“	✓	✓	✗	✗
private*	✓	✗	✗	✗



Wie viele Instanzvariablen hat Chirurg? _____

Wie viele Instanzvariablen hat Hausarzt? _____

Wie viele Methoden hat Arzt? _____

Wie viele Methoden hat Chirurg? _____

Wie viele Methoden hat Hausarzt? _____

Kann ein Hausarzt patientBehandeln()? _____

Kann ein Hausarzt schneiden()? _____

```
public class Arzt {

  boolean arbeitetImKrankenhaus;

  void patientBehandeln() {
    // Patient untersuchen
  }
}
```

```
public class Hausarzt extends Arzt {

  boolean machtHausbesuche;
  void ratGeben() {
    // Ratschlag geben
  }
}
```

```
public class Chirurg extends Arzt {

  void patientBehandeln() {
    // Operation durchführen
  }

  void schneiden(){
    // Schnitt durchführen (Autsch!)
  }
```