TECHNISCHE UNIVERSITÄT DRESDEN

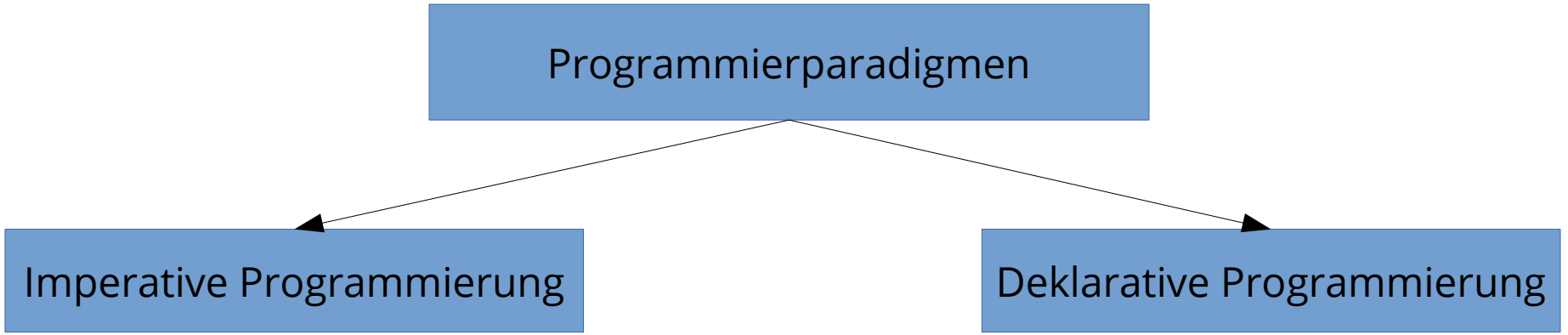
DRESDEN concept

Professur für Didaktik der Informatik
Dr. Thiemo Leonhardt

Programmierparadigmen
Objektorientierte Modellierung

Professur für **Didaktik der Informatik**

Klassifikation von Programmiersprachen



Objektorientierung – OOM und OOP

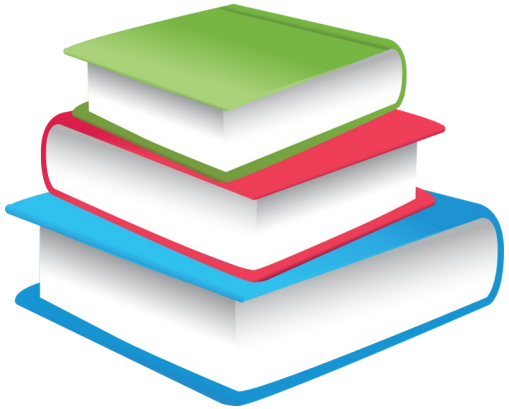
Zentrales Konzept der objektorientierten Sicht

In der objektorientierten Sicht ist das zentrale Konzept die **Klasse** und das **Objekt**.

Die objektorientierte Sicht

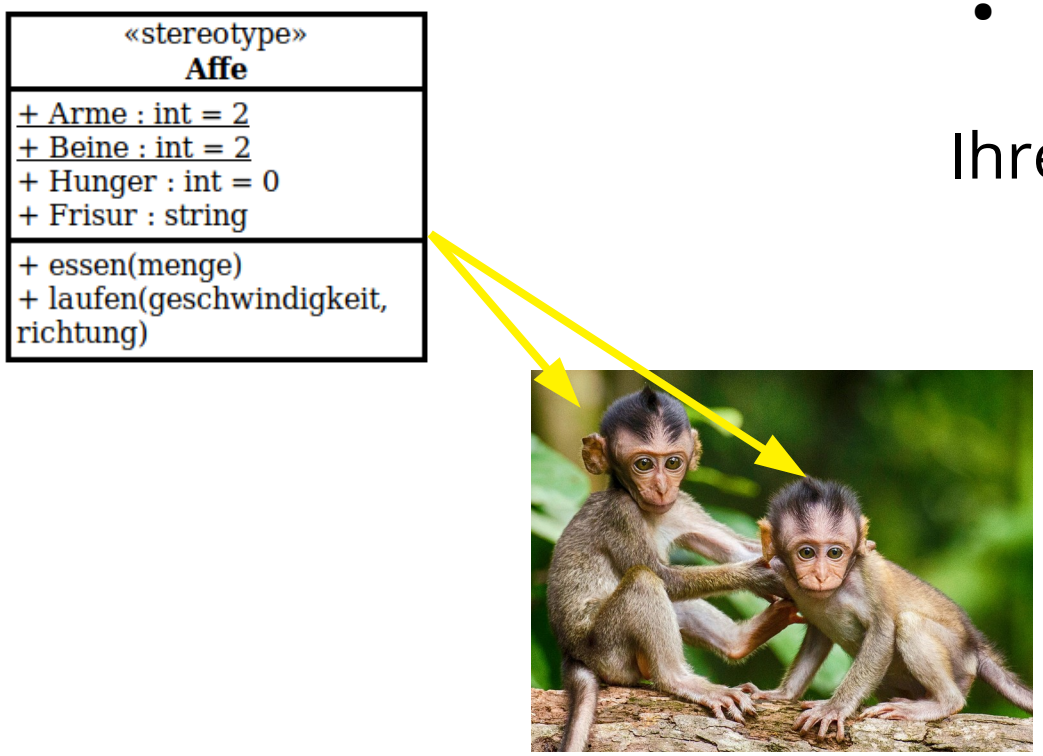
- Objekt
  - In der realen Welt sind dies greifbare Gegenstände
    - Buch
    - Auto
  - oder auch abstrakte Konstrukte
    - Absatz in einem Buch
    - Ton in deinem Lied

- Charakterisiert werden Objekte durch ihre Eigenschaften
- Welche Eigenschaften charakterisieren die Beispielobjekte?



Objektorientierte Modellierung - OOM

- Objekte werden je nach ihren charakteristischen Eigenschaften in **Klassen** eingeteilt



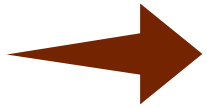
Eine Klasse beschreibt, welche

- **Eigenschaften** und
- **Fähigkeiten**

Ihre Objekte aufweisen.

Fachsprache in der Objektorientierten Sicht

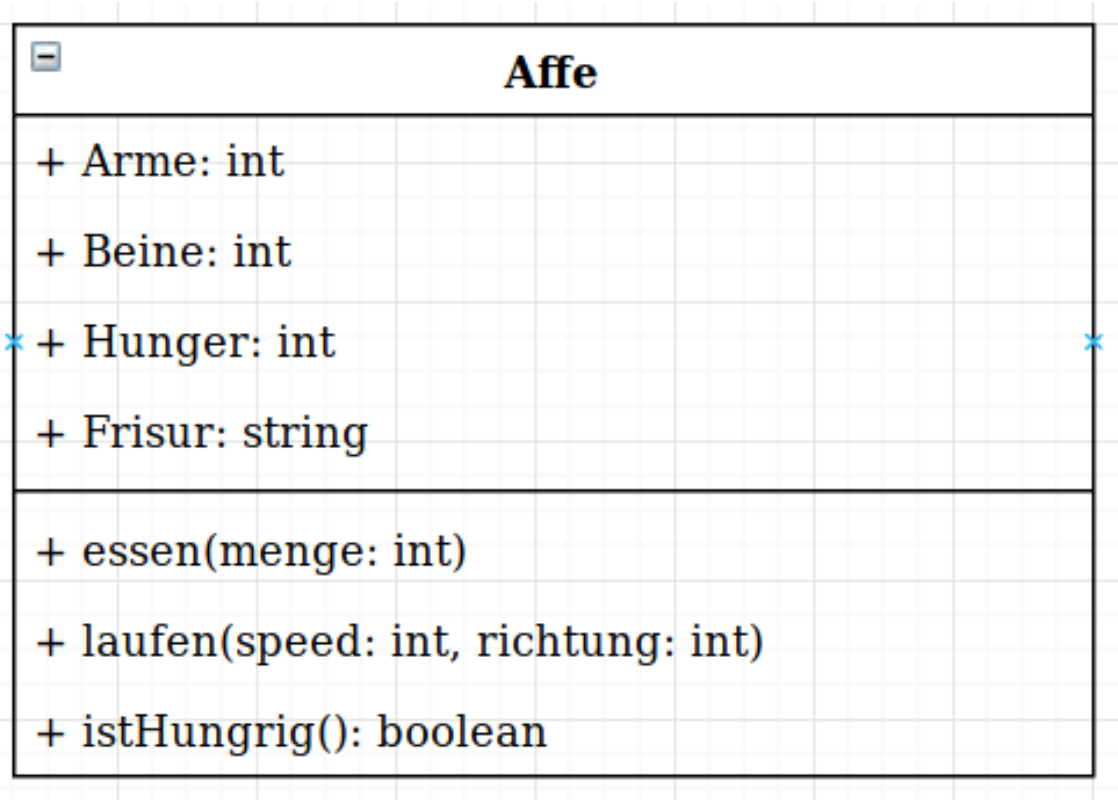
• Klasse	• Klasse
• Eigenschaften	• Attribute
• Name der Eigenschaft	• Bezeichner
• Art der Eigenschaft	• Datentyp
• Fähigkeiten	• Methoden
• Echte Ausprägung einer Klasse	• Objekt



Grafische Beschreibung einer Klasse

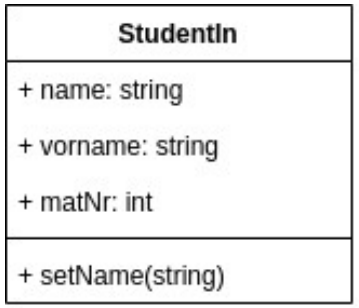
Darstellung in einer Klassenkarte

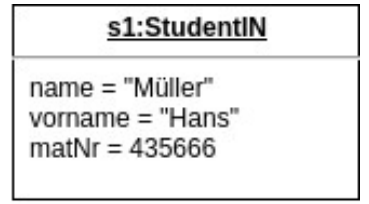
- Klassenname
- Attribute
  - Datentypen
- Methoden
  - Rückgabewerte werden mit : angegeben
  - Beispiel istHungrig(): boolean



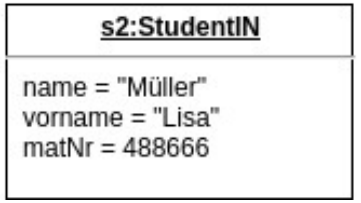
Grafische Beschreibung eines Objektes

- Objektdiagramm
  - UML Spezifikation
  - Objekt als **Instanz** einer Klasse
  - Werte der Attribute eines Objektes



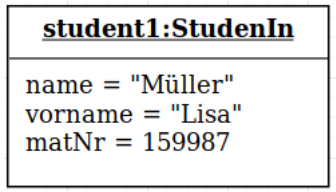


- Zustands eines Programms durch die Darstellung der Objekte

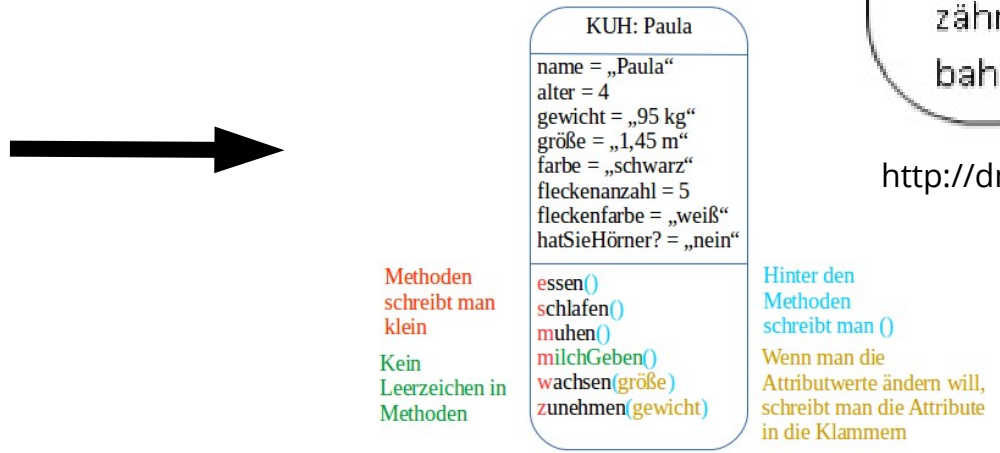


Grafische Beschreibung eines Objektes

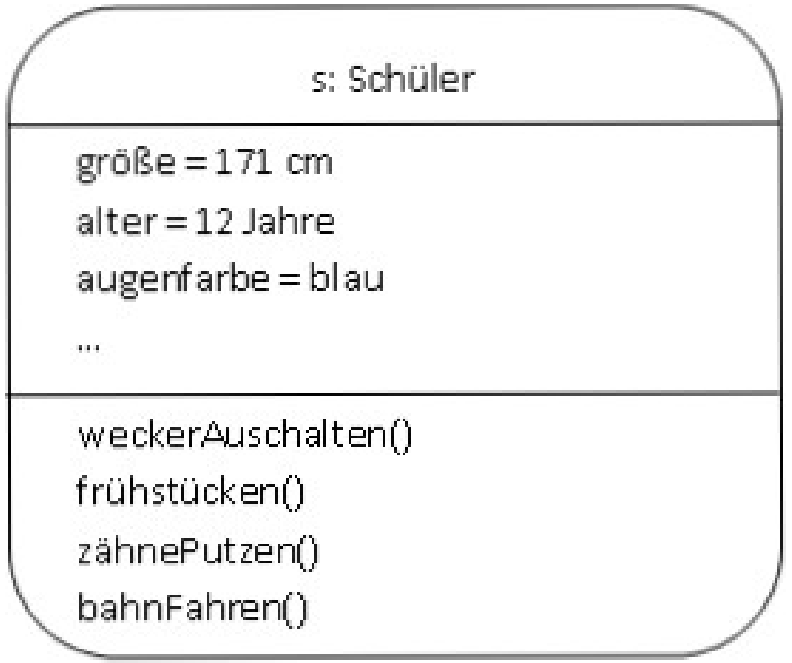
- Objektkarte
  - Informatik in der Schule
  - KOAM-Modell
  - Verschiedene Interpretationen der *richtigen Darstellung*



Korrektes UML Objektdiagramm



https://assets.serlo.org/



http://dmuw.zum.de/

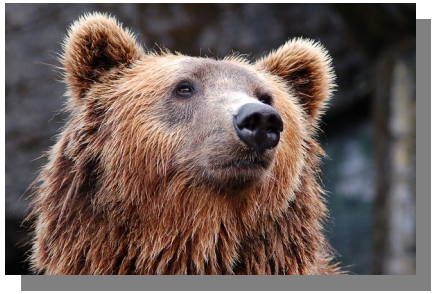
Schuldidaktische Veränderung

Was Sie mitnehmen und vertiefen sollten…

- Welches zentrale Konzept liegt der Objektorientierten Sicht zugrunde?
- Was ist der Unterschied zwischen Klassen und Objekten?
- Welche Fachbegriffe definieren Klassen und Objekte?
- Wie kann ich Klassen und Objekte grafisch darstellen?

Objektorientierte Modellierung

Beispiel: Exponat Wilde Tiere



Was soll es tun:
Sie berühren ein Bild. Das Tier brüllt oder es wird Ihnen etwas über das Tier erzählt!

Ein imperativer Ansatz

```
def textWiedergeben(tierart):
  if tierart == 'Affe':
    # Text Affe laden und abspielen
  elif tierart == 'Löwe':
    # Text Löwe laden und abspielen
  elif tierart == 'Bär':
    # Text Bär laden und abspielen
```

```
def bruellen(tierart):
  if tierart == 'Affe':
    # Affensound laden und abspielen
  elif tierart == 'Löwe':
    # Löwensound laden und abspielen
  elif tierart == 'Bär':
    # Bärensound laden und abspielen
```

Ein imperativer Ansatz

```
def textWiedergeben(tierart):
  if tierart == 'Affe':
    # Text Affe laden und abspielen
  elif tierart == 'Löwe':
    # Text Löwe laden und abspielen
  elif tierart == 'Bär':
    # Text Bär laden und abspielen
```

```
def bruellen(tierart):
  if tierart == 'Affe':
    # Affensound laden und abspielen
  elif tierart == 'Löwe':
    # Löwensound laden und abspielen
  elif tierart == 'Bär':
    # Bärensound laden und abspielen
```

- Es sind mindestens 2 Prozeduren nötig
  - eine zum Brüllen
  - eine zur Ausgabe der Information über das Tier
- In den Prozeduren:
  - IF-Anweisungen, um auf jedes Tier zu reagieren
    - Sound-Datei auswählen
    - Sound ausgeben
    - Text laden
    - Text wiedergeben

Ein imperativer Ansatz

```
def textWiedergeben(tierart):
  if tierart == 'Affe':
    # Text Affe laden und abspielen
  elif tierart == 'Löwe':
    # Text Löwe laden und abspielen
  elif tierart == 'Bär':
    # Text Bär laden und abspielen
```

```
def bruellen(tierart):
  if tierart == 'Affe':
    # Affensound laden und abspielen
  elif tierart == 'Löwe':
    # Löwensound laden und abspielen
  elif tierart == 'Bär':
    # Bärensound laden und abspielen
```

- Eine Endlos-Schleife überwacht das Drücken auf der GUI.
- Was passiert, wenn ein neues Tier hinzugefügt werden soll?
  - Zur Laufzeit?

Ein imperativer Ansatz

```
def textWiedergeben(tierart):
  if tierart == 'Affe':
    # Text Affe laden und abspielen
  elif tierart == 'Löwe':
    # Text Löwe laden und abspielen
  elif tierart == 'Bär':
    # Text Bär laden und abspielen
```

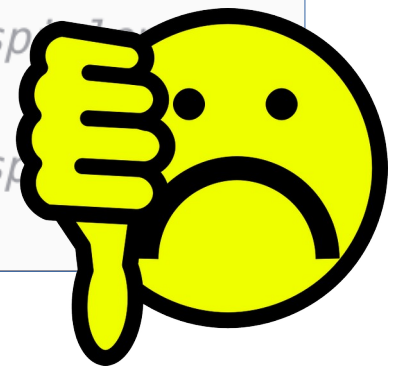
```
def bruellen(tierart):
  if tierart == 'Affe':
    # Affensound laden und abspielen
  elif tierart == 'Löwe':
    # Löwensound laden und abspielen
  elif tierart == 'Bär':
    # Bärensound laden und abspielen
```

- Eine Endlos-Schleife überwacht das Drücken auf der GUI.
- Was passiert, wenn ein neues Tier hinzugefügt werden soll?
  - Zur Laufzeit?
- Was passiert, wenn die Tiere eine neue Fähigkeit wie die Anzeige eines Videos erhalten sollen?

Ein imperativer Ansatz

```
def textWiedergeben(tierart):
  if tierart == 'Affe':
    # Text Affe laden und abspielen
  elif tierart == 'Löwe':
    # Text Löwe laden und abspielen
  elif tierart == 'Bär':
    # Text Bär laden und abspielen
```

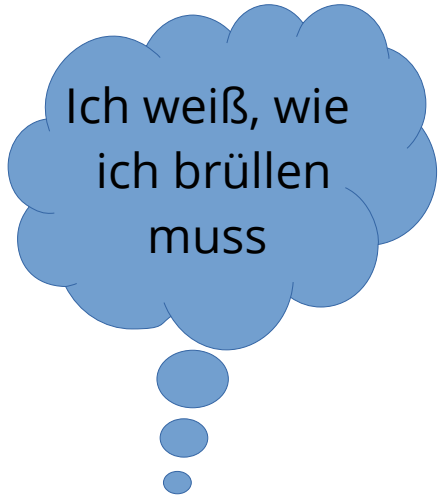
```
def bruellen(tierart):
  if tierart == 'Affe':
    # Affensound laden und abspielen
  elif tierart == 'Löwe':
    # Löwensound laden und abspielen
  elif tierart == 'Bär':
    # Bärensound laden und abspielen
```



- Eine Endlos-Schleife überwacht das Drücken auf der GUI.
- Was passiert, wenn ein neues Tier hinzugefügt werden soll?
  - Zur Laufzeit?
- Was passiert, wenn die Tiere eine neue Fähigkeit wie die Anzeige eines Videos erhalten sollen?

Objektorientierter Ansatz

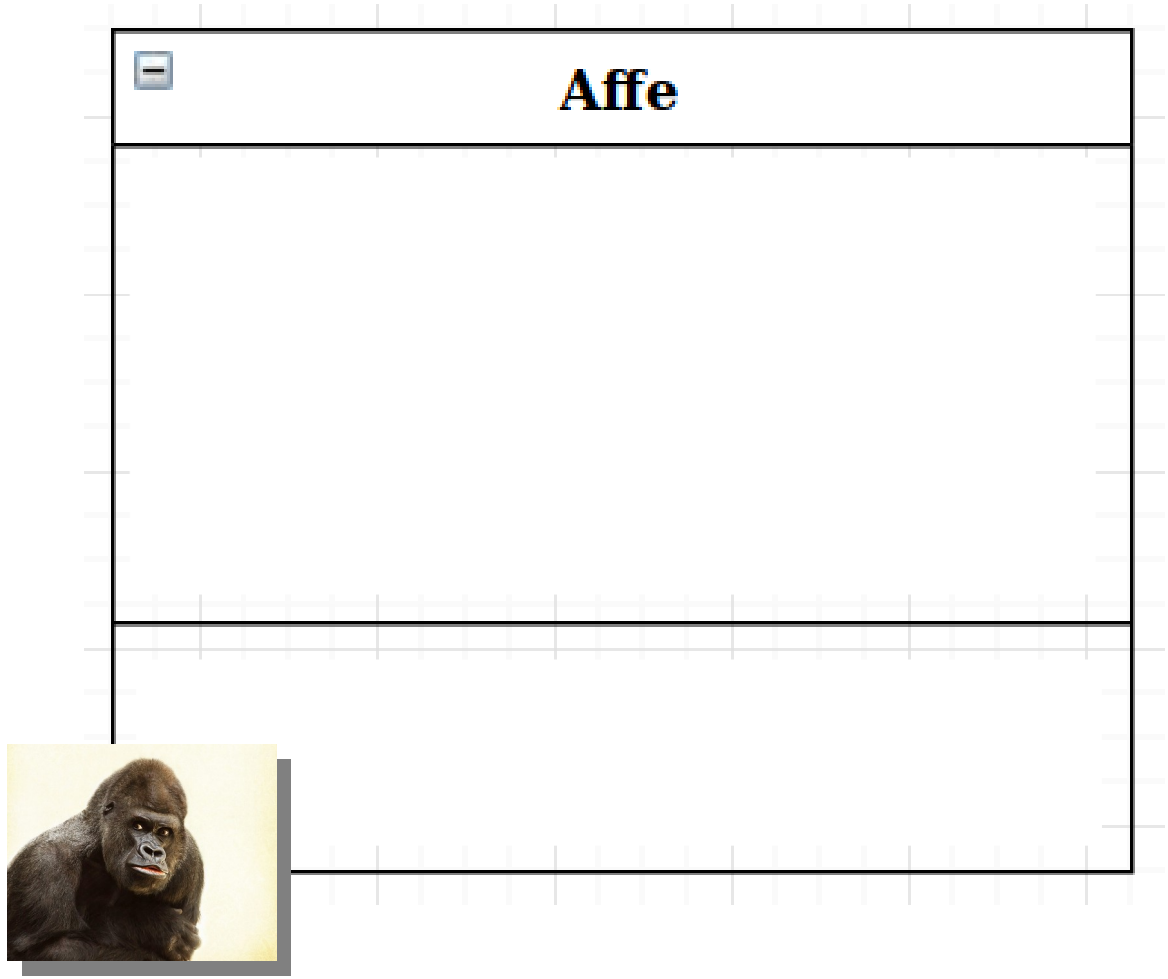
Sagen Sie dem Affen direkt, was er machen soll!



Affe, brüll! $!@#%$!!!#

Affe, erzähl etwas! Ich mag Bananen

Objektorientierte Klasse ableiten



Laden sich sich aus dem

- OPAL-→ Linkliste → Desktop UML

das Programm herunter (no_installer) und starten Sie es.

Modellieren Sie die Klasse Affe

- mit einer Soundatei affe.mp3 als Pfad
- mit einer Bilddatei affe.png als Pfad
- mit einer Höhe und Breite des Bildes
- mit der Möglichkeit zu brüllen
- mit der Möglichkeit etwas zu erzählen

Objektorientierte Klasse ableiten

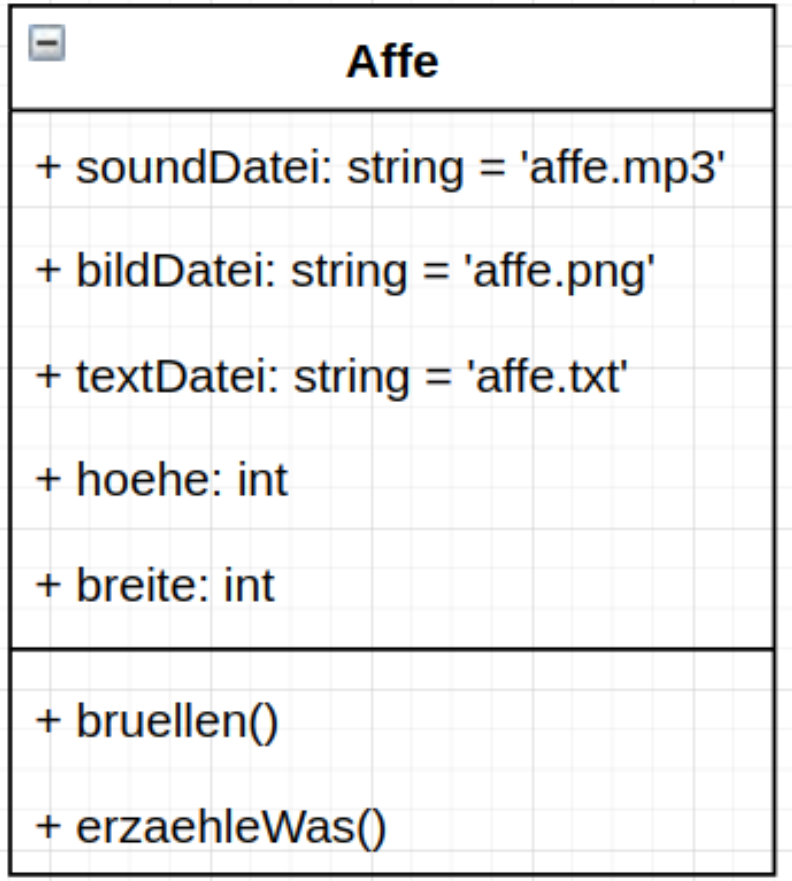
Laden sich sich aus dem

- OPAL-→ Linkliste → Desktop UML

das Programm herunter (no_installer) und starten Sie es.

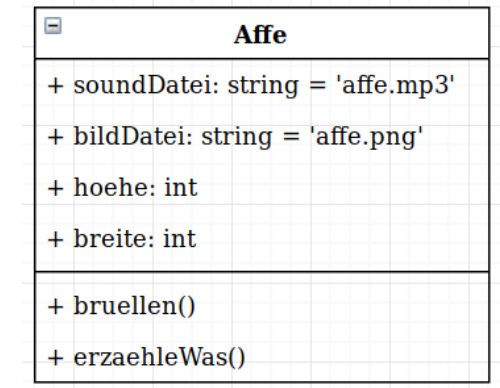
Modellieren Sie die Klasse Affe

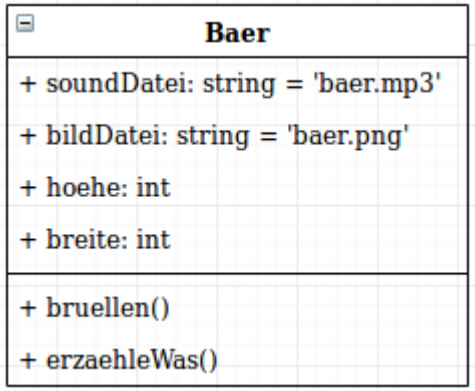
- mit einer Soundatei affe.mp3 als Pfad
- mit einer Bilddatei affe.png als Pfad
- mit einer Höhe und Breite des Bildes
- mit der Möglichkeit zu brüllen
- mit der Möglichkeit etwas zu erzählen

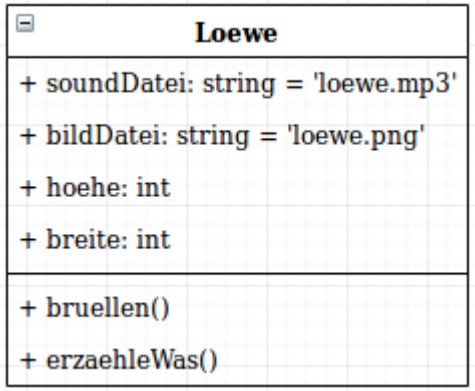


Noch mehr Klassen...

- Für jedes Tier wird eine ähnliche Klasse geschrieben
  - Sollte sich z.B. das Format für die Sounddatei für den Bären ändern, müssen Sie nur in der Bären-Klasse die Änderungen vornehmen.
  - Den Code für die anderen Tiere müssen Sie nicht ändern!
  - Oder wenn Sie später noch einen Tiger hinzufügen wollen, dann schreiben Sie ihm eben eine eigene Tiger-Klasse.



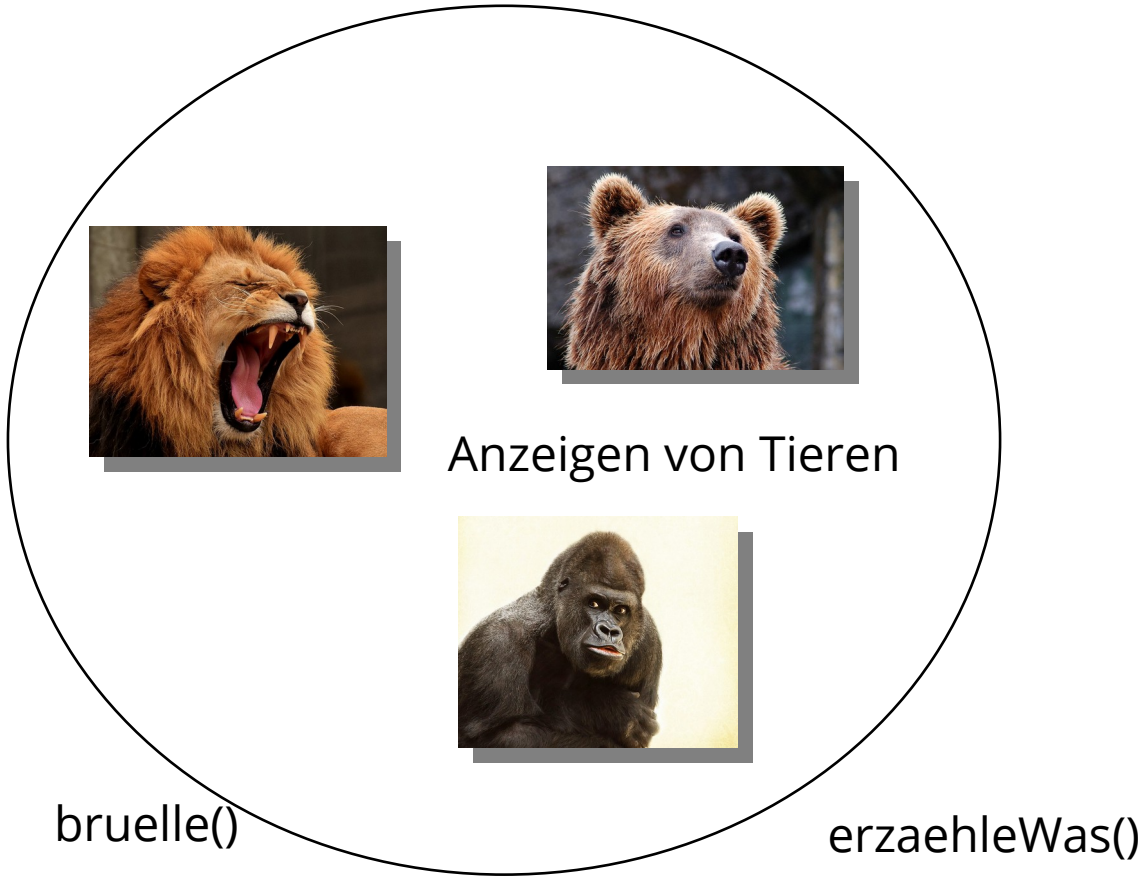




Was haben Affen, Löwen und Bären gemeinsam

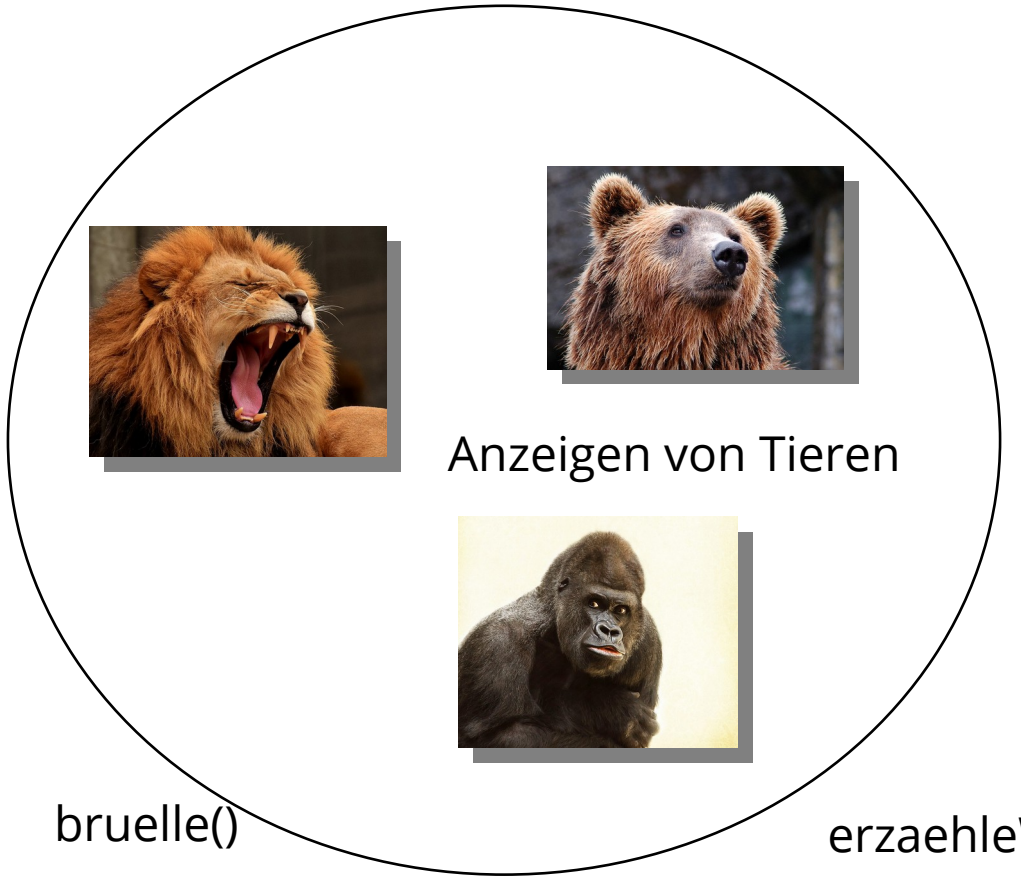
- OOM erfordert,
  - dass Sie sich über die Struktur ihres Programms Gedanken machen.
  - Sie werden das Programm erweitern wollen, um zusätzliche Tiere hinzuzufügen.

Was haben Affen, Löwen und Bären gemeinsam



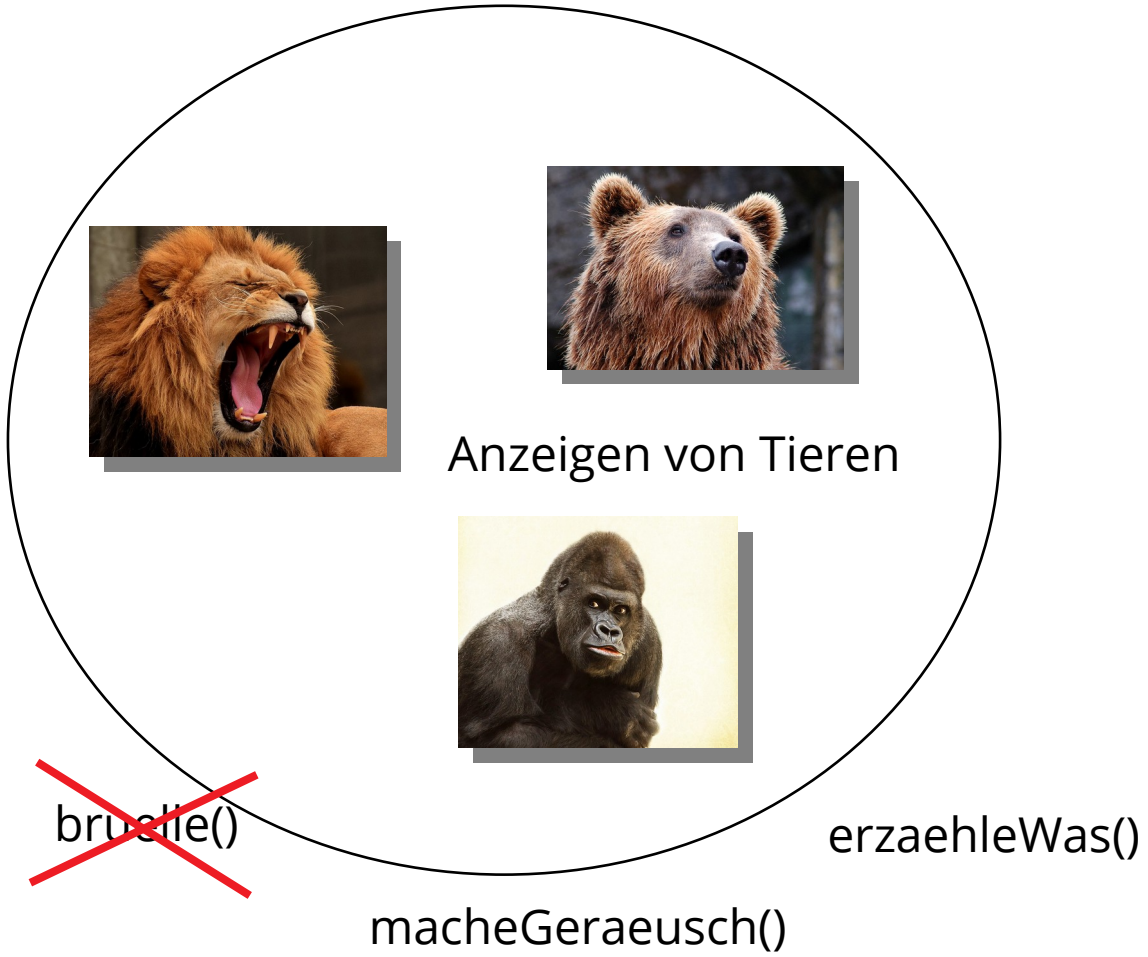
- OOM erfordert,
  - dass Sie sich über die Struktur ihres Programms Gedanken machen.
  - Sie werden das Programm erweitern wollen, um zusätzliche Tiere hinzuzufügen.

Was haben Affen, Löwen und Bären gemeinsam



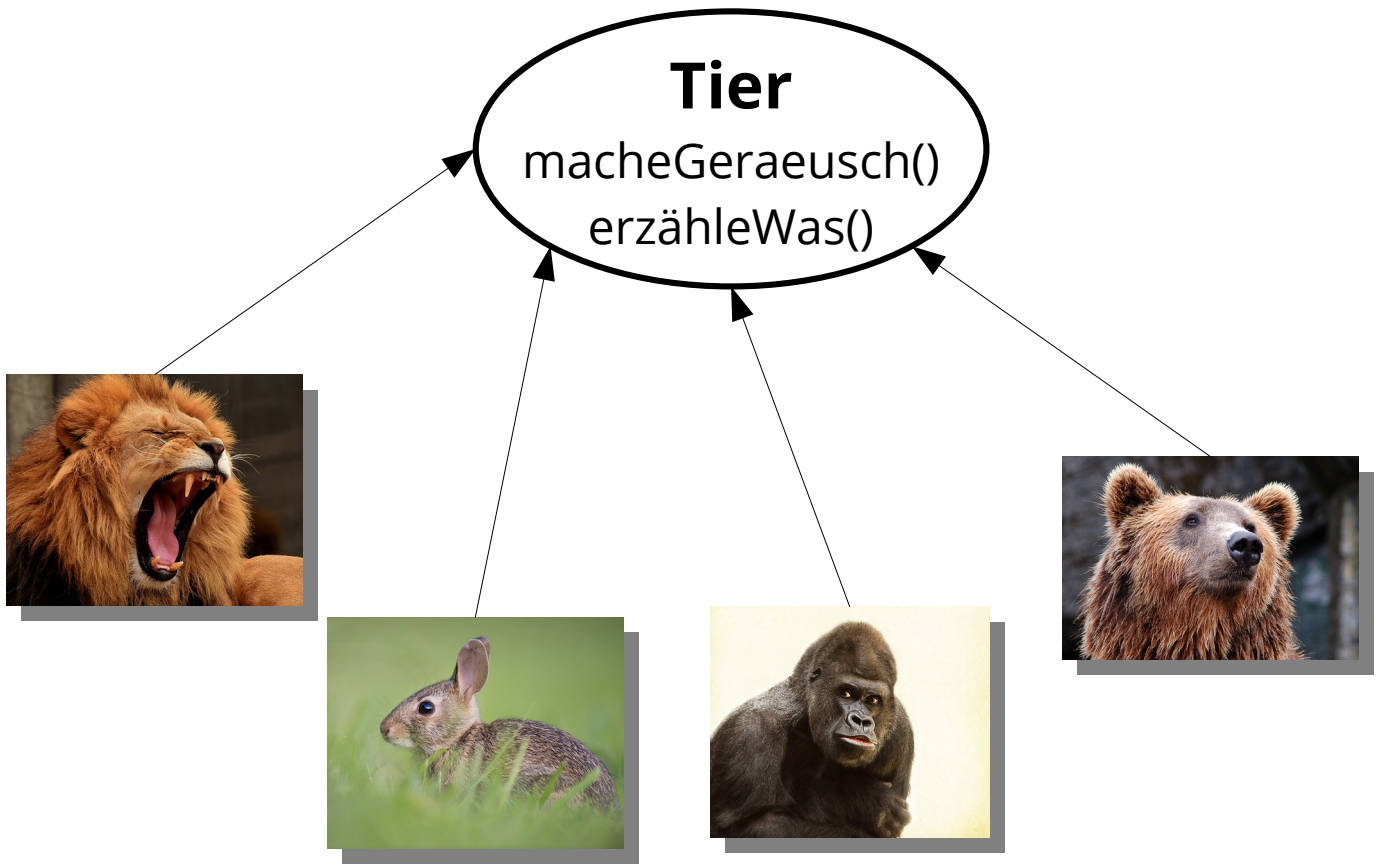
- OOM erfordert,
  - dass Sie sich über die Struktur ihres Programms Gedanken machen.
  - Sie werden das Programm erweitern wollen, um zusätzliche Tiere hinzuzufügen.

Was haben Affen, Löwen und Bären gemeinsam



- OOM erfordert,
  - dass Sie sich über die Struktur ihres Programms Gedanken machen.
  - Sie werden das Programm erweitern wollen, um zusätzliche Tiere hinzuzufügen.

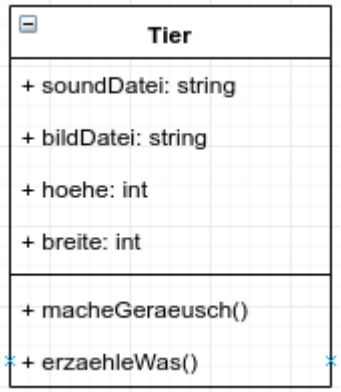
Allgemeine Eigenschaften aller Tierabbildungen im Programm



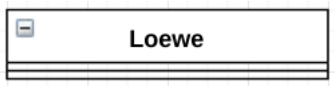
- Denken Sie darüber nach, was die einzelnen Klassen, die Sie schreiben, **gemeinsam** haben.
- Es sind **Abbildungen von Tieren**.
  - Sie können Geräusche machen und etwas erzählen.
  - Sie haben eine Breite und Höhe sowie einen Pfad zu einer Bilddatei und einer Sounddatei.
- Diese Gemeinsamkeiten können wir abstrahieren

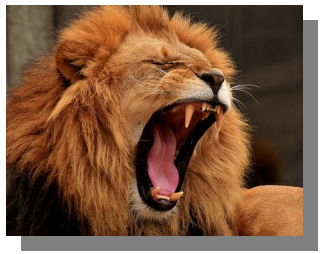
Generalisierung durch Vererbung

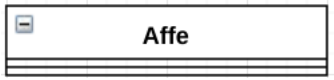
Generalisierung durch Vererbung

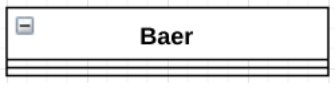


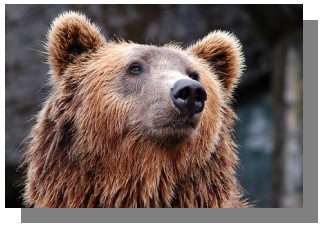
Superklassen vererben alle Attribute und Methoden an Unterklassen

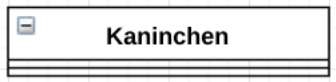


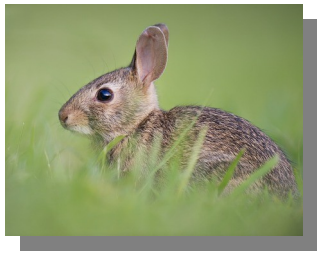




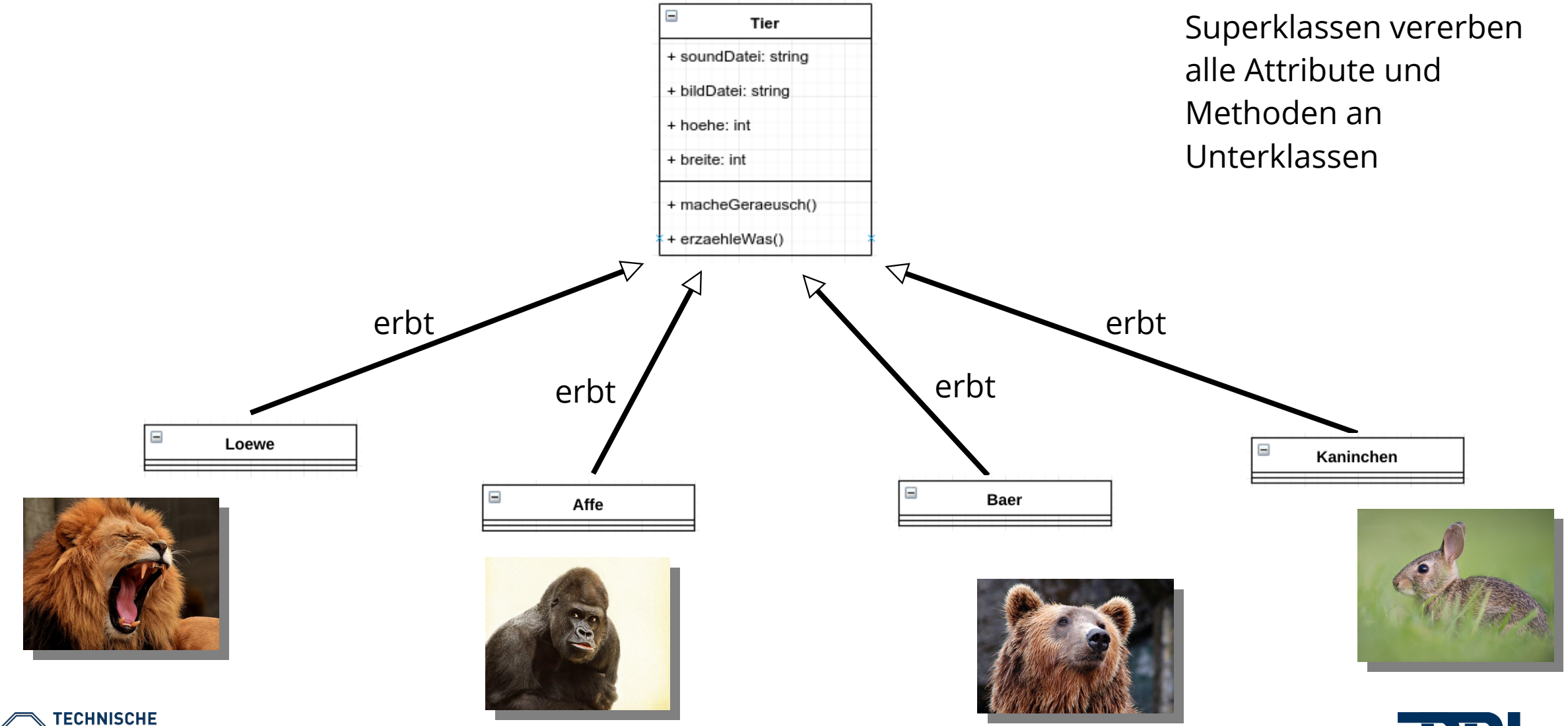






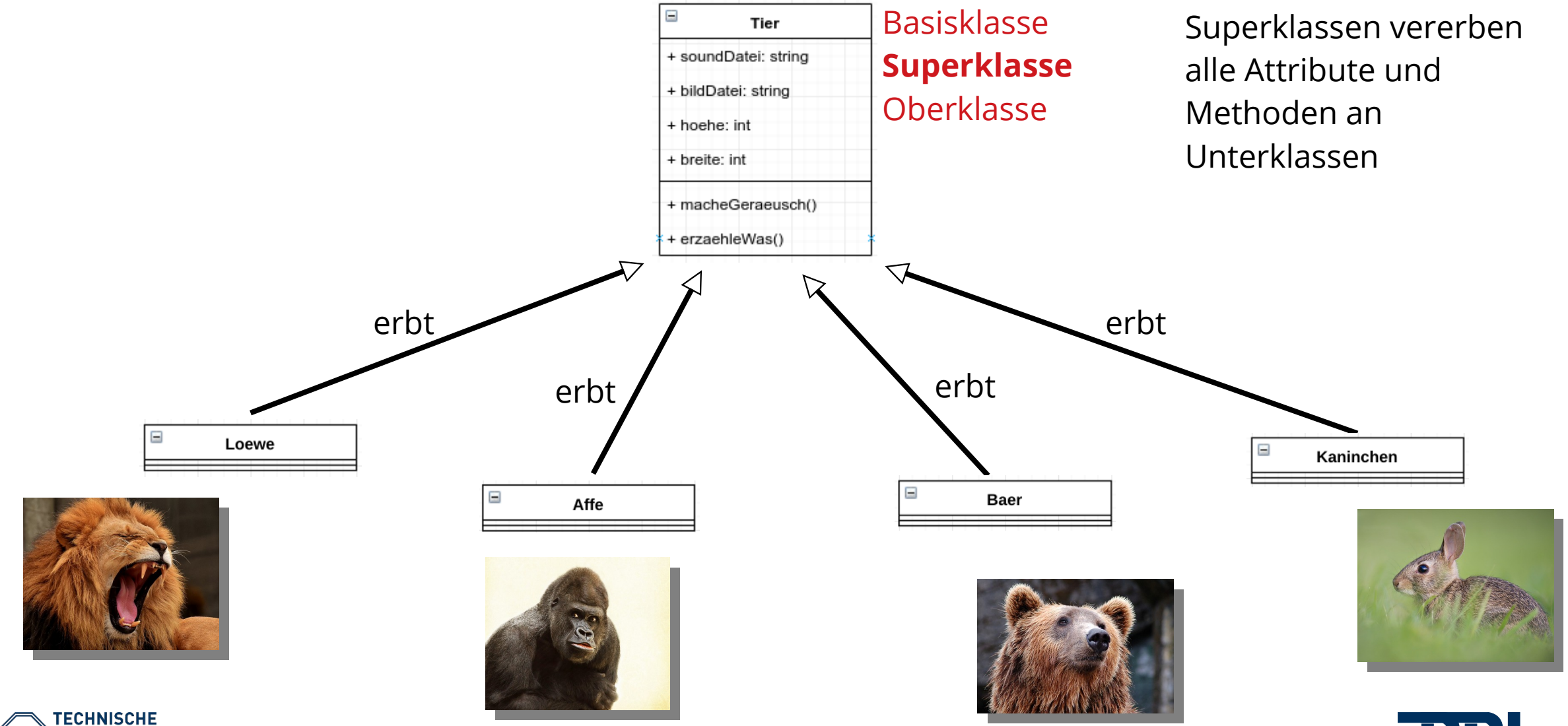


Generalisierung durch Vererbung



Superklassen vererben alle Attribute und Methoden an Unterklassen

Generalisierung durch Vererbung

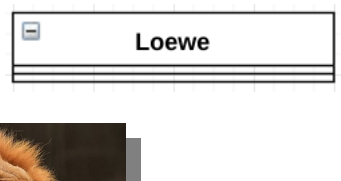


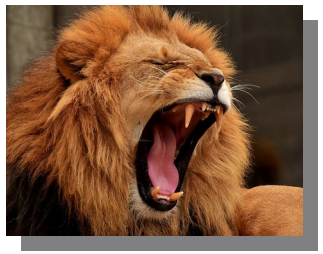
Superklassen vererben alle Attribute und Methoden an Unterklassen

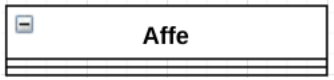
Generalisierung durch Vererbung

Basisklasse
Superklasse
Oberklasse

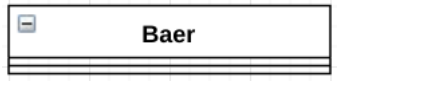
Superklassen vererben alle Attribute und Methoden an Unterklassen

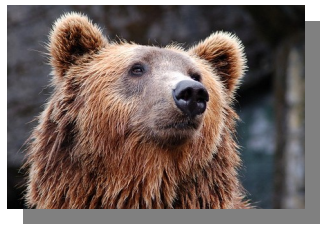


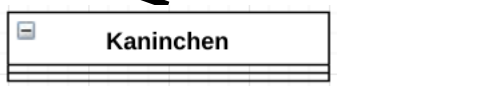


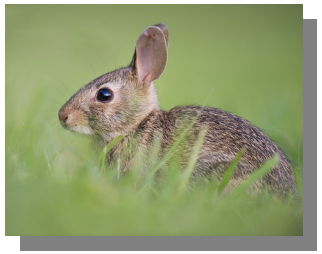


Abgeleitete Klasse
Unterklasse
Subklasse









erbt

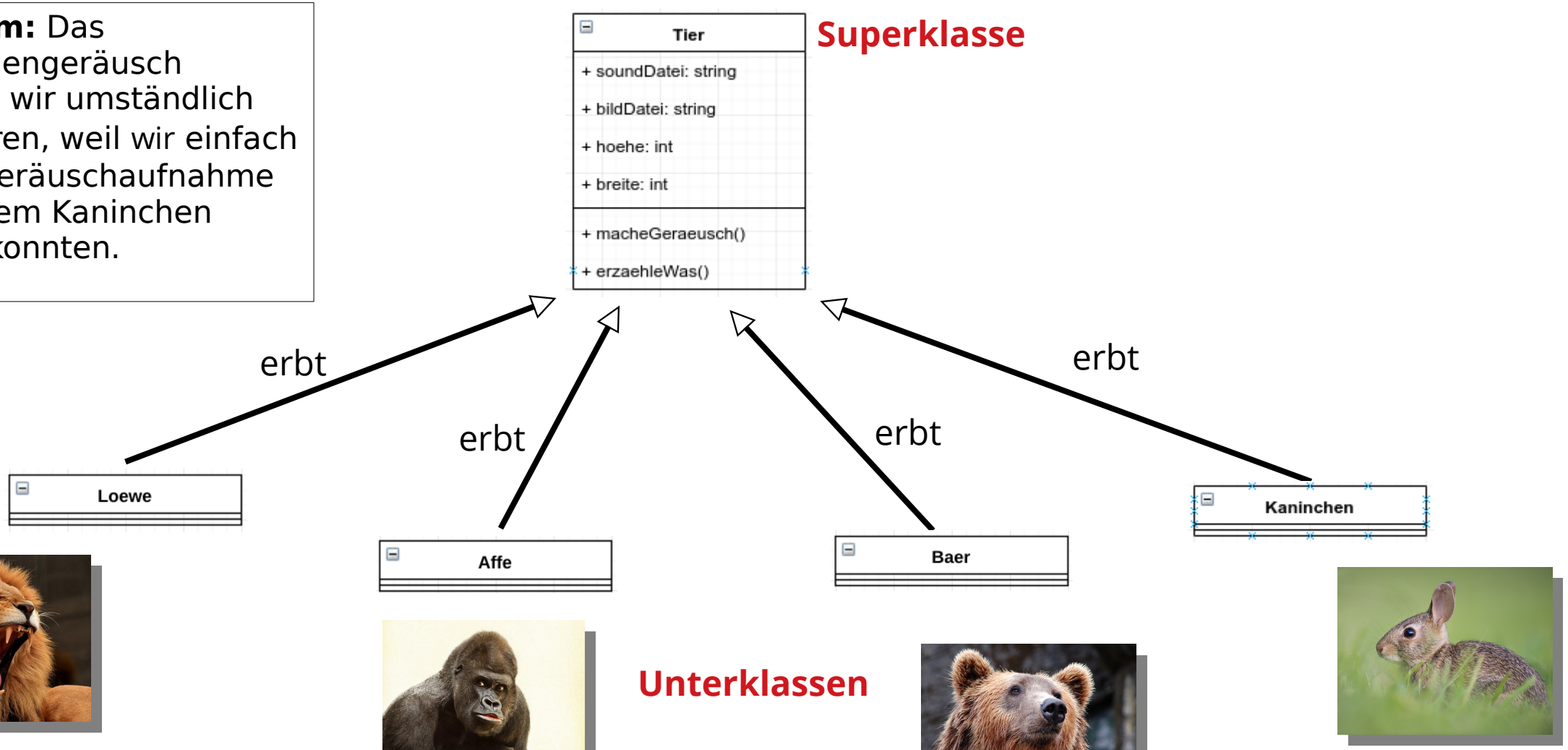
erbt

erbt

erbt

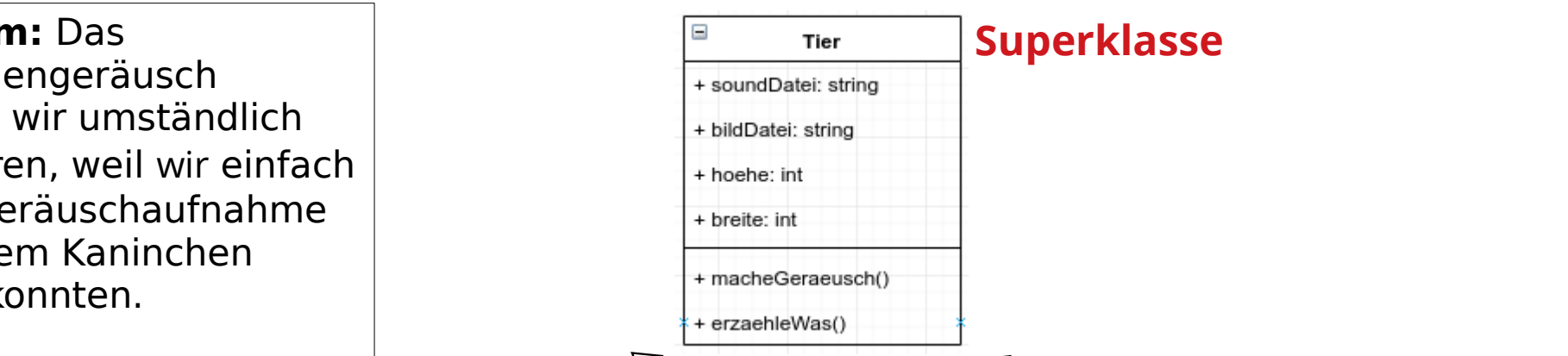
Spezialisierung: Überschreiben

Problem: Das Kaninchengeräusch müssen wir umständlich simulieren, weil wir einfach keine Geräuschaufnahme von einem Kaninchen finden konnten.



Spezialisierung: Überschreiben

Problem: Das Kaninchengeräusch müssen wir umständlich simulieren, weil wir einfach keine Geräuschaufnahme von einem Kaninchen finden konnten.



Superklasse

erbt

erbt

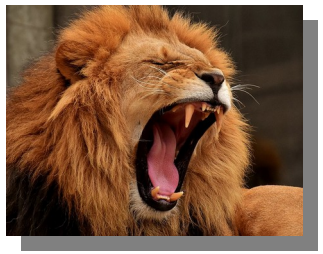
erbt

erbt

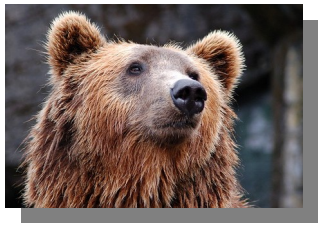
Loewe

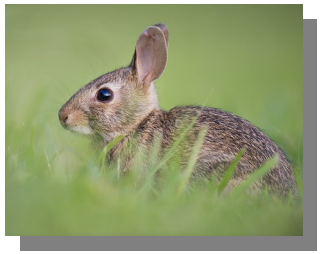
Affe

Baer



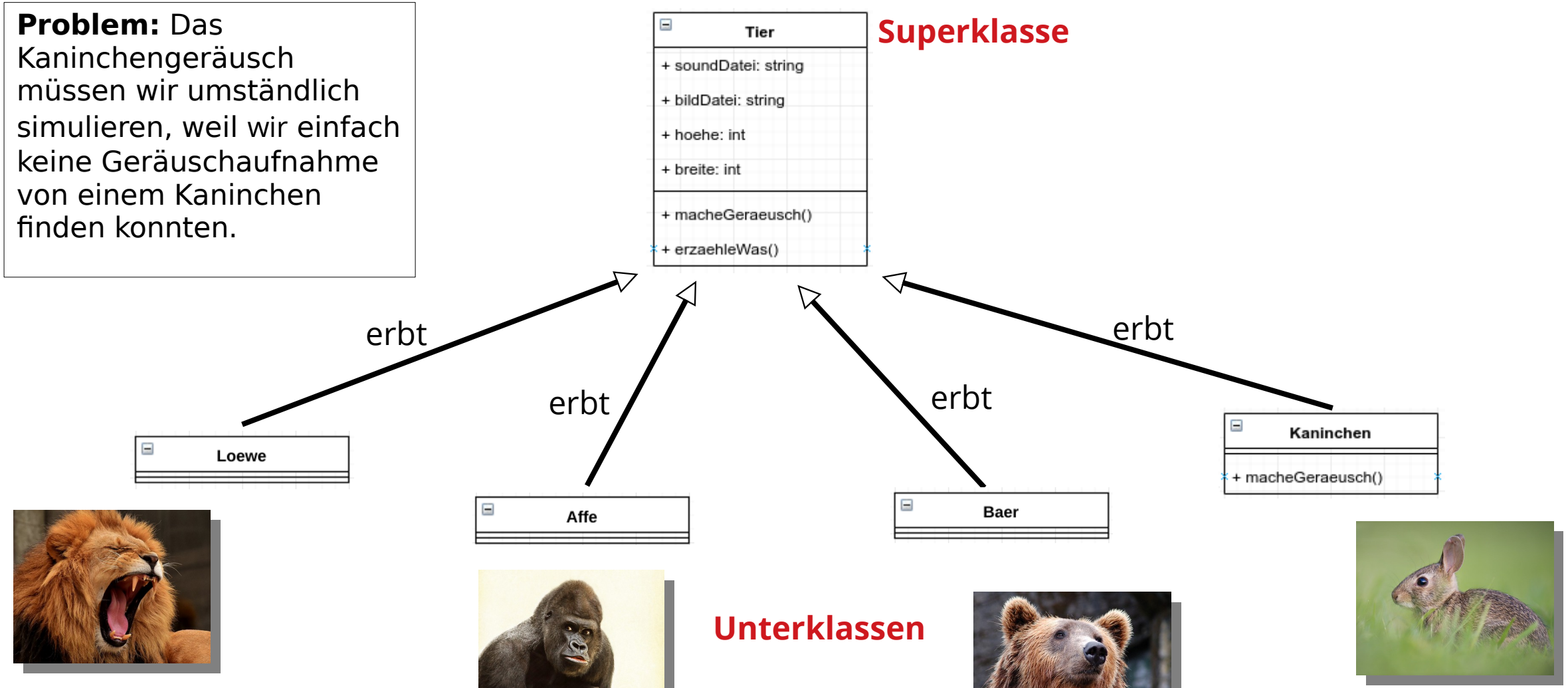
Unterklassen





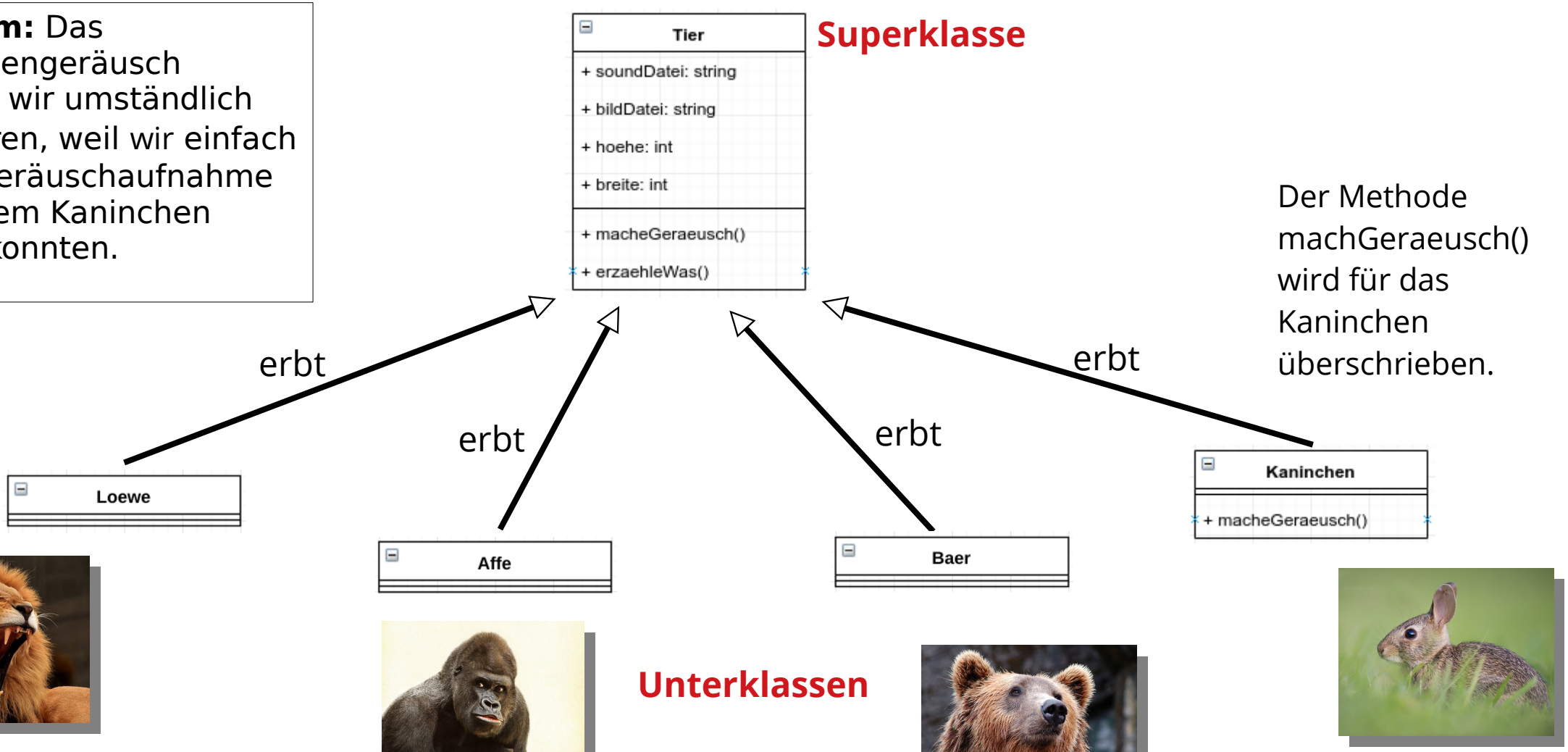
Spezialisierung: Überschreiben

Problem: Das Kaninchengeräusch müssen wir umständlich simulieren, weil wir einfach keine Geräuschaufnahme von einem Kaninchen finden konnten.



Spezialisierung: Überschreiben

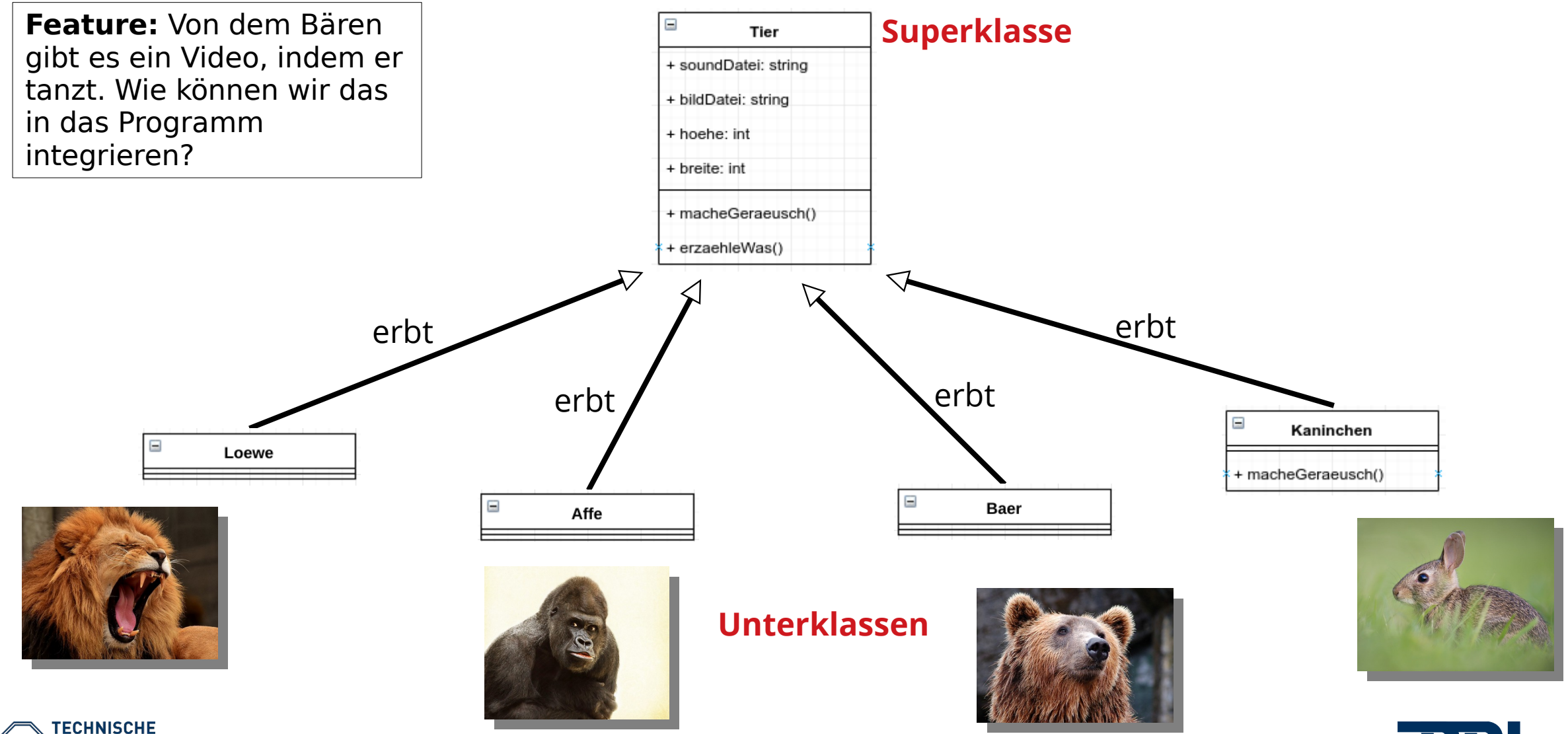
Problem: Das Kaninchengeräusch müssen wir umständlich simulieren, weil wir einfach keine Geräuschaufnahme von einem Kaninchen finden konnten.



Der Methode machGeraeusch() wird für das Kaninchen überschrieben.

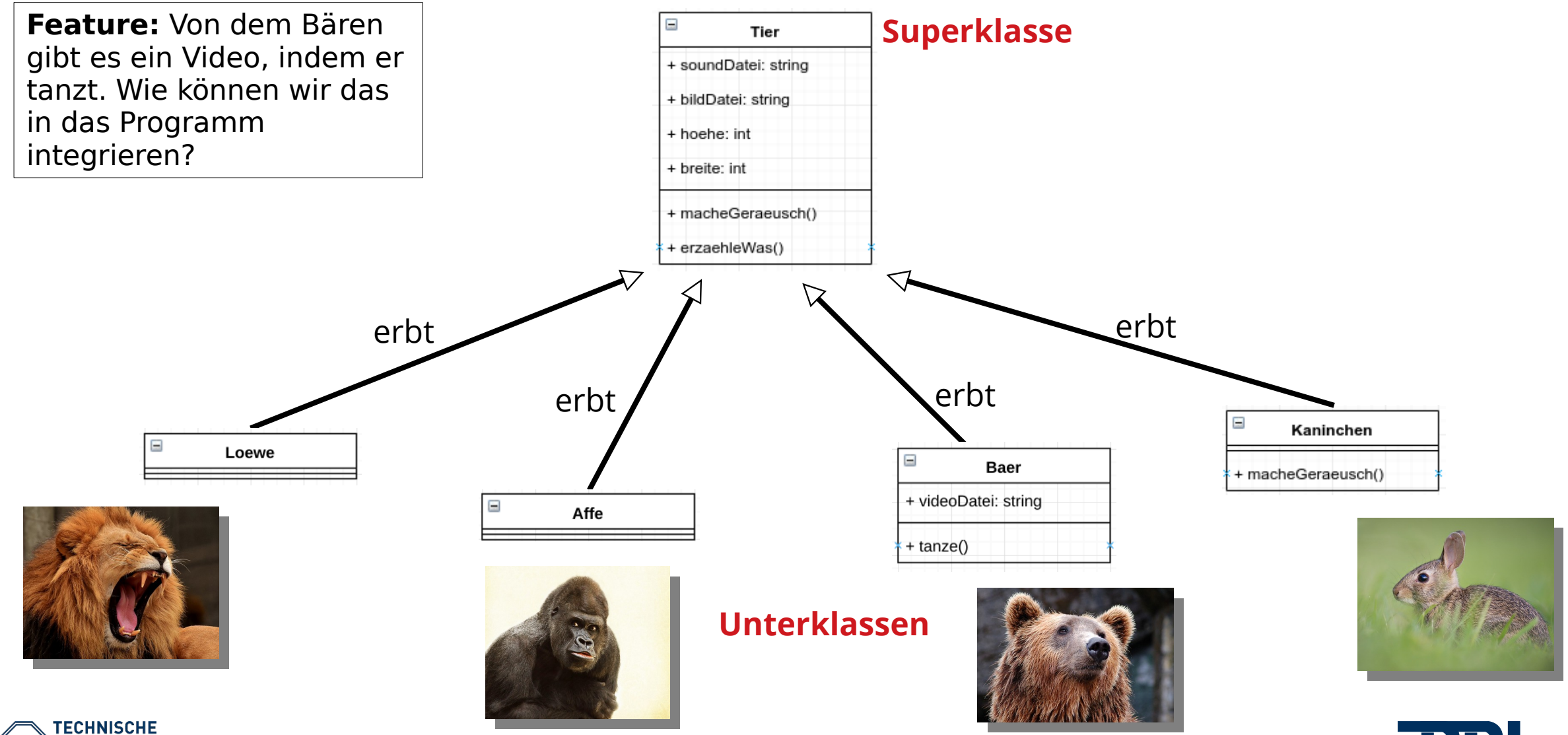
Spezialisierung: Erweitern

Feature: Von dem Bären gibt es ein Video, indem er tanzt. Wie können wir das in das Programm integrieren?



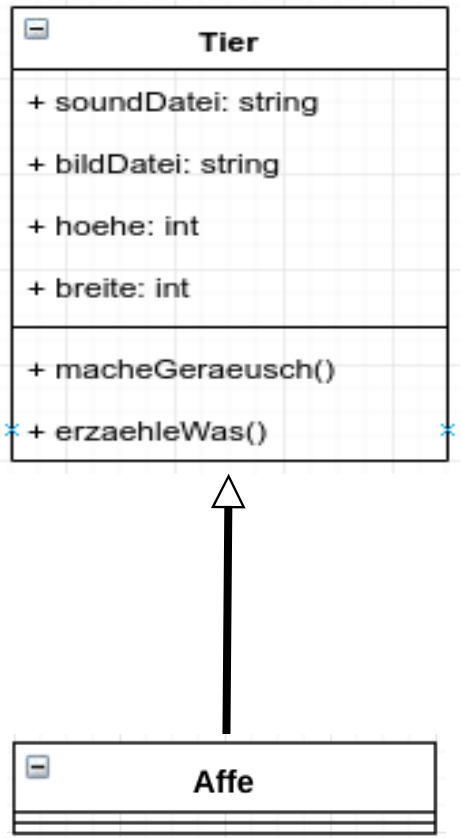
Spezialisierung: Erweitern

Feature: Von dem Bären gibt es ein Video, indem er tanzt. Wie können wir das in das Programm integrieren?



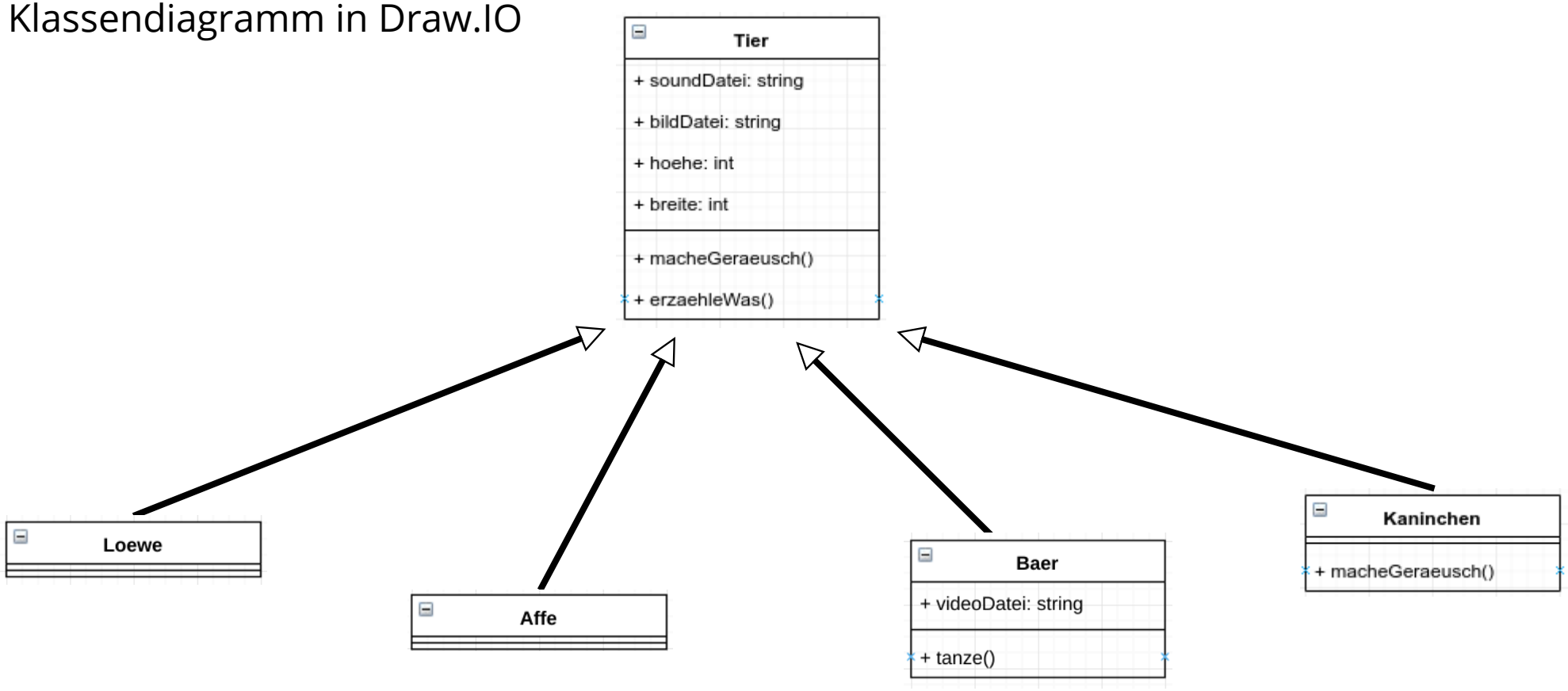
Notation in UML

- In der Unified Modeling Language (**UML**) wird eine Vererbungsbeziehung durch einen Pfeil mit einer dreieckigen Spitze dargestellt,
- Dieser zeigt von der Unterklasse zur Superklasse.



Objektorientierte Klasse ableiten

Modellieren Sie folgendes Klassendiagramm in Draw.IO

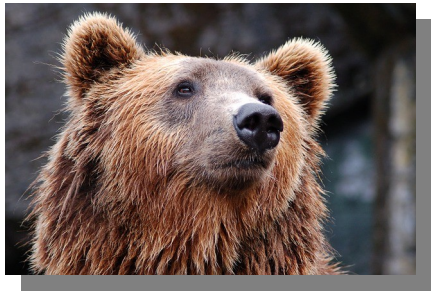


Ein kleiner Cut an dieser Stelle

- Lernziele (BBQ, SE-**, WP)
  - Zusammenhang zwischen Klassen und Objekten wiedergeben
  - Klassen und Objekte in UML darstellen
  - Objekte analysieren und in Klassen generalisieren
  - Vererbung und Unterklassen spezialisieren
  - Modellieren in UML (Kardinalität, Aggregation, Abhängigkeiten)

Programme objektorientiert entwerfen

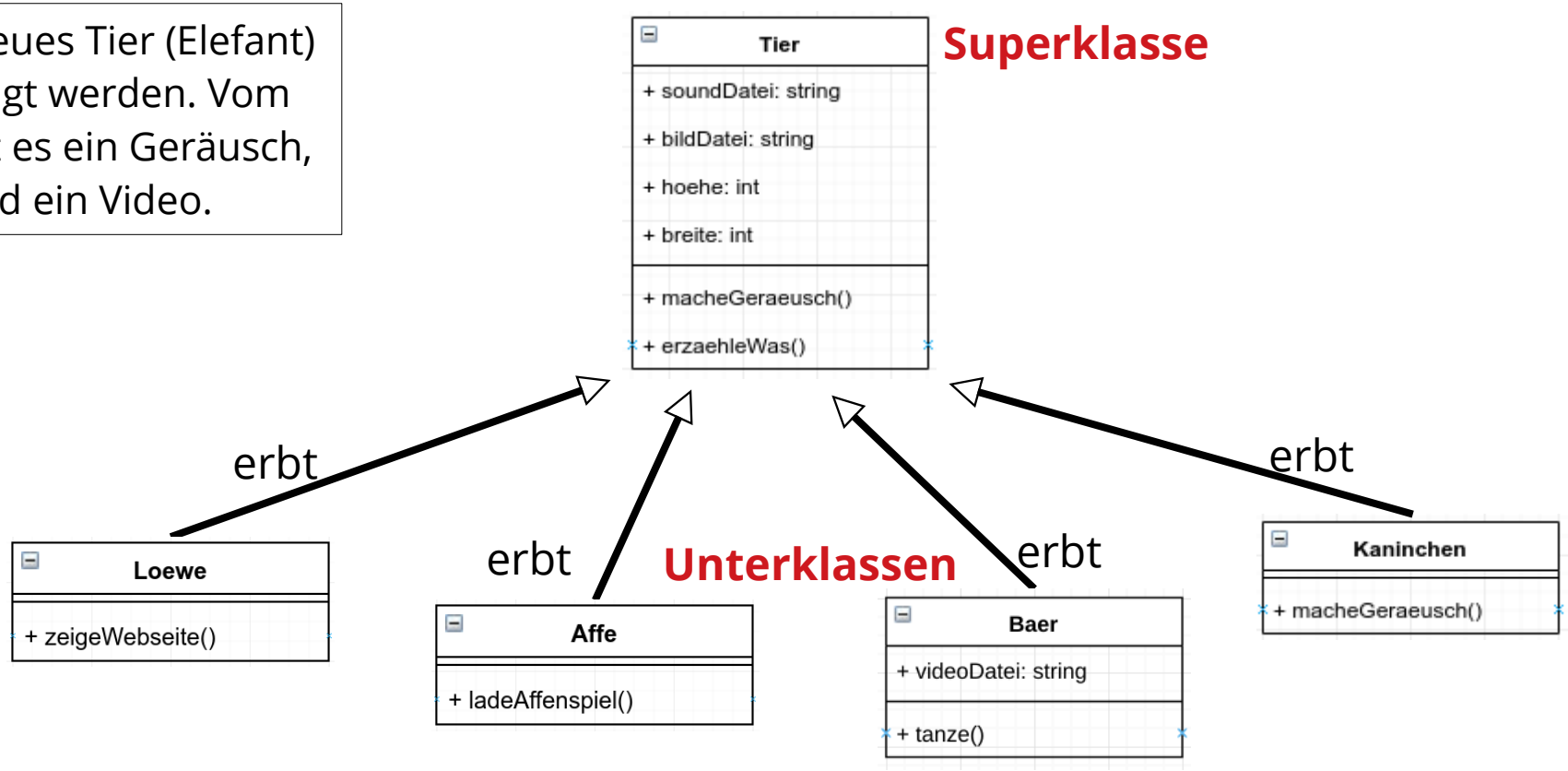
Beispiel: Exponat Wilde Tiere



Was soll es tun:
Sie berühren ein Bild. Das Tier brüllt oder es wird Ihnen etwas über das Tier erzählt!

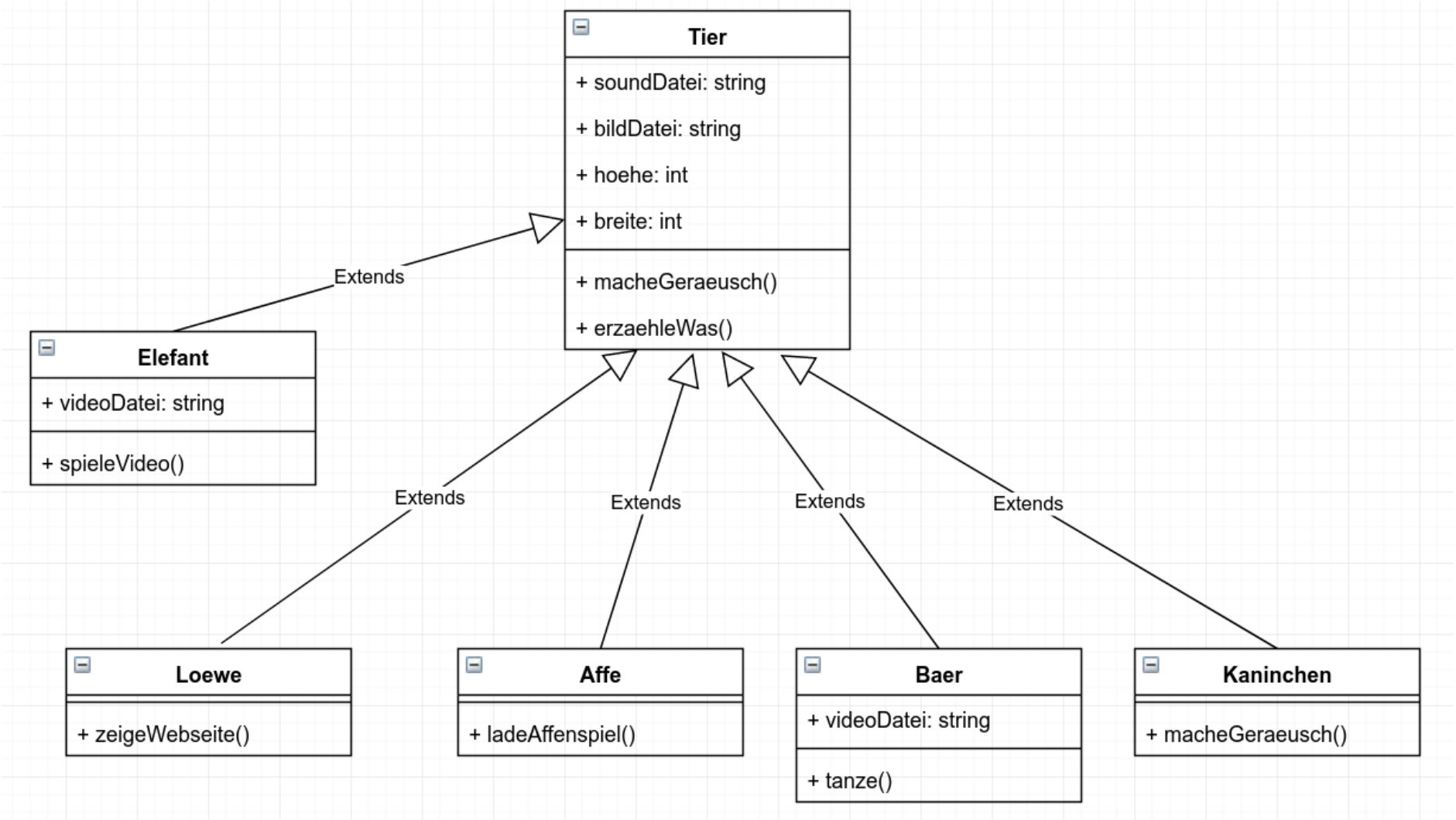
Wiederholung: Spezialisierung

Zusatz: Ein neues Tier (Elefant) soll hinzugefügt werden. Vom Elefanten gibt es ein Geräusch, einen Text und ein Video.

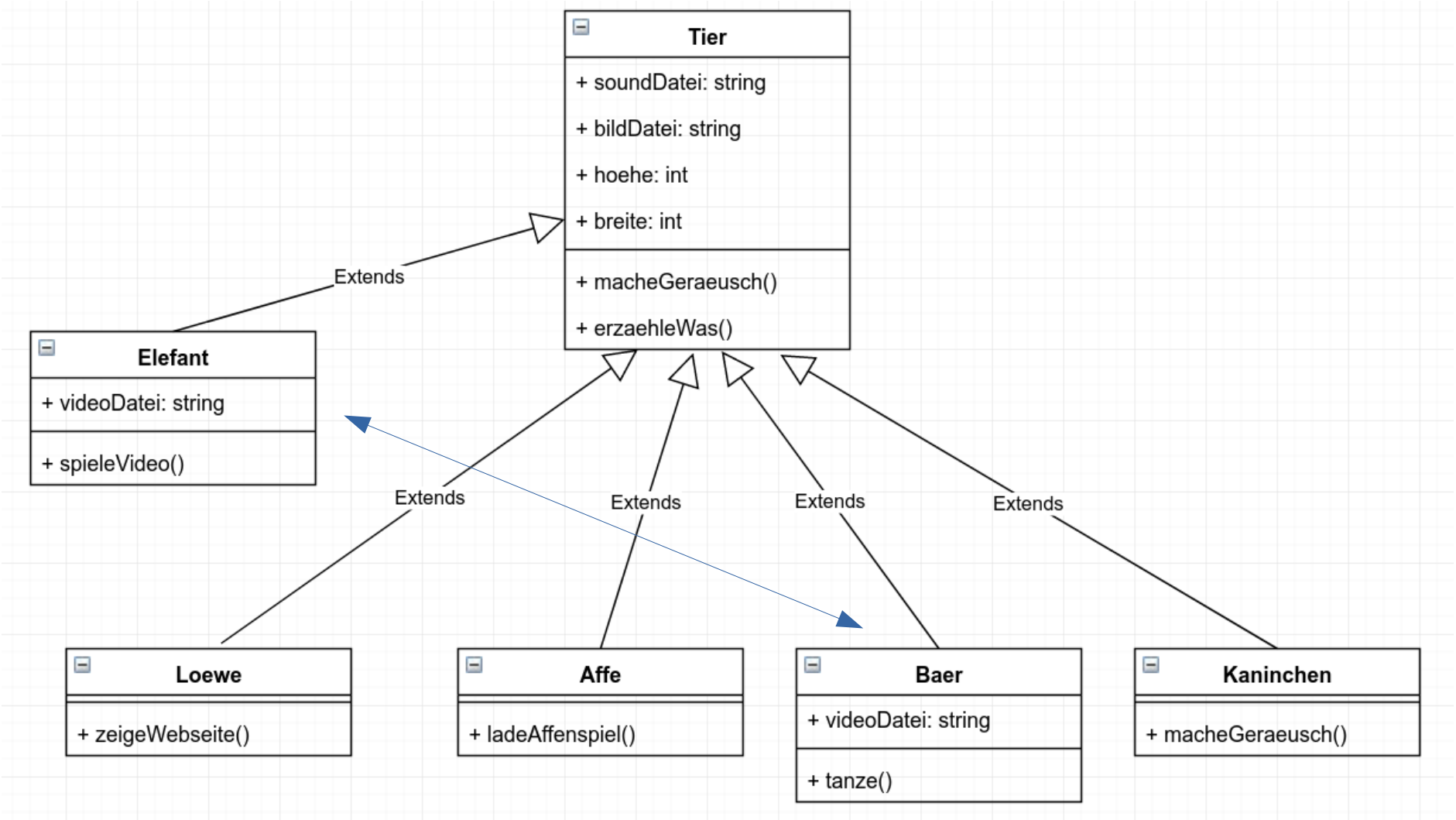


Passen Sie das Klassendiagramm in Draw.IO an die neue Situation an.
→ Aufgabe1.drawio

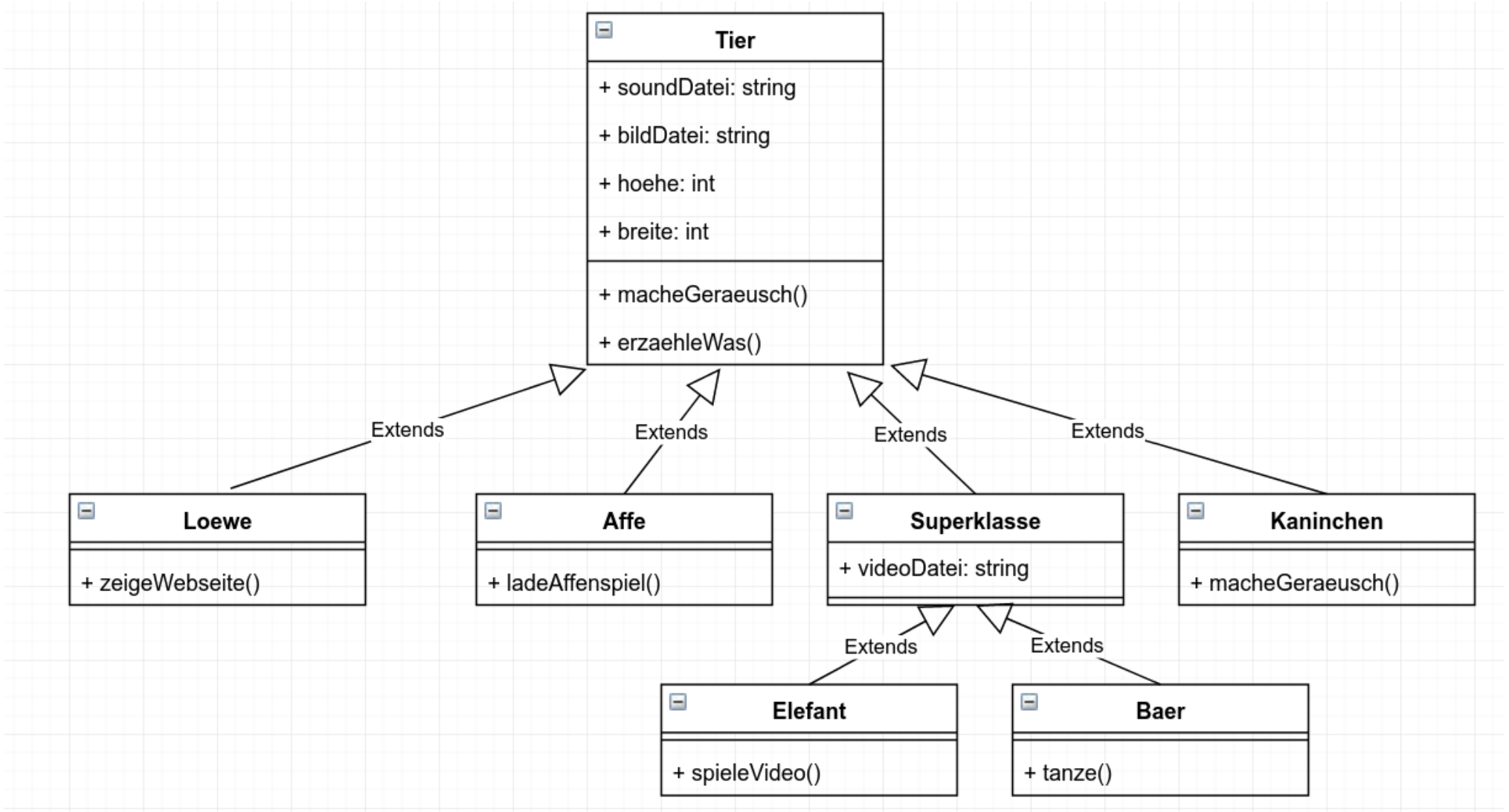
Lösung


Tier
+ soundDatei: string
+ bildDatei: string
+ hoehe: int
+ breite: int
+ macheGeraeusch()
+ erzaehleWas()
Extends
Elefant
+ videoDatei: string
+ spieleVideo()
Extends
Extends
Extends
Extends
Loewe
+ zeigeWebseite()
Affe
+ ladeAffenspiel()
Baer
+ videoDatei: string
+ tanze()
Kaninchen
+ macheGeraeusch()

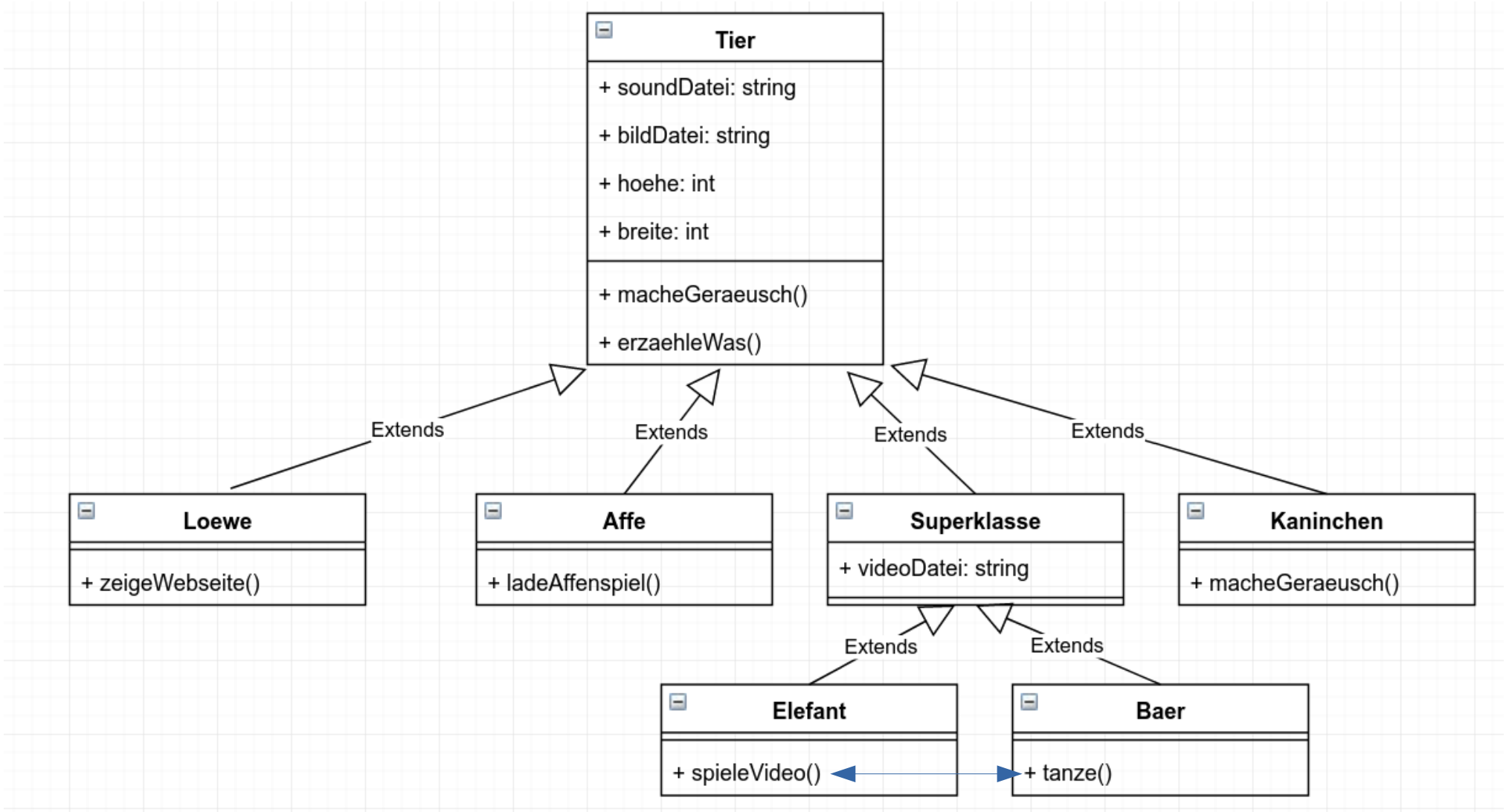
Lösung


Tier
+ soundDatei: string
+ bildDatei: string
+ hoehe: int
+ breite: int
+ macheGeraeusch()
+ erzaehleWas()
Extends
Elefant
+ videoDatei: string
+ spieleVideo()
Extends
Extends
Extends
Extends
Loewe
+ zeigeWebseite()
Affe
+ ladeAffenspiel()
Baer
+ videoDatei: string
+ tanze()
Kaninchen
+ macheGeraeusch()

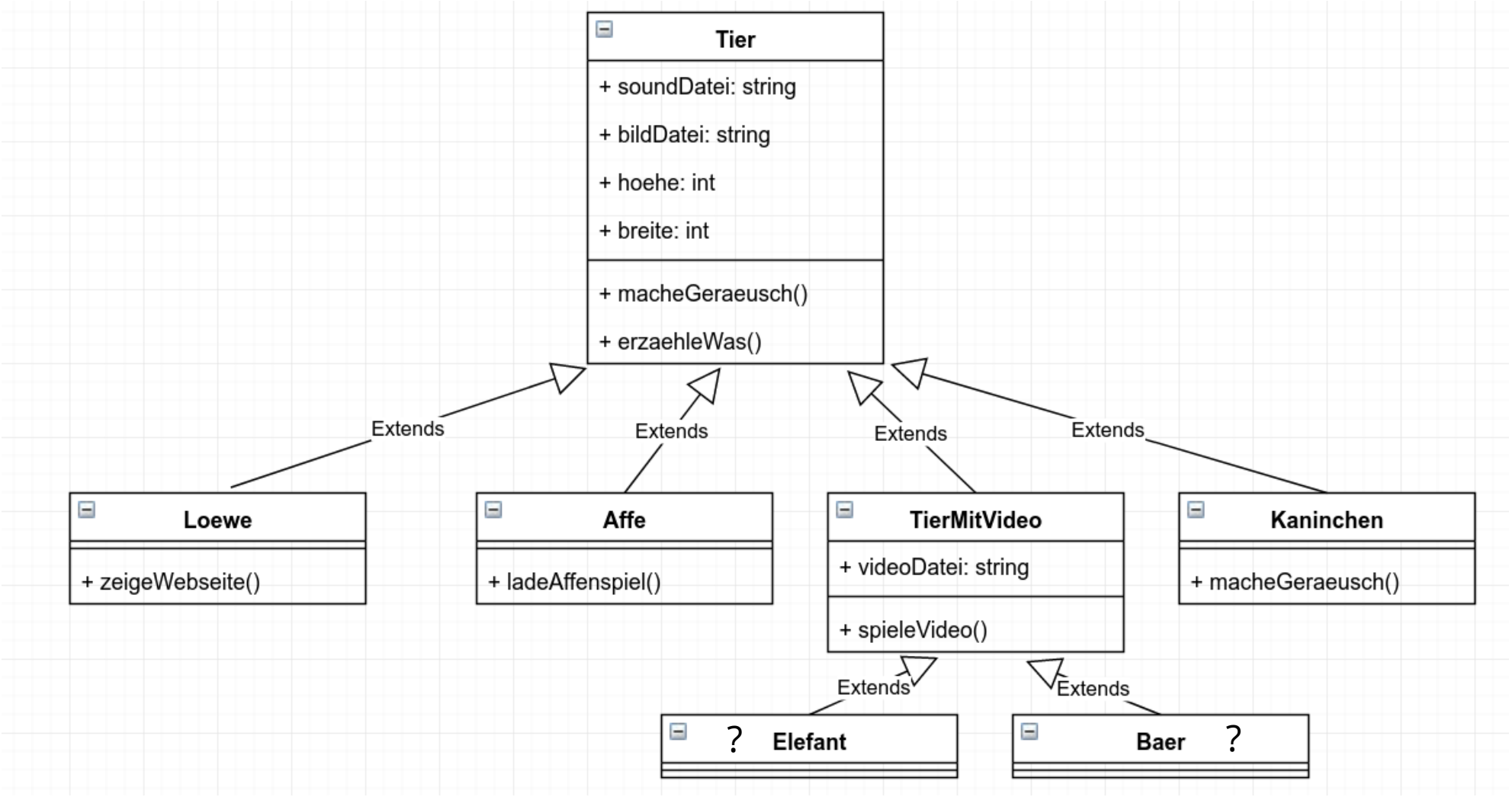
Lösung


Tier
+ soundDatei: string
+ bildDatei: string
+ hoehe: int
+ breite: int
+ macheGeraeusch()
+ erzaehleWas()
Extends
Extends
Extends
Extends
Loewe
+ zeigeWebseite()
Affe
+ ladeAffenspiel()
Superklasse
+ videoDatei: string
Kaninchen
+ macheGeraeusch()
Extends
Extends
Elefant
+ spieleVideo()
Baer
+ tanze()

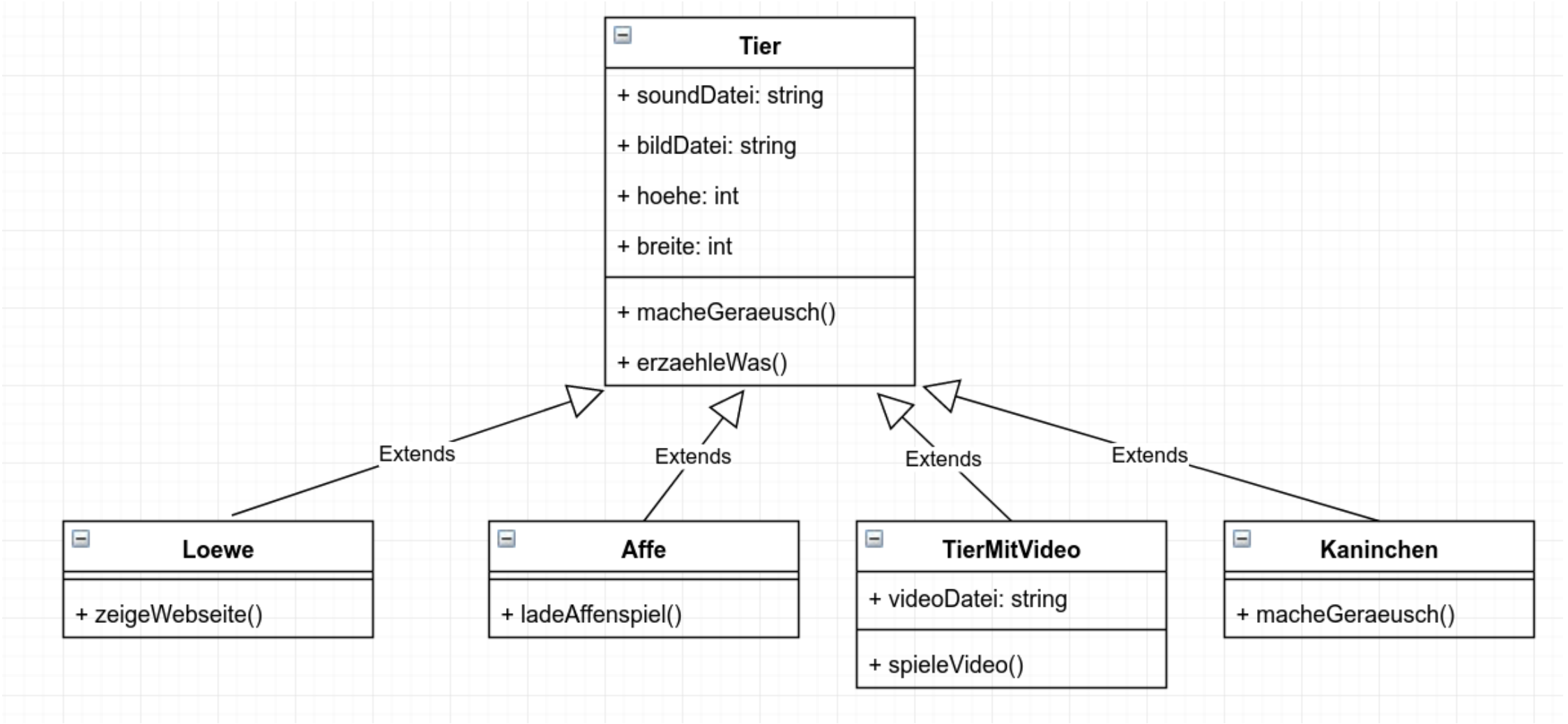
Lösung


Tier
+ soundDatei: string
+ bildDatei: string
+ hoehe: int
+ breite: int
+ macheGeraeusch()
+ erzaehleWas()
Extends
Extends
Extends
Extends
Loewe
+ zeigeWebseite()
Affe
+ ladeAffenspiel()
Superklasse
+ videoDatei: string
Kaninchen
+ macheGeraeusch()
Extends
Extends
Elefant
+ spieleVideo()
Baer
+ tanze()

Lösung

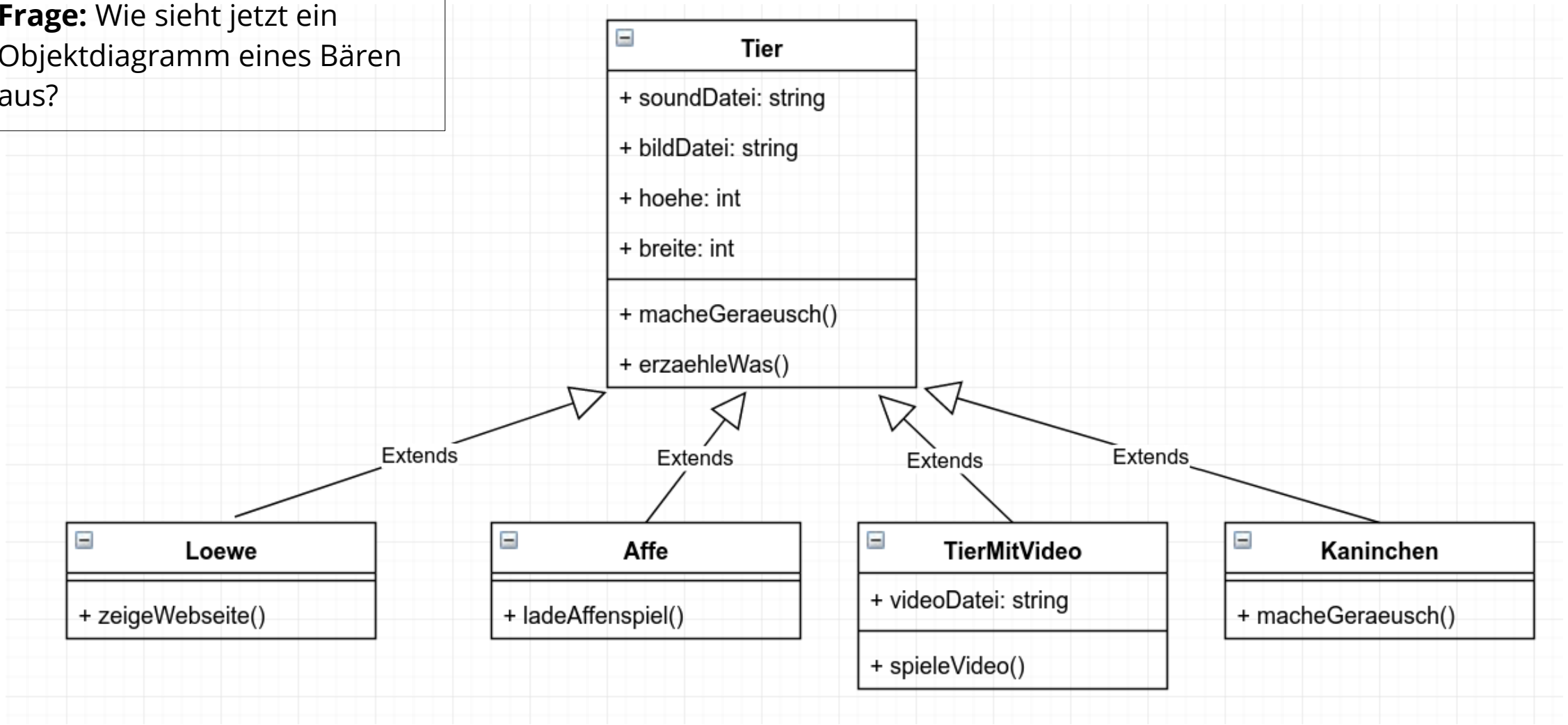

Tier
+ soundDatei: string
+ bildDatei: string
+ hoehe: int
+ breite: int
+ macheGeraeusch()
+ erzaehleWas()
Extends
Extends
Extends
Extends
Loewe
+ zeigeWebseite()
Affe
+ ladeAffenspiel()
TierMitVideo
+ videoDatei: string
+ spieleVideo()
Kaninchen
+ macheGeraeusch()
Extends
Extends
? Elefant
Baer ?

Lösung


Tier
+ soundDatei: string
+ bildDatei: string
+ hoehe: int
+ breite: int
+ macheGeraeusch()
+ erzaehleWas()
Extends
Extends
Extends
Extends
Loewe
+ zeigeWebseite()
Affe
+ ladeAffenspiel()
TierMitVideo
+ videoDatei: string
+ spieleVideo()
Kaninchen
+ macheGeraeusch()

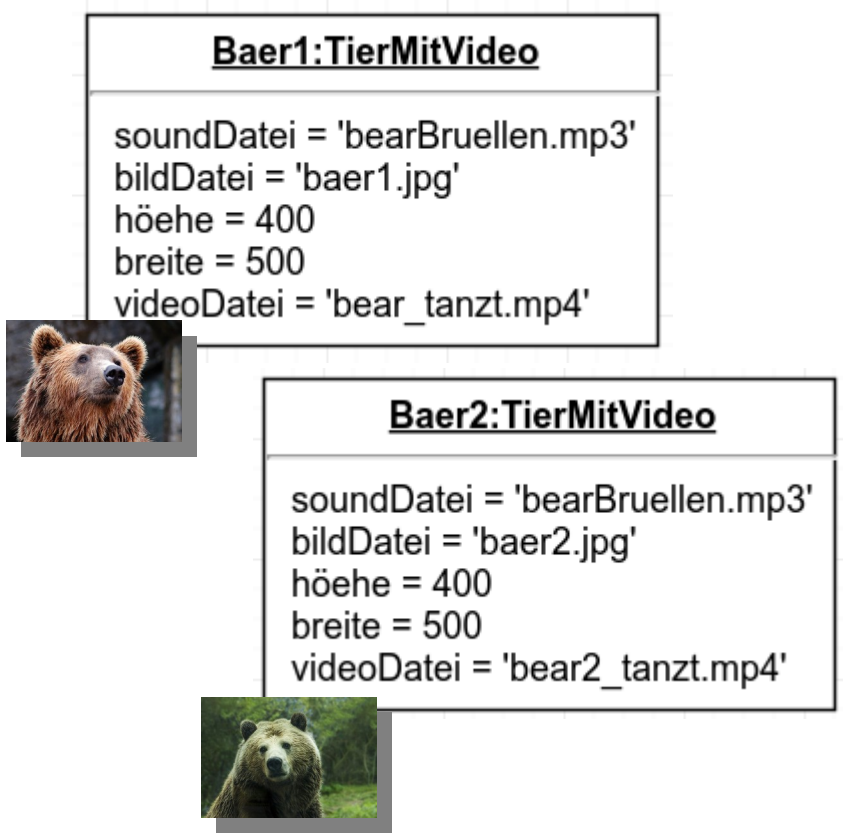
Lösung

Frage: Wie sieht jetzt ein Objektdiagramm eines Bären aus?

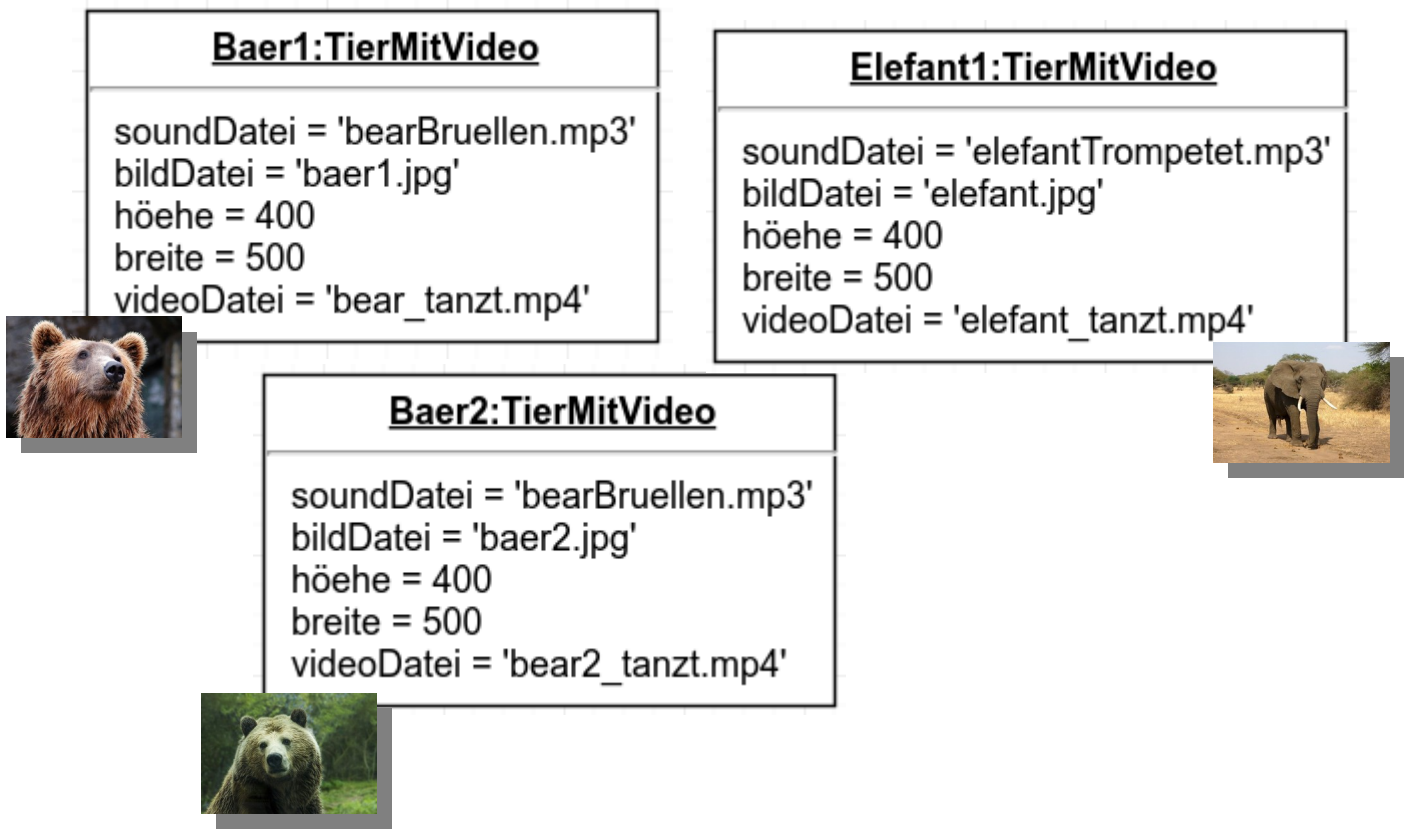


Objektdiagramme

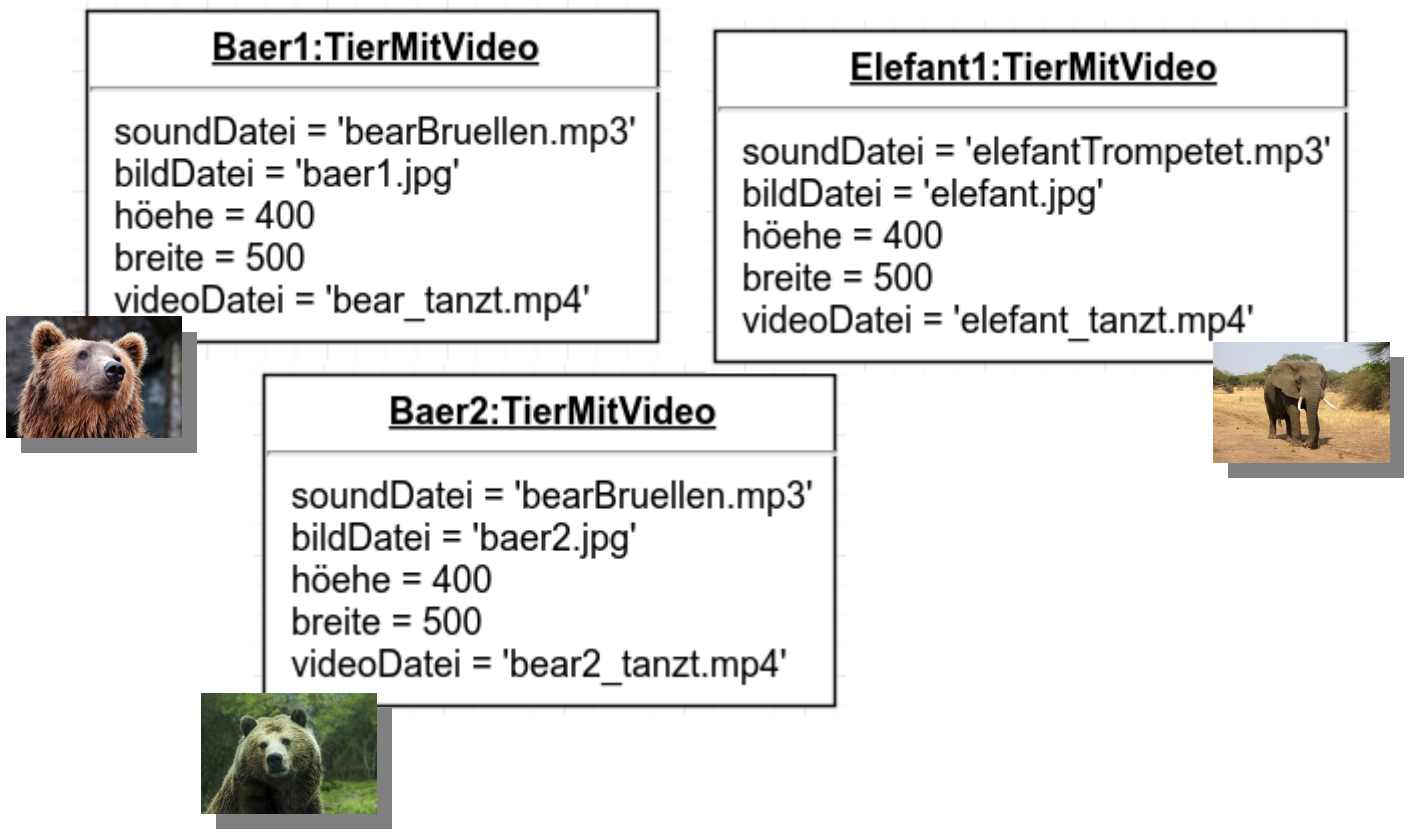
Objektdiagramme



Objektdiagramme



Objektdiagramme



Fragen:

- Wie unterscheiden wir jetzt Elefanten und Bären?
- Warum haben die Objektdiagramme keine Methoden?

Assoziationen

UML - Assoziationen

Assoziation

- Es gibt eine Assoziation zwischen den Klassen A und B, wenn es eine semantische Beziehung zwischen den Klassen gibt.
- Üblicherweise werden Assoziationen durch Referenzen realisiert
  - d.h. eine der Klassen hat ein Attribut vom Typ der anderen Klasse.

Zuerst eine neue Klasse...

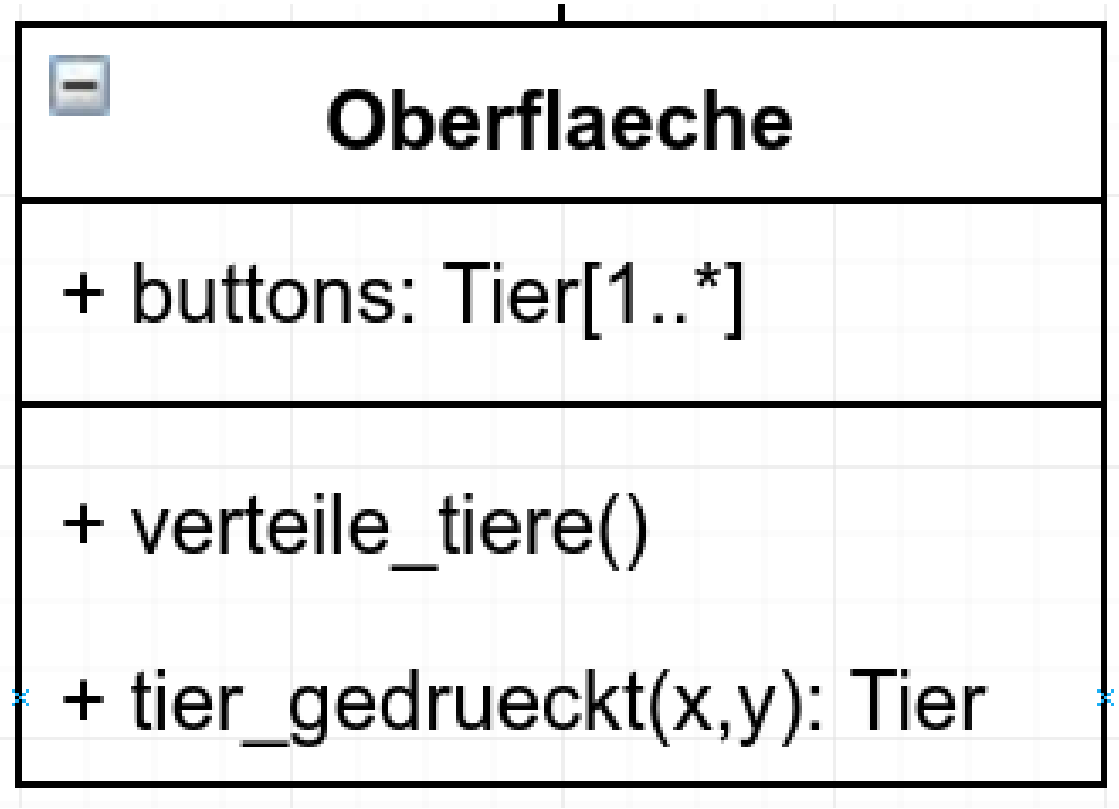
Oberfläche

- Wir wollen eine Klasse modellieren, die die Oberfläche unseres Spiels verwaltet.
- Diese soll alle Tierobjekte enthalten, eine Methode zum Verteilen der Tiere auf der Oberfläche und eine Methode, die uns sagt welches Tier gedrückt wurde.

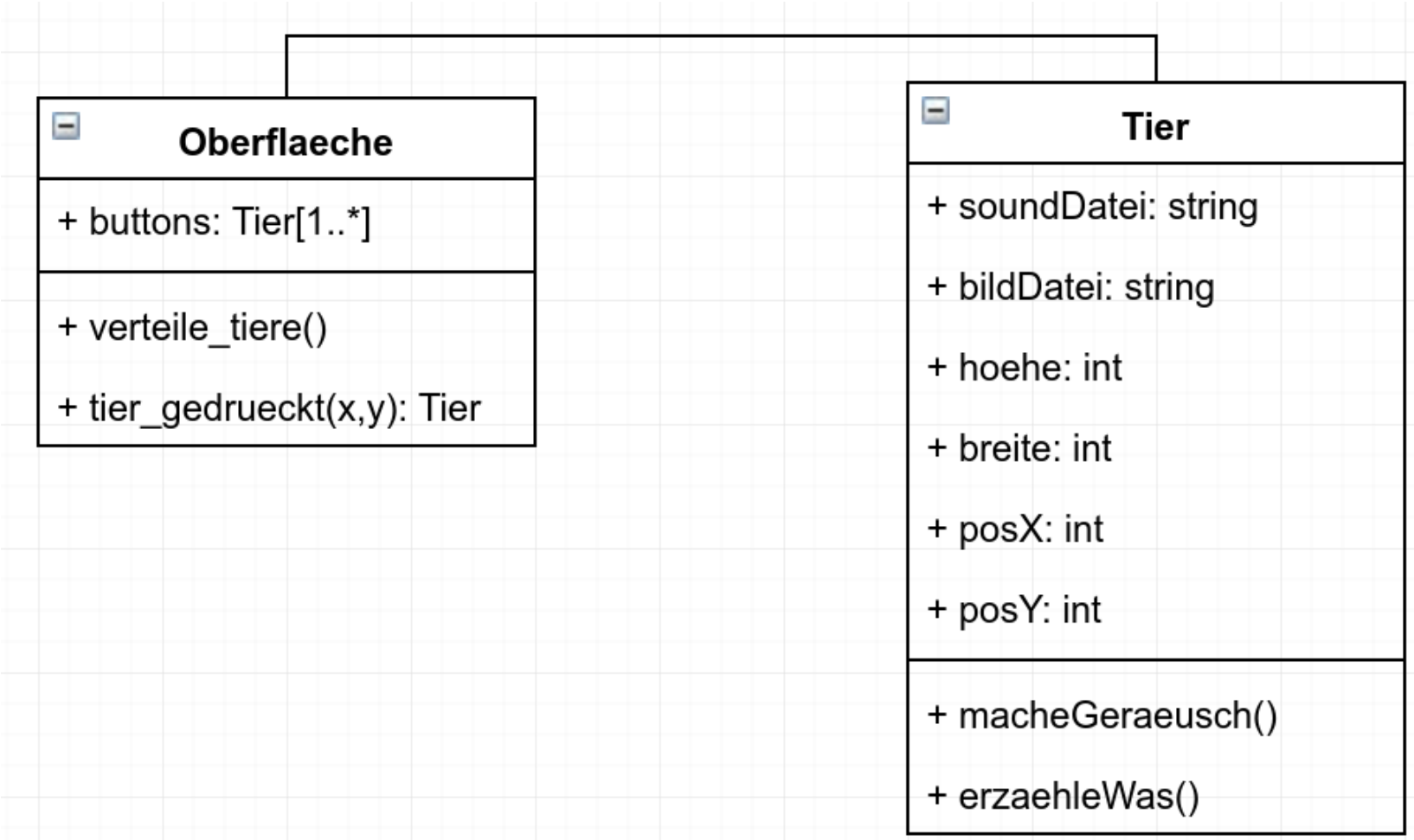
Zuerst eine neue Klasse...

Oberfläche

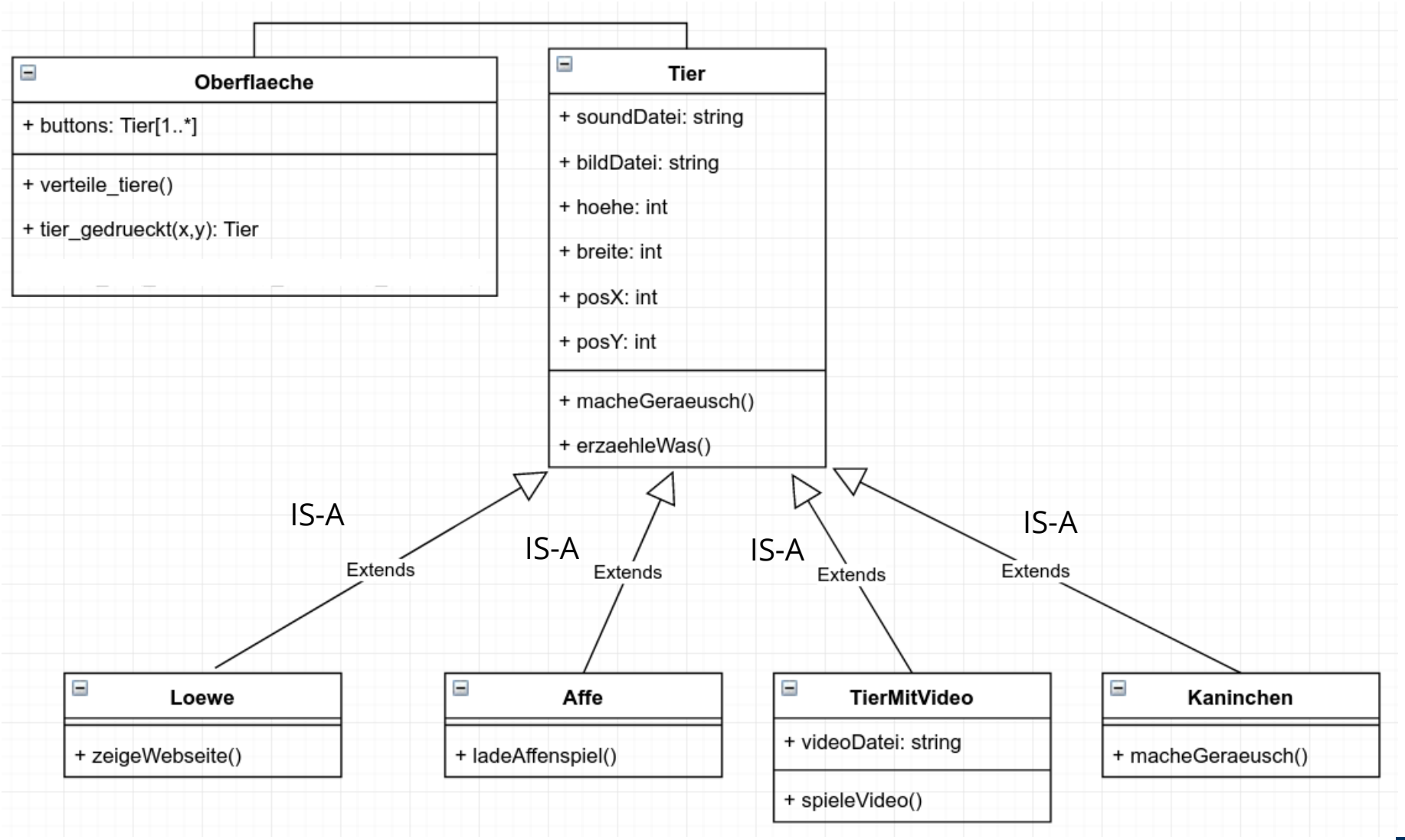
- Wir wollen eine Klasse modellieren, die die Oberfläche unseres Spiels verwaltet.
- Diese soll alle Tierobjekte enthalten, eine Methode zum Verteilen der Tiere auf der Oberfläche und eine Methode, die uns sagt welches Tier gedrückt wurde.



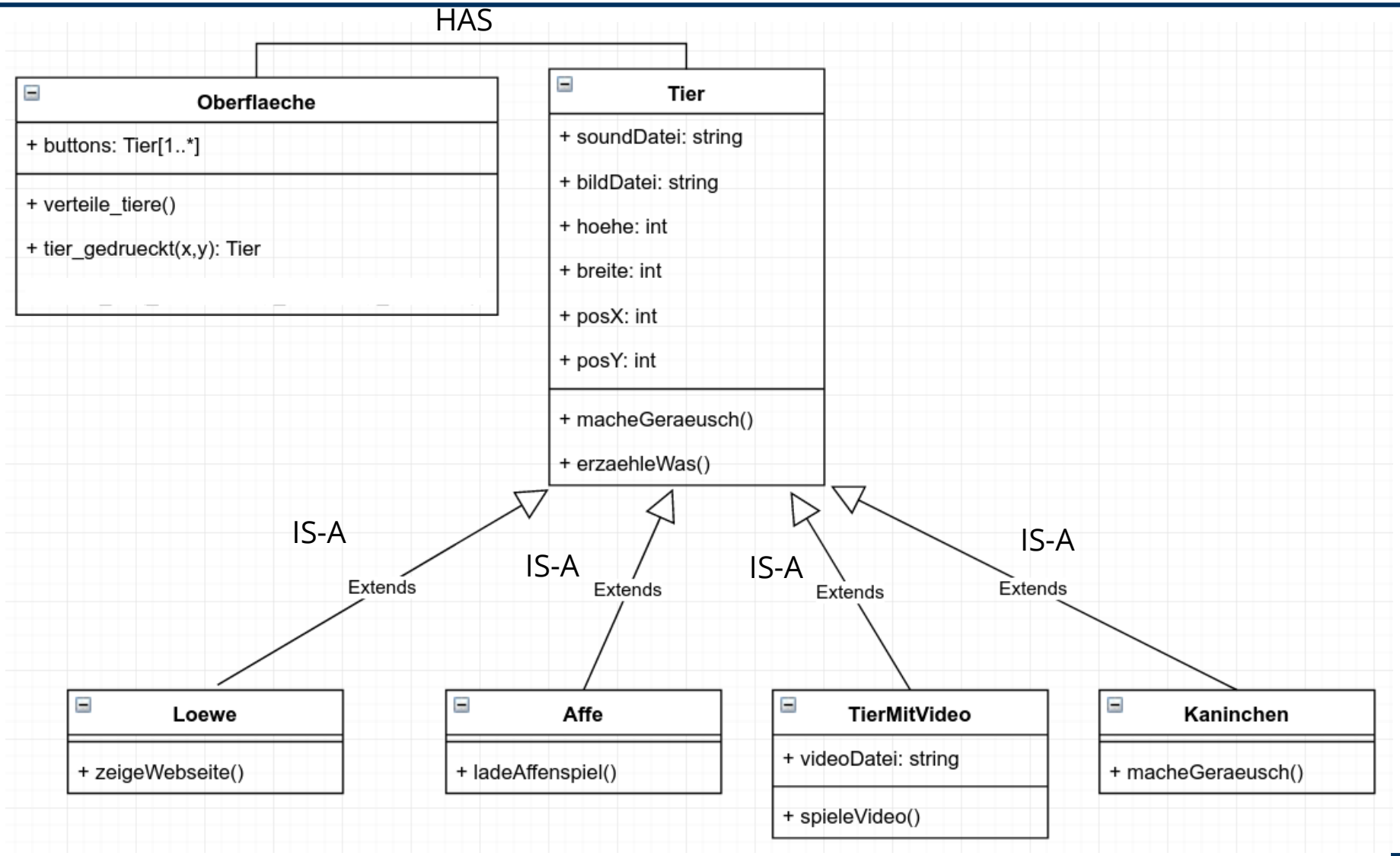
Assoziation in unserem Beispiel



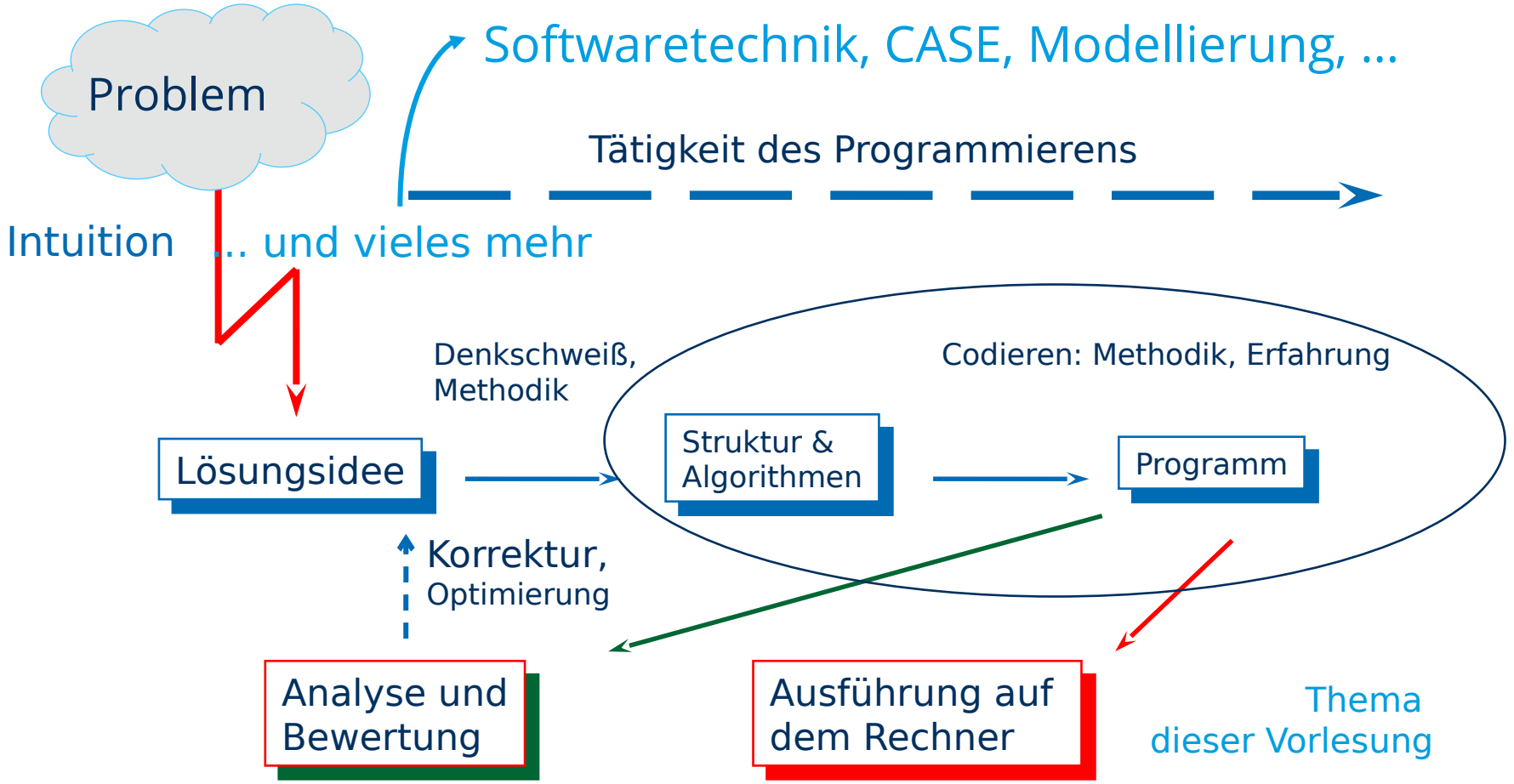
Assoziation in unserem Beispiel



Assoziation in unserem Beispiel

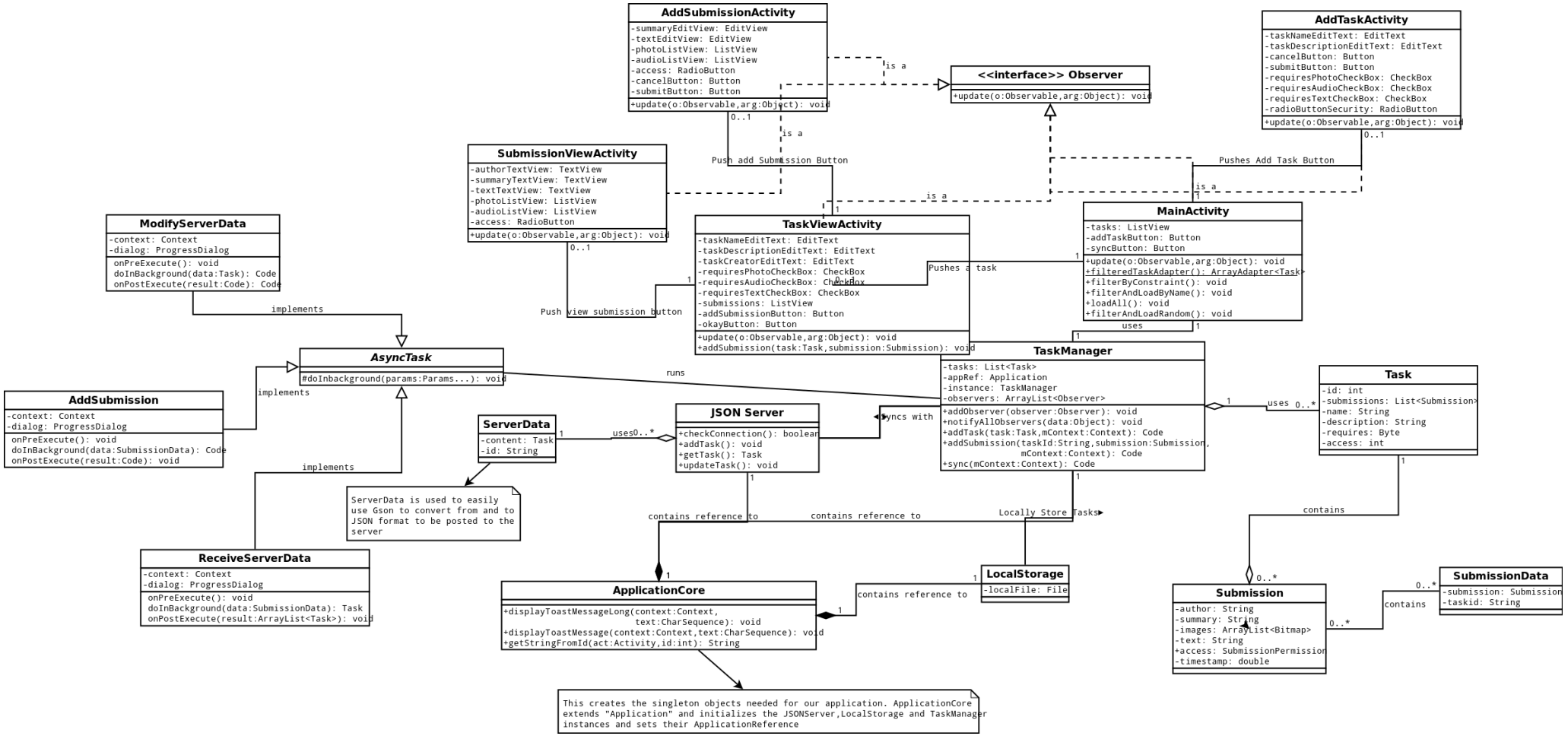


Vom *Problem* zum *Programm* und wieder zurück



Programmieren im Kleinen

Einfache Android-Applikation



https://github.com/CMPUT301F12T06/classproject/wiki/3.)-Object-Oriented-Analysis-(UML-Class-Diagram)

Was wir alles nicht behandelt haben…

Objektorientierung

- *Kapselung*
- Mehrfachvererbung
- Abstrakte Klassen
- Interfaces
- *Polymorphe Funktionen*

UML

- Use-Case-Diagramme
- Aktivitätsdiagramme
- *Multiplizitäten*
- Navigierbarkeit
- Rollen
- …

Was Sie mitnehmen und vertiefen sollten…

- Lernziele (BBQ, SE-**, WP)
  - Zusammenhang zwischen Klassen und Objekten wiedergeben
  - Klassen und Objekte in UML darstellen
  - Objekte analysieren und in Klassen generalisieren
  - Vererbung und Unterklassen spezialisieren
  - Modellieren in UML (Kardinalität, Aggregation, Abhängigkeiten)
  - Assoziationen nutzen, um Beziehungen von Klassen zu beschreiben.
  - Klassennamen und Modellierung nicht nach der Realität, sondern nach der Verwendung im Programm vornehmen.
- Lernziele (SE-GY)
  - Zusatzübung und Theorie-Input
    - Zugriffsmodifikatoren anwenden
    - Zugriffsmodifikatoren in UML darstellen