TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Professur für Didaktik der Informatik
Dr. Thiemo Leonhardt

Programmierparadigmen
Objektorientierte Programmierung

Professur für **Didaktik der Informatik**

Was Sie können sollten…

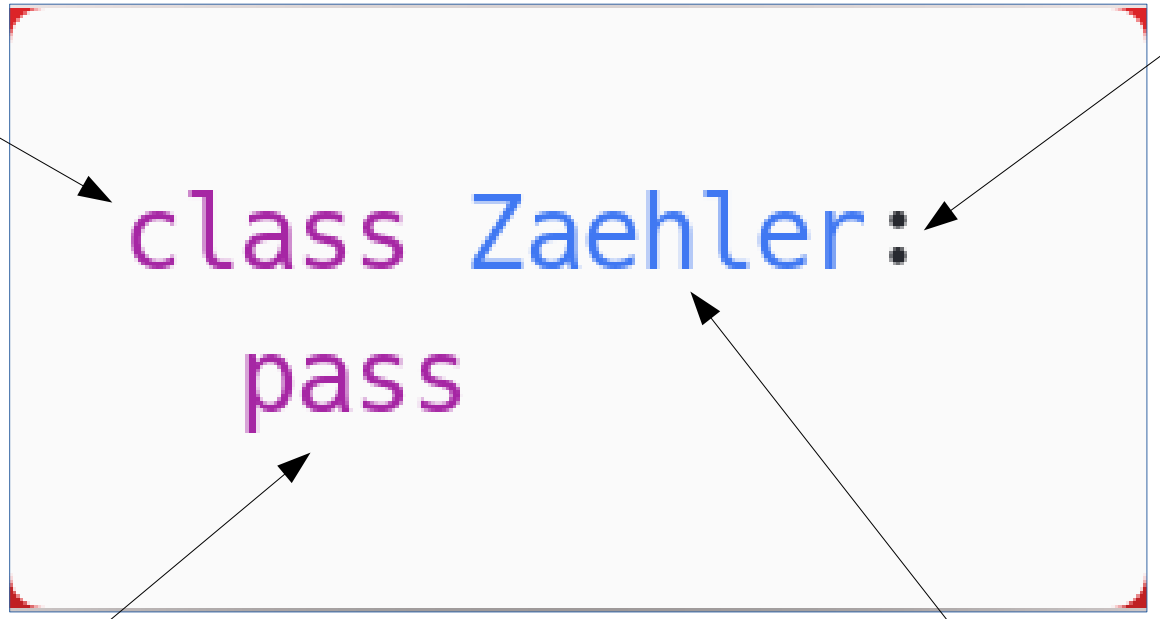
- Objektorientierte Modellierung
  - Zusammenhang zwischen Klassen und Objekten wiedergeben
  - Klassen und Objekte in UML darstellen
  - Objekte analysieren und zu Klassen überführen
  - Klassen generalisieren
  - Vererbung modellieren in UML
  - Unterklassen spezialisieren

Warum werden auf Objektdiagrammen in UML im Gegensatz zu den Objektkarten in der Schule keine Methoden angegeben?

Klassen und Objekte in Python

Am Anfang steht immer die Definition...

Klassen werden durch das Schlüsselwort **class** eingeleitet.



Doppelpunkt leitet in Python einen neuen Block ein

Der Code der Klasse

Der Name der Klasse

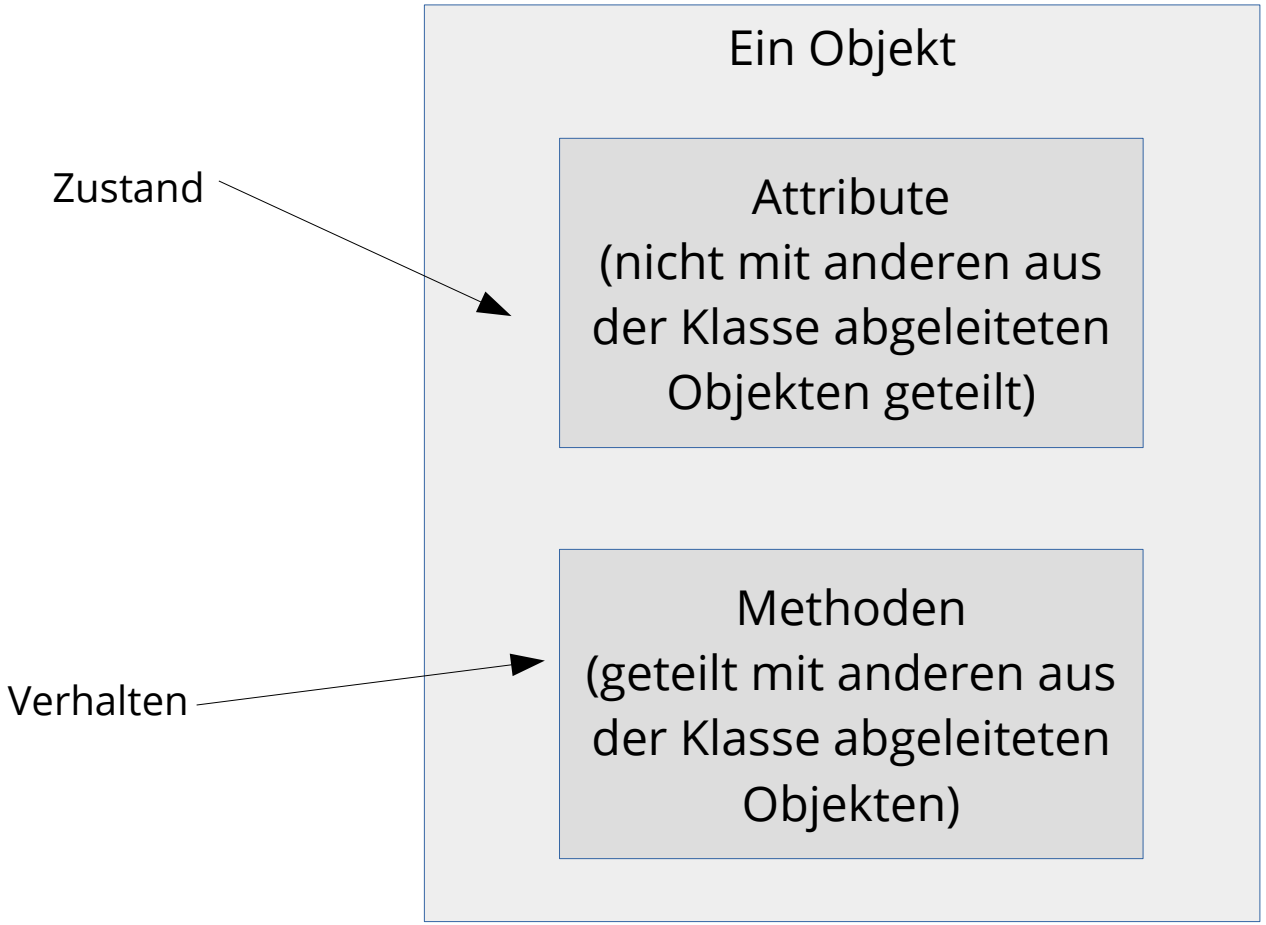
Aufrufen, erzeugen, initialisieren...instanziieren

```
class Zaehler:
  pass

a = Zaehler()
b = Zaehler()
```

Zuweisung an eine Variable

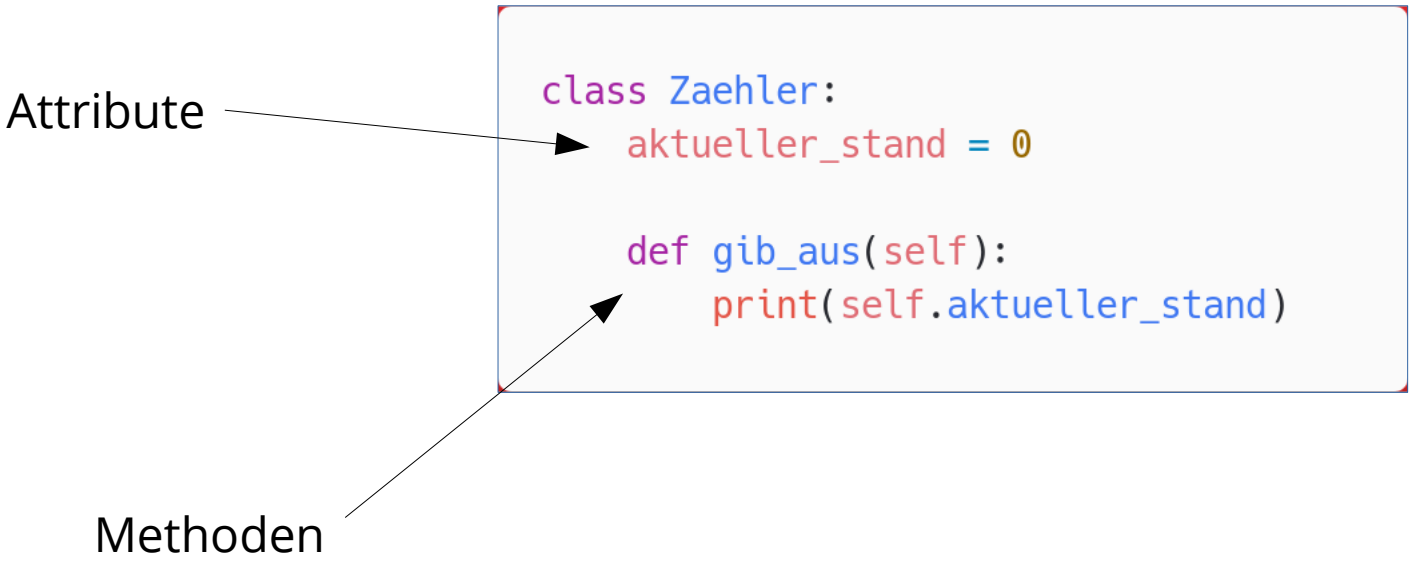
Um ein Objekt zu erzeugen, hängt man dem Klassennamen ein Paar runde Klammern an

Objekte übernehmen das Verhalten, aber nicht den Zustand…



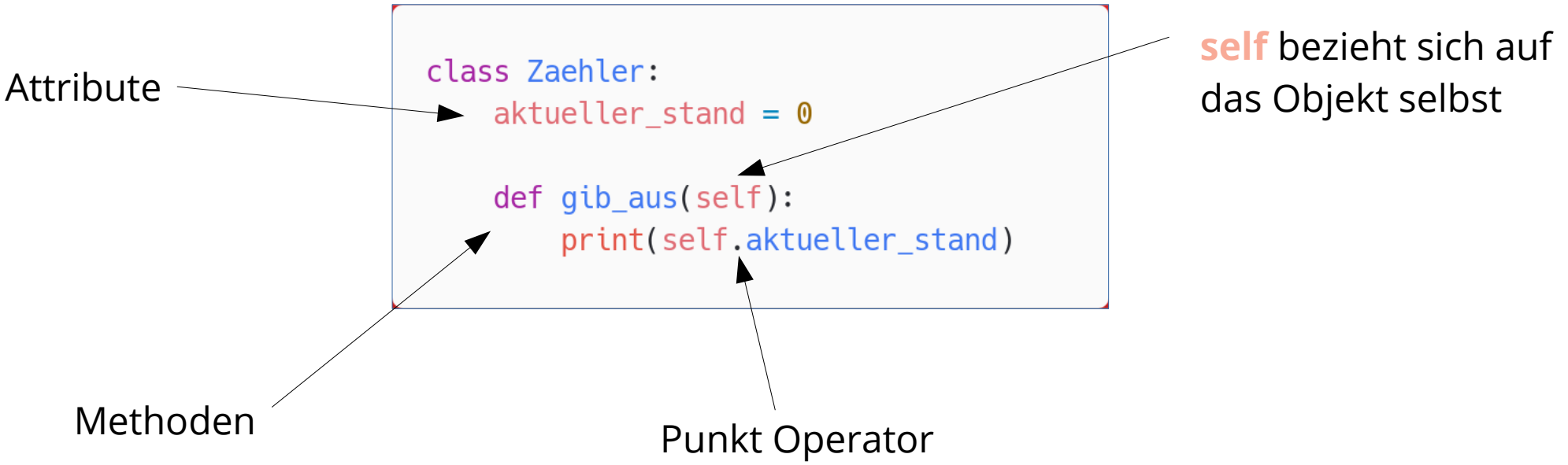
Unterschied Klasse und Objekt unterscheiden!

Wie sieht das nun in einer Klasse aus?



Opal→ zaehler.py

Wie sieht das nun in einer Klasse aus



Opal→ zaehler.py

Vererbung in Python

```
class A:
  pass

class B(A):
  pass
```

Klasse B erbt
alle Attribute
und Methoden
der Klasse A

Spezialisieren - Überschreiben

```
class A:
  def ausgabe(self):
    print('Klasse A')


class B(A):
  def ausgabe(self):
    print('Klasse B')
```

Original Methode der Superklasse

Überschreiben in der Unterklasse

Spezialisieren - Überlagern

```
class A:
  def ausgabe(self):
    print('Klasse A')


class B(A):
  def ausgabe(self):
    super().ausgabe()
    print('Klasse B')
```

Original Methode der Superklasse

Überschreiben in der Unterklasse

Methode der Superklasse wird zuerst ausgeführt

Aufgabe

- Implementieren Sie eine Klasse Zaehler, die
  - ein Attribut aktueller_stand enthält
  - eine Methode increment() besitzt, die den aktuellen Stand um 1 erhöht
  - eine Methode decrement() besitzt, die den aktuellen Stand um 1 verringert und
  - eine Methode ausgabe() besitzt, die den aktuellen Stand ausgibt.
- Erzeugen Sie drei Objekte der Klasse Zaehler z1, z2 und z3
- Rufen Sie die Methoden der Klasse der Objekte in unterschiedlicher Reihenfolge und Häufigkeit auf und lassen Sie sich die Zwischenstände der Zähler ausgeben.

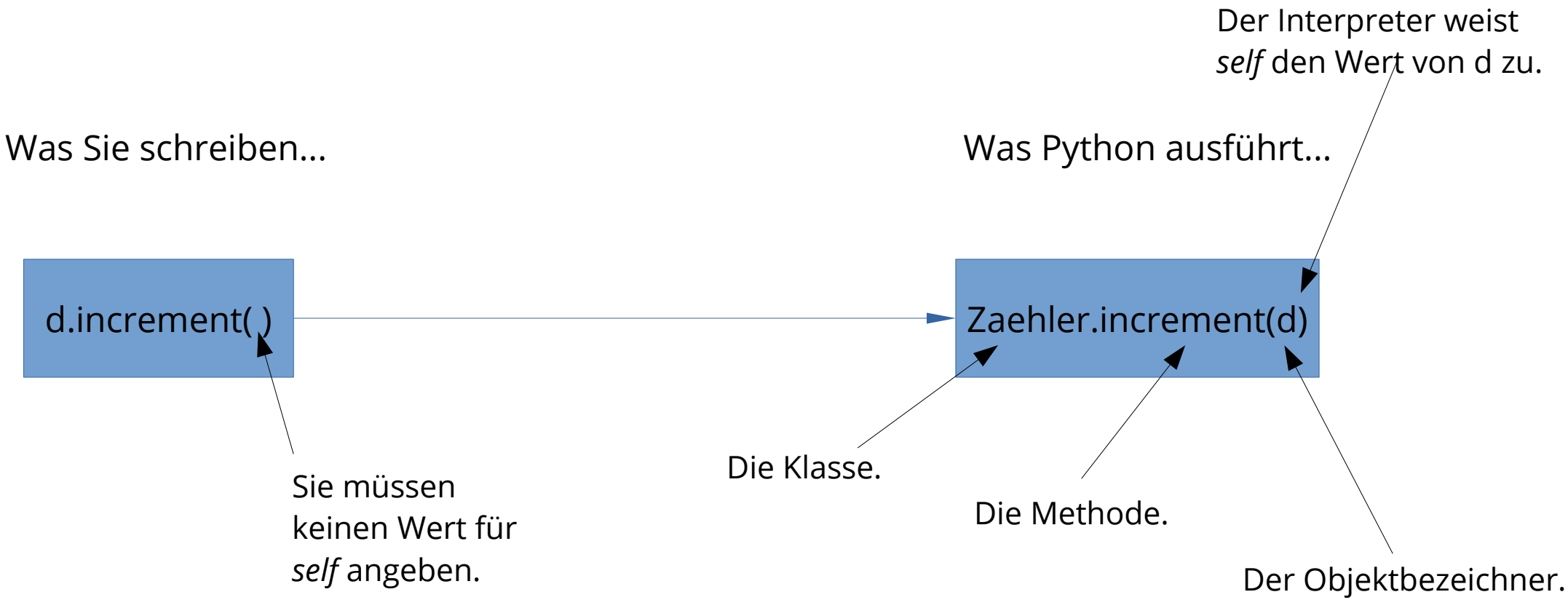
Warum *self* in den Methoden der Klasse übergeben?

```
class Zaehler:
    aktueller_stand = 0

    def increase(self):
        self.aktueller_stand += 1

    def gib_aus(self):
        print(self.aktueller_stand)
```

Zur Erinnerung

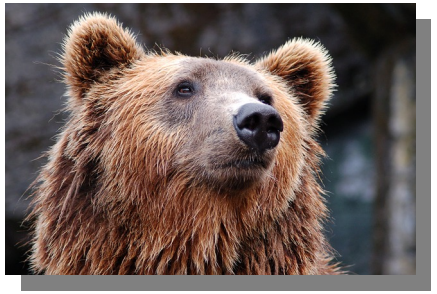
- self muss immer als erster Parameter in den Methoden einer Klasse aufgeführt werden
- Objekte übernehmen das Verhalten (Methoden) aber nicht den Zustand (Attribute) der Klasse

Was Sie sehen, ist nicht das, was passiert...



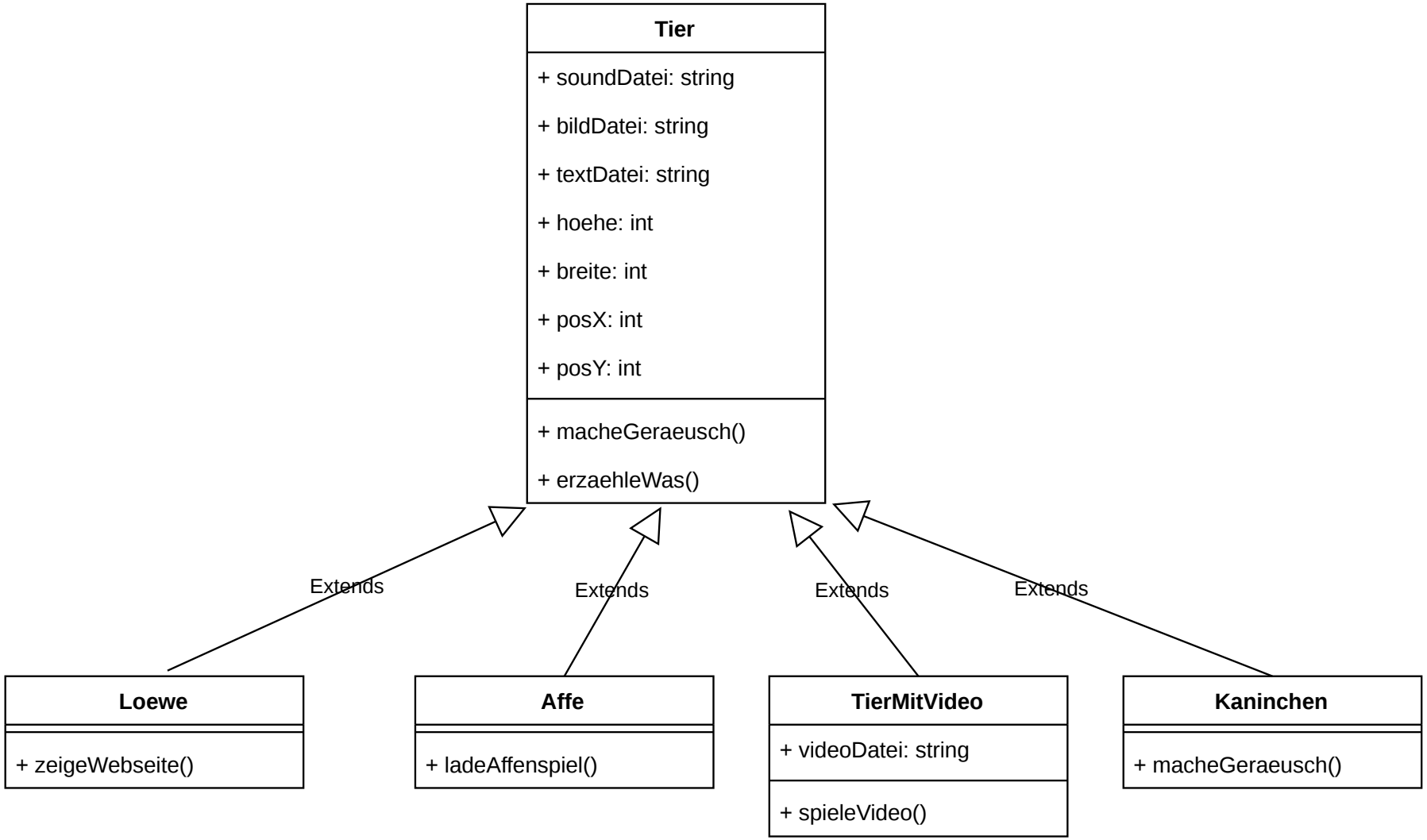
Programme objektorientiert implementieren

Beispiel: Exponat Wilde Tiere

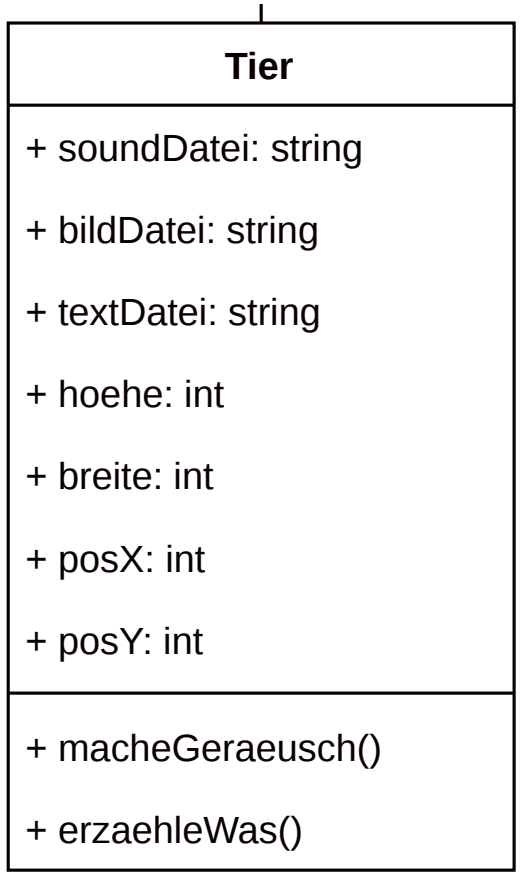


Was soll es tun:
Sie berühren ein Bild. Das Tier brüllt oder es wird Ihnen etwas über das Tier erzählt!

Lösung

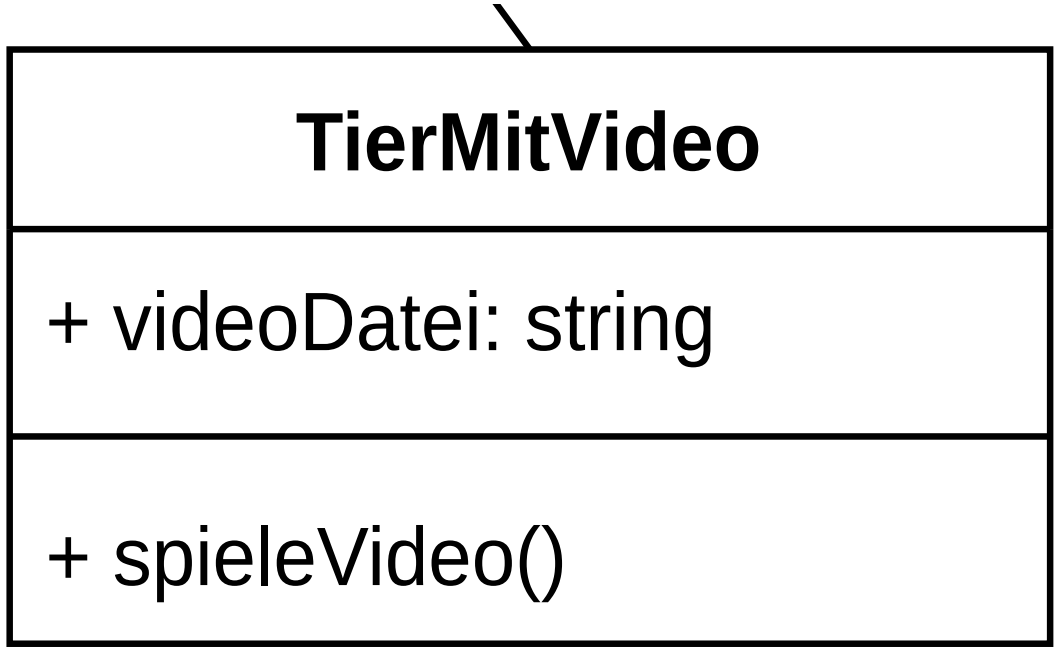

Tier
+ soundDatei: string
+ bildDatei: string
+ textDatei: string
+ hoehe: int
+ breite: int
+ posX: int
+ posY: int
+ macheGeraeusch()
+ erzaehleWas()
Extends
Extends
Extends
Extends
Loewe
+ zeigeWebseite()
Affe
+ ladeAffenspiel()
TierMitVideo
+ videoDatei: string
+ spieleVideo()
Kaninchen
+ macheGeraeusch()

Blick in den Code – Klasse Tier




```python
class Tier:
    soundDatei = ''
    bildDatei = ''
    textDatei = ''
    hoehe = 0
    breite = 0
    x = 0
    y = 0

    def macheGerausch(self):
        print('Spiele soundDatei ab')

    def erzaehleWas(self):
        print('Lese textDatei vor.')
```

Blick in den Code – Klasse TierMitVideo




```
class TierMitVideo(Tier):
    videoDatei = ''

    def spieleVideo(self):
        print('Spiele videoDatei ab')
```

Blick in den Code – Objekt Baer1

Baer1:TierMitVideo

soundDatei = 'bearBruellen.mp3'
bildDatei = 'baer1.jpg'
höehe = 400
breite = 500
textDatei = 'textBaer.txt'
videoDatei = 'bear_tanzt.mp4'

```
baer1 = TierMitVideo()
baer1.soundDatei = 'baerBruellen.mp3'
baer1.bildDatei = 'baer1.jpg'
baer1.hoehe = 400
baer1.breite = 500
baer1.textDatei = 'textBaer.txt'
baer1.videoDatei = 'baer_tanzt.mp4'
```

Einschub *__init__* Funktion

- Eine der sogenannten *magischen* Funktionen.
- Wird aufgerufen direkt nachdem das Objekt instanziiert wurde.
- Mit ihr können alle Attribute gesetzt werden.

```
class Zaehler:
    def __init__(self):
        self.aktueller_stand = 0

    def increase(self):
        self.aktueller_stand += 1

    def gib_aus(self):
        print(self.aktueller_stand)
```

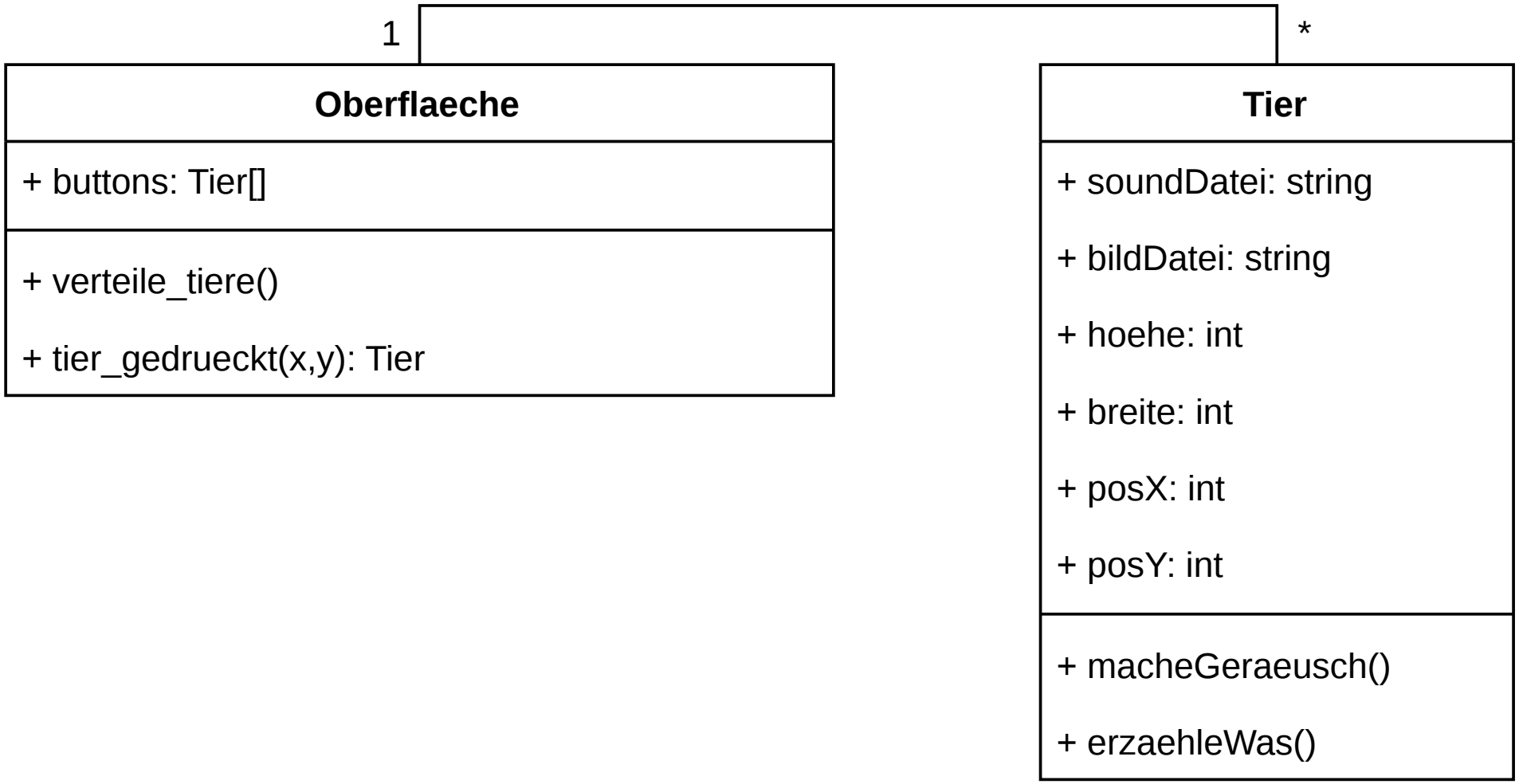
Einschub *__init__* Funktion

```
class Zaehler:
    def __init__(self, zahl=0):
        self.aktueller_stand = zahl

    def increase(self):
        self.aktueller_stand += 1

    def gib_aus(self):
        print(self.aktueller_stand)
```

- Kann durch Parameter erweitert werden.
- Dadurch können beliebige Anfangswerte
  - **gesetzt** und
  - **übergeben**

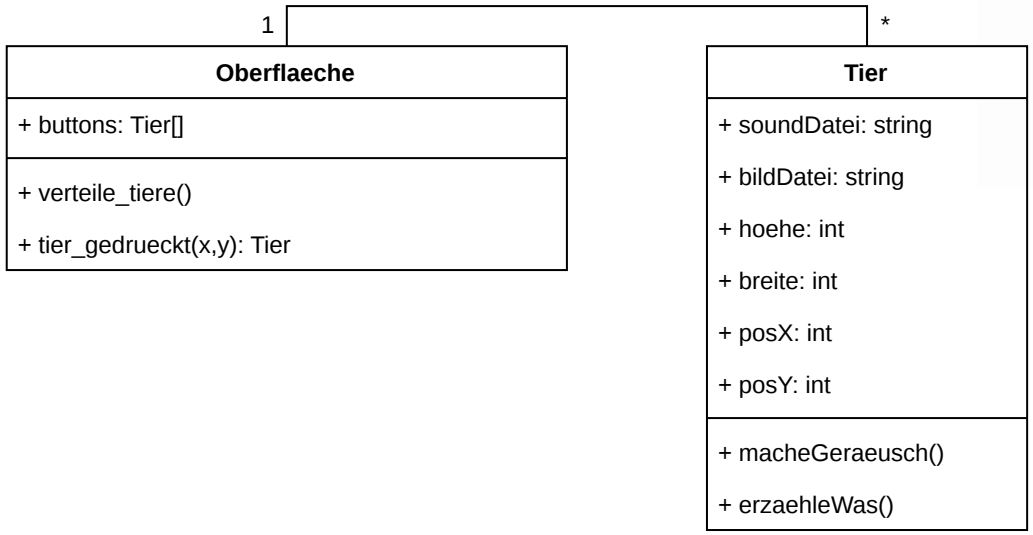
  werden

Assoziation in unserem Beispiel



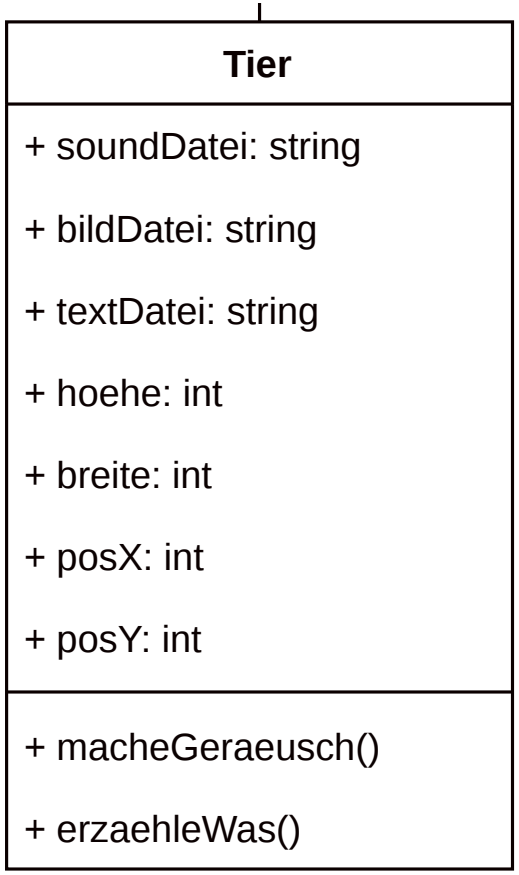
Assoziation in unserem Beispiel – Blick in den Code

Frage: Wie fügen wir jetzt ein neues Tier der Oberfläche hinzu?




```
class Oberflaeche:
    def __init__(self):
        buttons = []

    def verteileTiere(self):
        print('Verteile auf dem Bildschirm.')

    def tier_gedrueckt(self, x, y):
        print('Suche passendes Tier.')
        print('Gebe gefundenes Tier zurück.')
```

Tierklasse mit __init__ Funktion



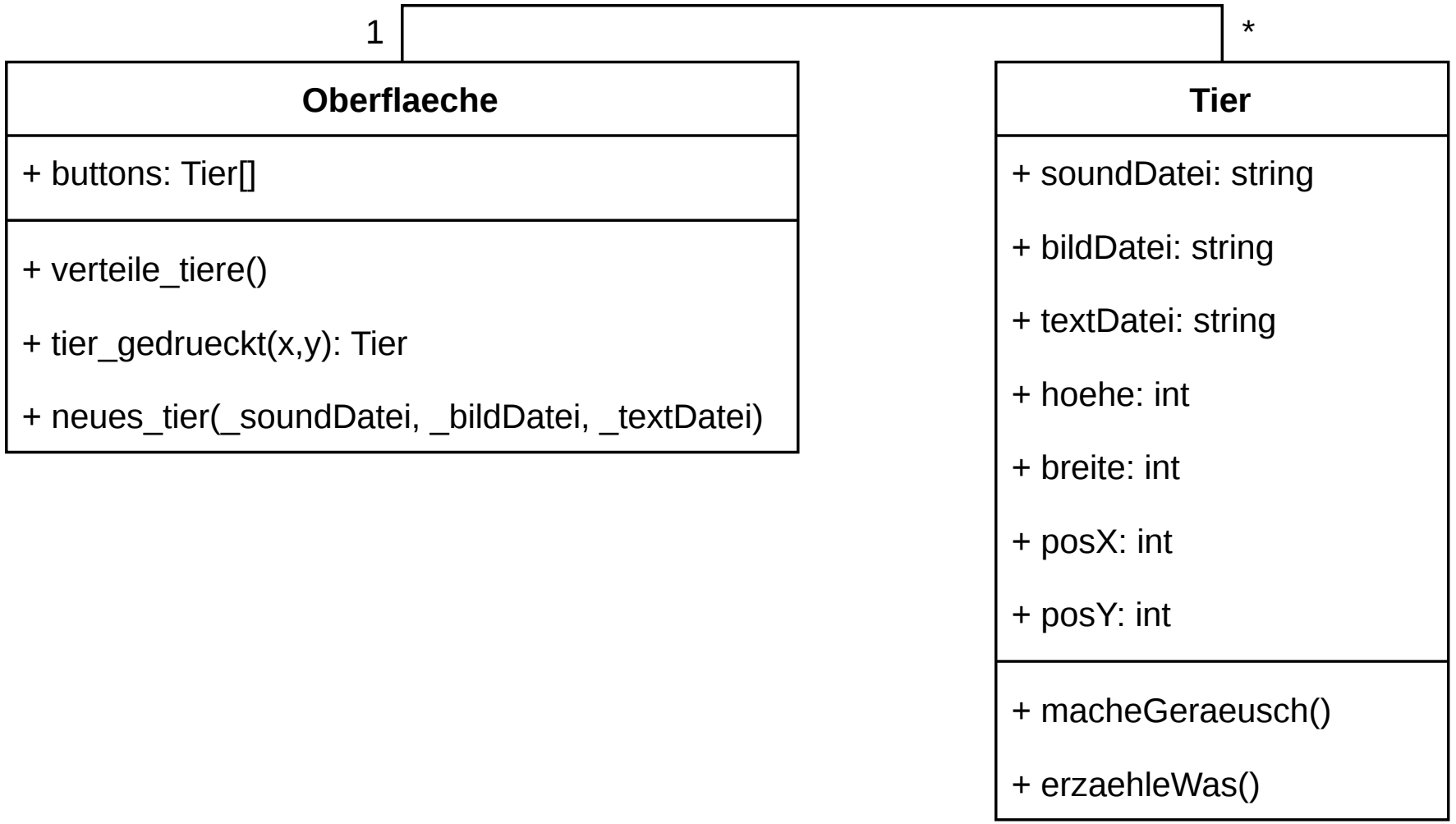

```python
class Tier:
    def __init__(self, _soundDatei, _bildDatei, _textDatei):
        self.soundDatei = _soundDatei
        self.bildDatei = _bildDatei
        self.textDatei = _textDatei
        self.hoehe = 400
        self.breite = 500
        self.x = 0
        self.y = 0

    def macheGerausch(self):
        print('Spiele soundDatei ab')

    def erzaehleWas(self):
        print('Lese textDatei vor.')
```

Assoziation in unserem Beispiel – Blick in den Code

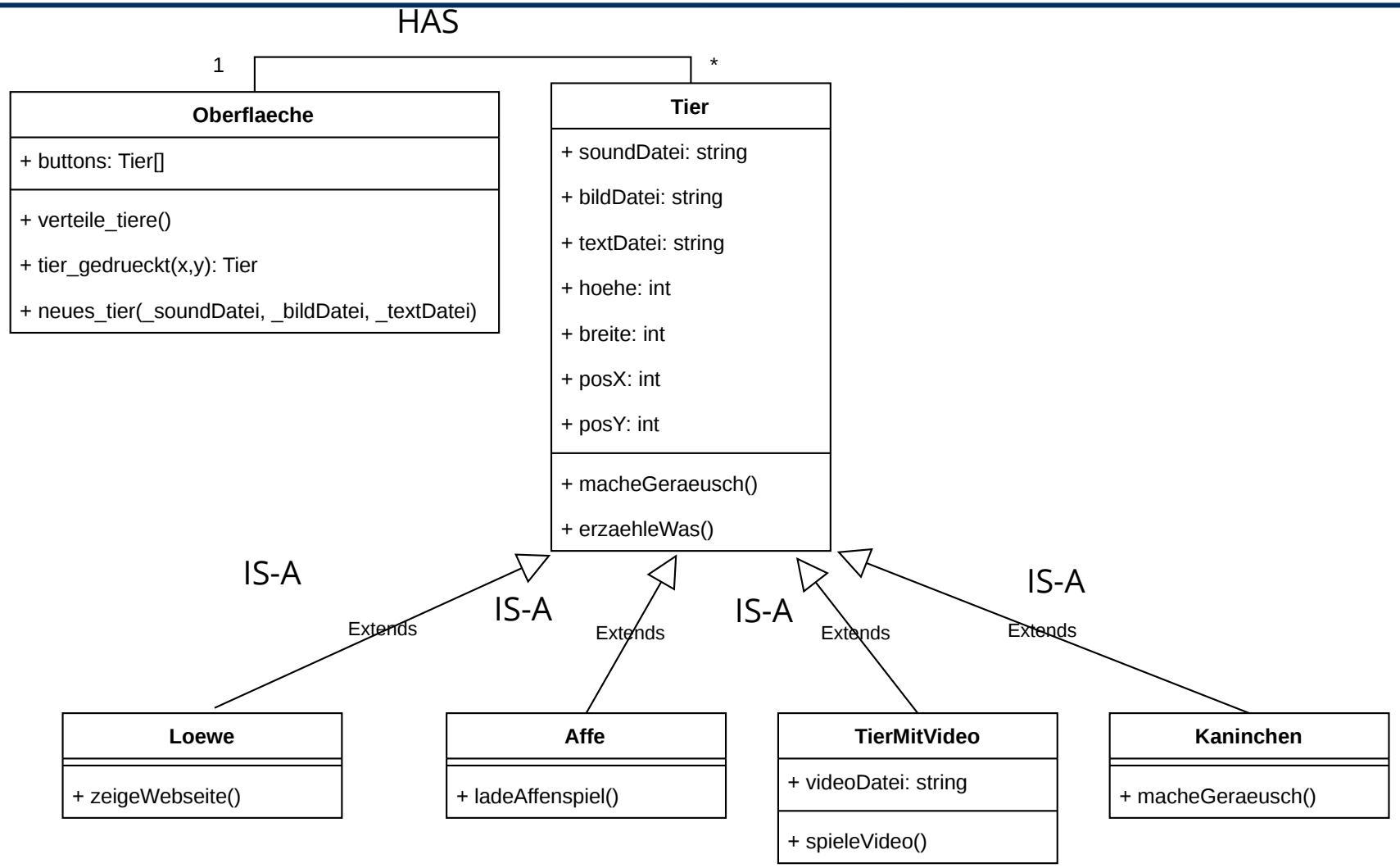
```
class Oberflaeche:
    def __init__(self):
        buttons = []

    def verteileTiere(self):
        print('Verteile auf dem Bildschirm.')

    def tier_gedrueckt(self, x, y):
        print('Suche passendes Tier.')
        print('Gebe gefundenes Tier zurück.')

    def neuesTier(self, _soundDatei, _bildDatei, _textDatei):
        neues_tier = Tier(_soundDatei, _bildDatei, _textDatei)
        self.buttons.append(neues_tier)
```

Oberflaeche
+ buttons: Tier[]
+ verteile_tiere() + tier_gedrueckt(x,y): Tier + neues_tier(_soundDatei, _bildDatei, _textDatei)

Assoziationen in unserem Beispiel



Assoziation in unserem Beispiel



Was Sie mitnehmen und vertiefen sollten…

- Python
  - Klassen definieren
  - Objekte instanziieren
  - Attribute und Methoden hinzufügen
  - Attribute und Methoden vererben
  - Methoden überschreiben und überlagern
  - Auf Methoden des Objekts zugreifen
- Umsetzung in Python
  - Vererbung (IS-A)
  - Assoziation (HAS)
- Magische Funktionen
  - __init__