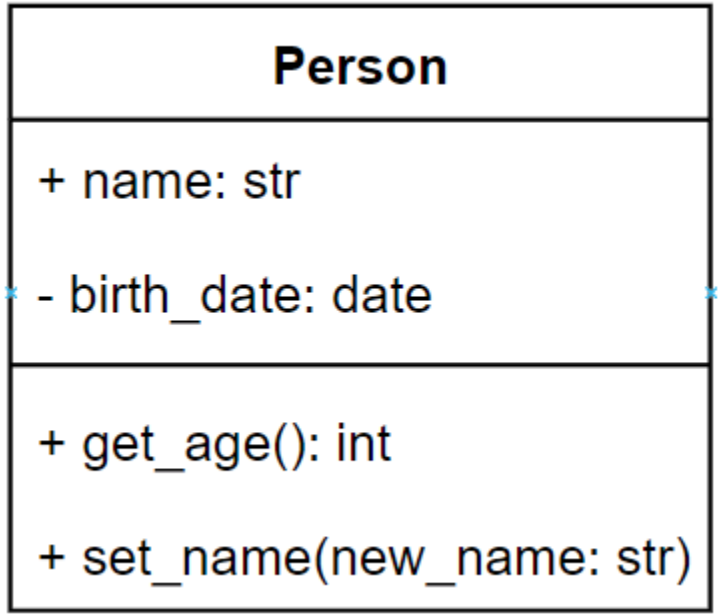
Klassendiagramme Advanced

Wiederholung Klassendiagramm

- Jede Klasse besteht aus 3 Teilen:
  - Klassenname, Attribute, Methoden
- Datentypen für Attribute, Parameter und Rückgabewerte
- Sichtbarkeit einschränken:
  - Public (+)
  - Protected (#)
  - Private (-)



Beziehungen zwischen Klassen

- Klassen können sich gegenseitig referenzieren
- UML unterscheidet zwischen verschiedenen Beziehungsarten
  - Abhängigkeit (existiert aber in IHK-Prüfungen so gut wie verwendet)
  - Assoziation
  - Aggregation
  - Komposition
- Priorität: Abhängigkeit (schwach) -> Komposition (stark)
  - Überschreiben schwächerer Beziehungen
- Darüber hinaus gibt es noch Vererbungsbeziehungen („Hierarchien“)
  - Vererbung bzw. Generalisierung
  - Implementierung

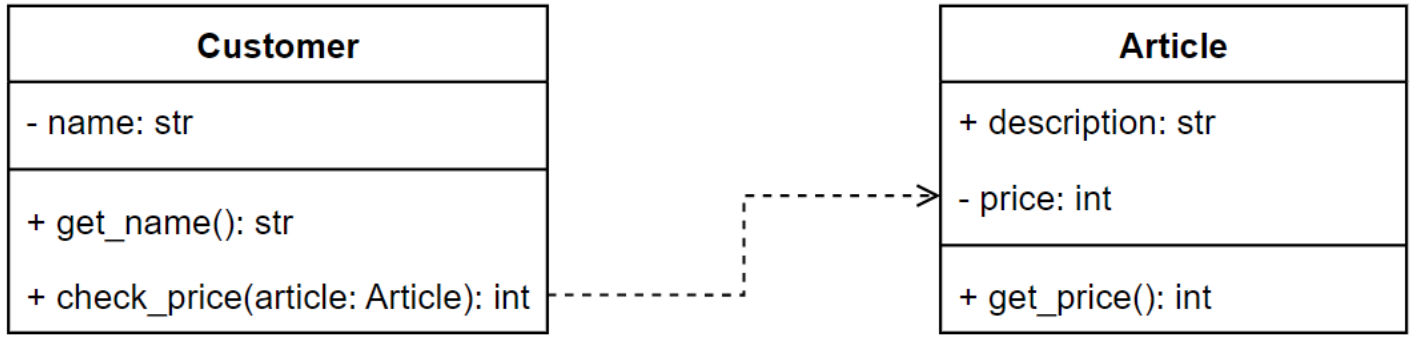
Beispielszenario

- Ihr seid Entwickler eines Onlineshops und habt folgende Klassen mithilfe von UML modelliert:

- Das Klassendiagramm ist noch nicht vollständig und soll erweitert werden

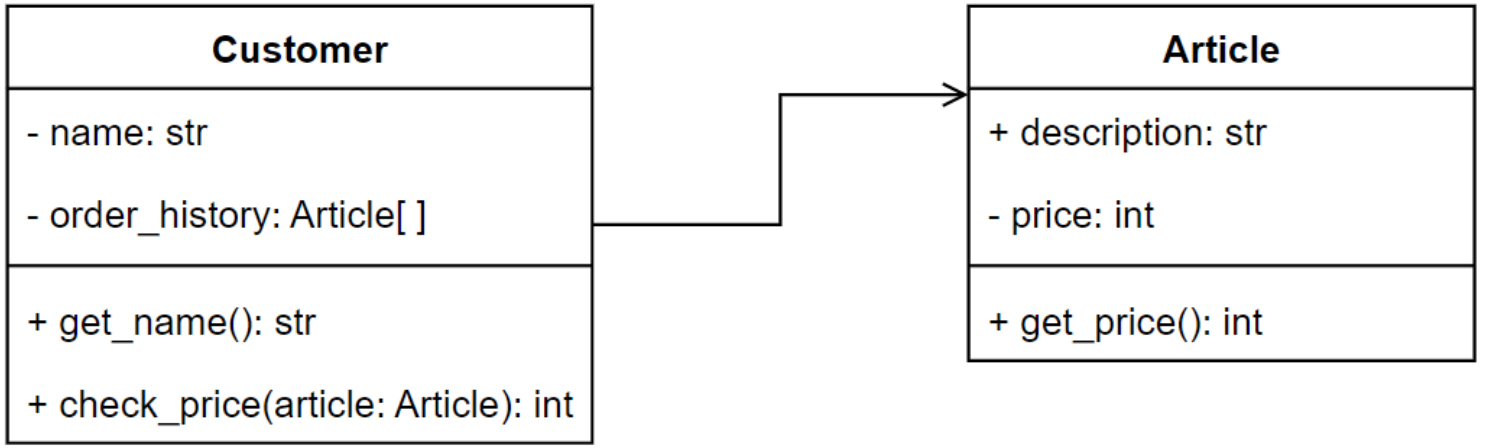
Abhängigkeit

- Anforderung:
  „Der Kunde soll Preise von Artikeln einsehen können.“
- Übersetzung:
  *„Die Klasse Customer erhält eine Methode, deren Parameter ein Objekt der Klasse Article sein **muss**.“*
- Umsetzung in UML:
  - **gestrichelte** Linie mit Pfeil von Customer zu Article
  - Es muss nicht von Methode zu Attribut sein!



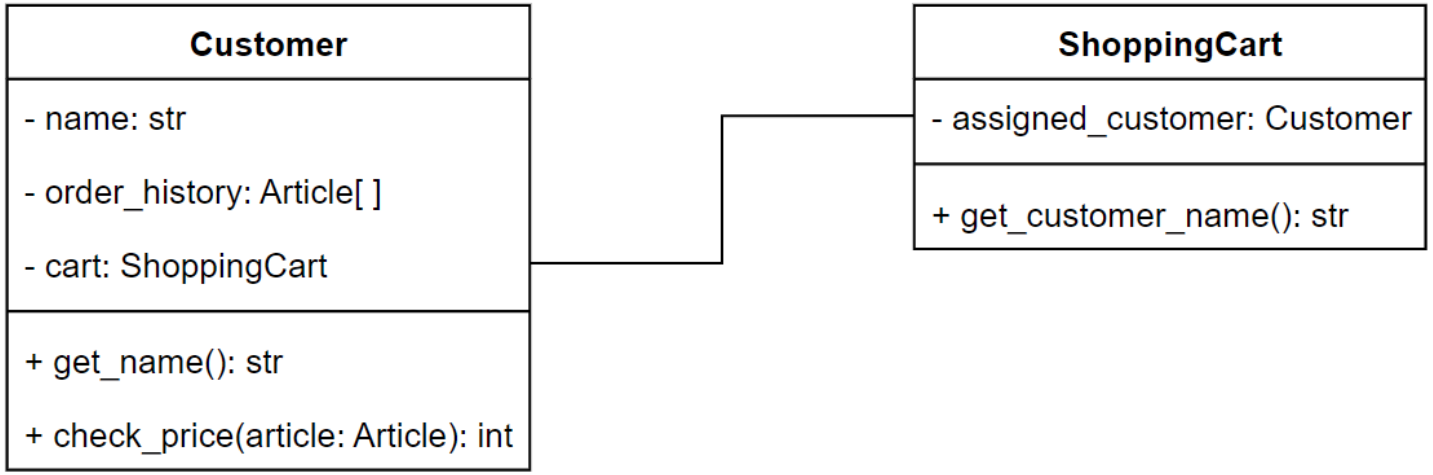
Assoziation

- Anforderung:
  „Zu jedem Kunden wird eine Historie aller bestellten Artikel gespeichert.“
- Übersetzung:
  *„Die Klasse Customer erhält ein Attribut, dessen Inhalt ein Objekt der Klasse Article sein **muss**.“*
- Umsetzung in UML:
  durchgezogene Linie mit Pfeil von Customer zu Article
  Achtung: andere Syntax als in Python!



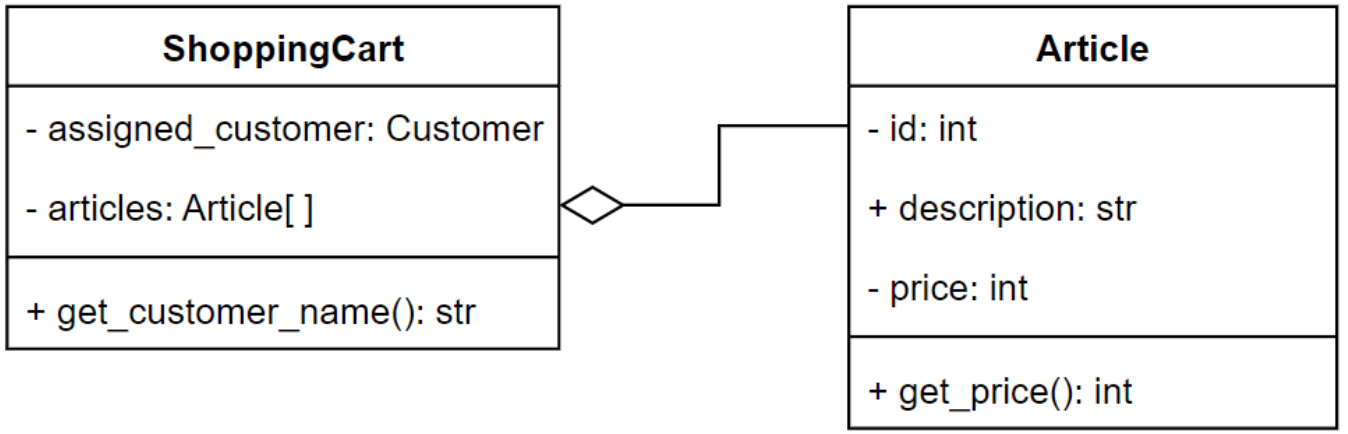
Assoziation (ungerichtet)

- Anforderung:
  „Jeder Warenkorb soll einem bestimmten Kunden zugeordnet sein.“
- Übersetzung:
  *„Die Klasse ShoppingCart erhält ein Attribut, dessen Inhalt ein Objekt der Klasse Customer sein **muss**. Die Klasse Customer erhält ein Attribut, dessen Inhalt ein Objekt der Klasse ShoppingCart sein **muss**.“*
- Umsetzung in UML:
  durchgezogene Linie ohne Pfeil von ShoppingCart zu Customer



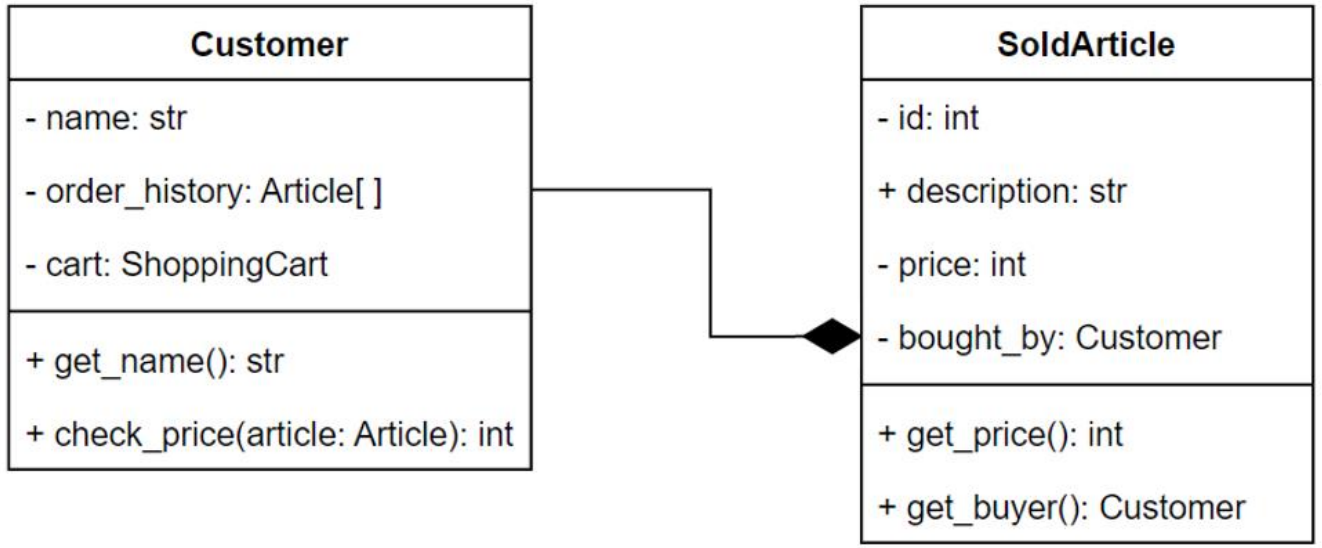
Aggregation

- Anforderung:
  „Jeder Artikel soll einzigartig sein. D.h. keine 2 Warenkörbe sollen den gleichen Artikel beinhalten können.“
- Übersetzung:
  *„Die Klasse ShoppingCart erhält ein Attribut, dessen Inhalt ein Objekt der Klasse Article sein **muss**. Über die Speicheradresse oder ein Attribut id könnte man garantieren, dass es einzigartig ist.“*
- Umsetzung in UML:
  Linie mit *unausgefülltem Diamant* von ShoppingCart zu Article
- Ist eine Spezialform der Assoziation

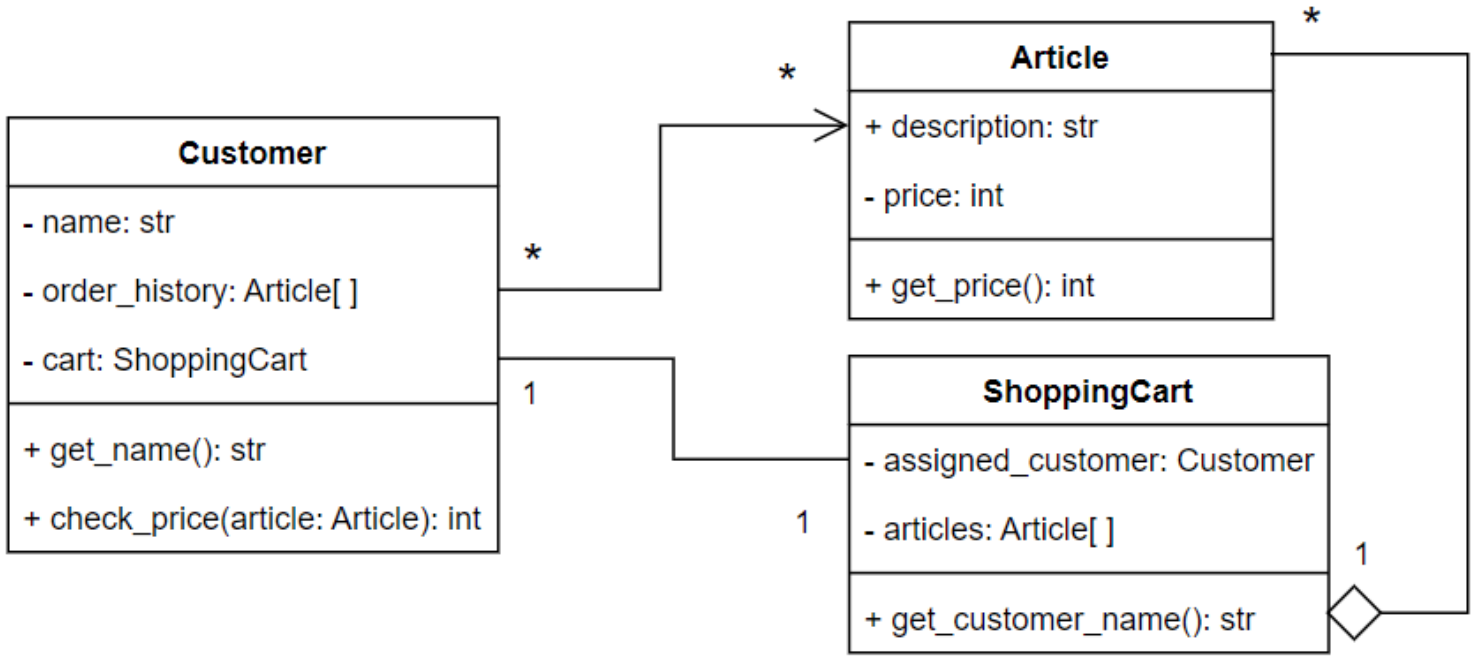


Komposition

- Anforderung:
  „Zu jedem verkauften Artikel soll der Käufer gespeichert werden. Der verkaufte Artikel soll nie wieder verkauft werden können.“
- Übersetzung:
  „Erstelle eine neue Klasse SoldArticle. Diese gleicht Article bis auf ein zusätzliches Attribut der Klasse Customer.“
- Umsetzung in UML:
  Linie mit *ausgefülltem Diamant* von Customer zu SoldArticle
- Ist eine Spezialform der Assoziation
- SoldArticle Objekt kann nur existieren wenn zugehöriger Customer existiert
- Sehr selten



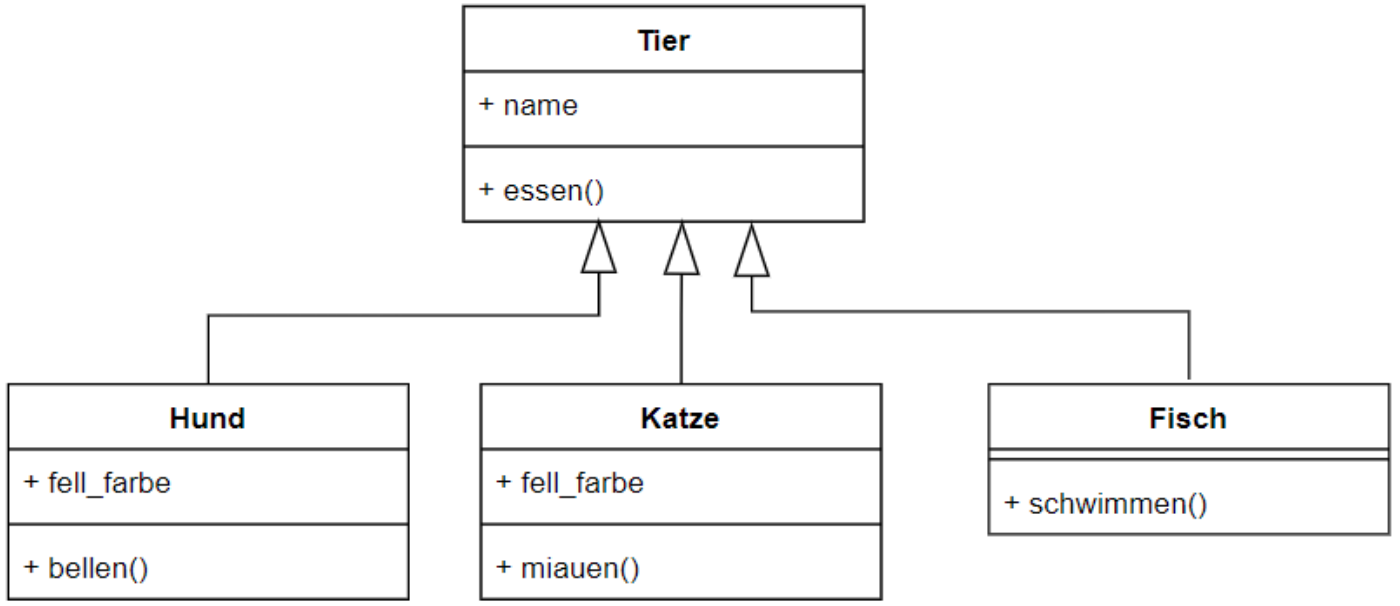
Kardinalität



- Angabe der numerischen Beziehung
- „Wieviel von X steht in Beziehung zu wieviel von Y“
- * = beliebig viele (0, 1, 2, …, ∞)
- Customer * --> * Article
  „Ein Kunde kann verschiedene Artikel in der order_history speichern.“
  „Ein Artikel kann in der order_history verschiedener Kunden sein.“
- Customer 1 --- 1 ShoppingCart
  „Ein Kunde hat genau einen Warenkorb. Ein Warenkorb gehört genau einem Kunden.“
- ShoppingCart 1 <>-- * Article
  „In einem Warenkorb können verschiedene Artikel sein. Ein Artikel lässt sich in genau einem Warenkorb wiederfinden (-> Aggregation).“
- Auch möglich: Angabe eines Minimum und Maximum:
  1..2 -> mind. 1 höchstens 2
  4..* -> mind. 4

Vererbung allgemein

- Idee: Modellierung einer Hierarchie (speziell: Taxonomie)
- Beispiel: Tierwelt
- Aus dem Diagramm lässt sich schlussfolgern:
  - Jeder Hund ist ein Tier
  - Jede Katze ist ein Tier
  - Jeder Fisch ist ein Tier
  - Jedes Tier hat einen Namen
    - Somit auch Hund, Katze, Fisch
  - Jedes Tier kann essen
  - **Nur** Hunde & Katzen haben ein Fell
  - **Nur** Hunde können bellen
  - **Nur** Katzen können miauen
  - **Nur** Fische können schwimmen



Vererbung in Python

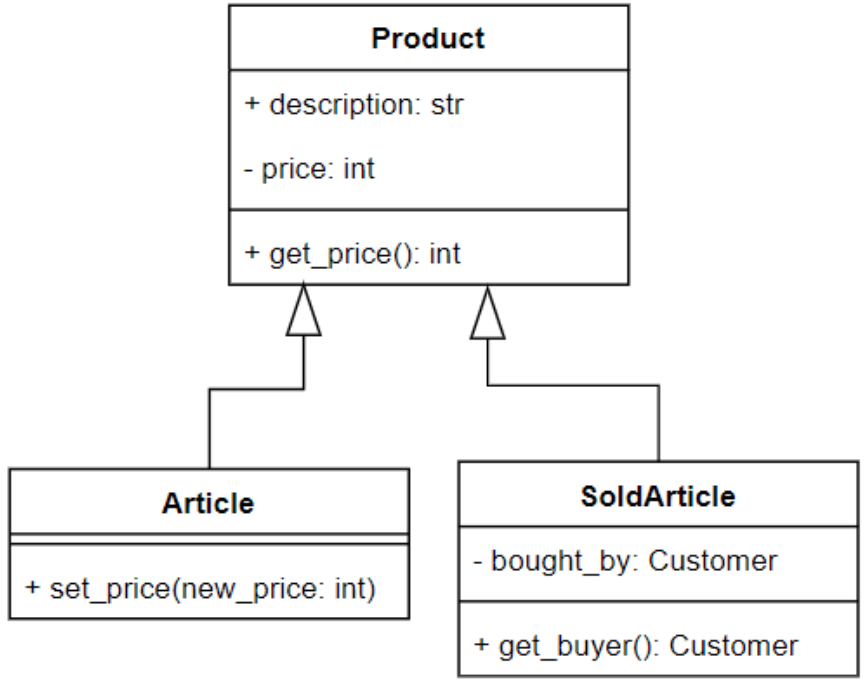
- Kennzeichnen einer Vererbung:
  Neben dem Klassennamen in Klammern
- Alle Methoden und Attribute der Elternklasse werden an die Kindklasse weitergegeben
  - Auch der Konstruktor!
- Konstruktor überschreiben:
  super() Funktion nutzen
  - Referenz zur Elternklasse
- Tipp: Fehlermeldungen sind auch Klassen

```
class MyError(Exception):
    pass

raise MyError()
```

```python
class Animal:

    def __init__(self, name: str):
        self.name = name

    def eat(self):
        print(self.name, "is eating...")


class Dog(Animal):

    def __init__(self, name, fur_color):
        super().__init__(name)
        self.fur_color = fur_color

    def bark(self):
        print("woof")


class Cat(Animal):

    def __init__(self, name, fur_color):
        super().__init__(name)
        self.fur_color = fur_color

    def meow(self):
        print("miau")


class Fish(Animal):

    def swim(self):
        print(self.name, "is swimming...")
```

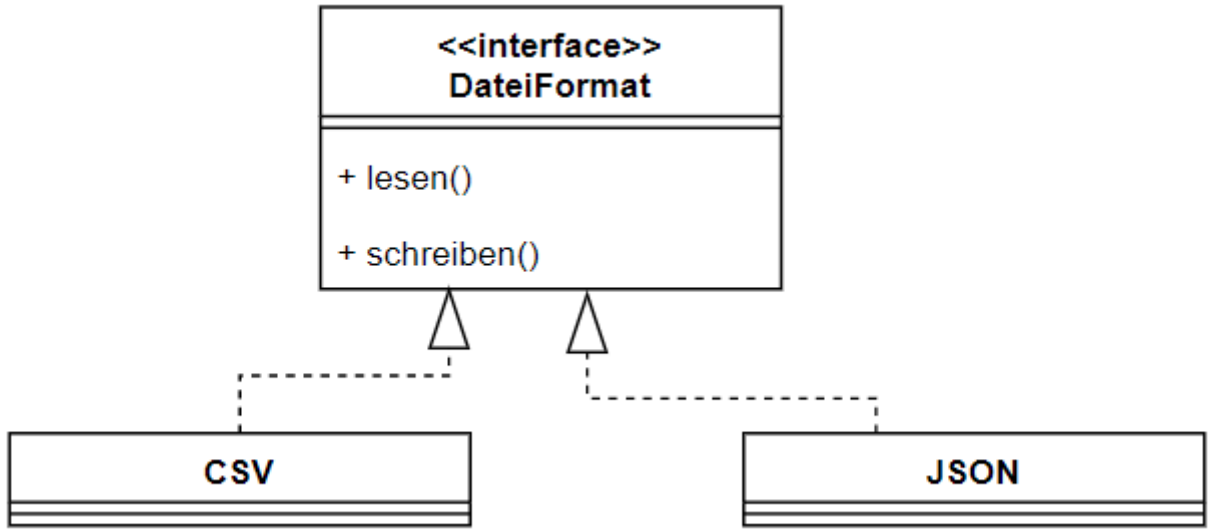
Vererbung am Beispiel

- Anforderung:
  „Es soll unterschieden werden zwischen Artikeln im Sortiment und Artikeln, die bereits verkauft wurden.“
- Übersetzung:
  „Betrachte Gemeinsamkeiten zwischen Artikeln im Sortiment und verkauften Artikeln. Fasse die in einer Klasse zusammen und bilde eine Hierarchie.“
- Umsetzung in UML:
  Linie mit *unausgefülltem Pfeil*
  von erbenden Klasse zur
  vererbenden Klasse
  (= Kinder zu Eltern)
- Alle Attribute und Methoden
  von Product werden vererbt
  (dürfen nicht in den Kindklassen
  angegeben werden!)



Implementierung (Realization)

- Nicht verwechseln mit Implementierung im Sinne von Programmierung
- Neue Klassenart: <<Interface>>
  - In UML, nicht in Python (zumindest nicht direkt)
  - Klasse, die nicht implementierte Methoden beinhaltet
  - Die in Beziehung stehenden Klassen sollen diese Methoden implementieren
- Umsetzung in UML:
  Gestrichelte Linie mit Pfeil
- Aus dem Beispiel folgt:
  Erst in den Klassen CSV & JSON
  werden die Methoden lesen() und
  schreiben() richtig definiert



Hausaufgaben und weitere Übungen

- https://files.ifi.uzh.ch/rerg/amadeus/teaching/courses/inf_oek_2_hs10/uebungen/uebung_4.pdf
  - Frage 1.4 (Seite 3)
  - Aufgabe 2.1 (Seite 4)
- https://docplayer.org/47600217-Uebungsaufgaben-softwaretechnologie.html
  - Aufgabe 1
- Tipp: Für weitere Aufgaben (es gibt sehr viele verschiedene):
  - https://www.google.com/search?q=uml+klassendiagramm+aufgaben