



# Vererbung



## Kapitel 11: Vererbung

11.1 Motivation und Begriffsdefinitionen

11.2 Vorgehensweise und Implementierung

11.3 Arten von Vererbung

11.4 Konstruktoren

11.5 Abstrakte Klasse

11.6 Verschattung

11.7 Wurzelklasse Object

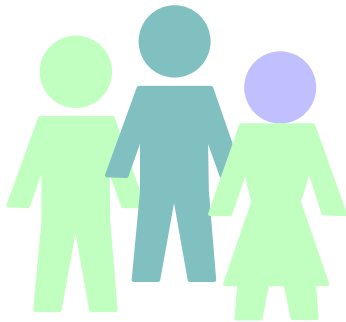
11.8 Zugriffsrechte und Sichtbarkeit

11.9 Schnittstelle

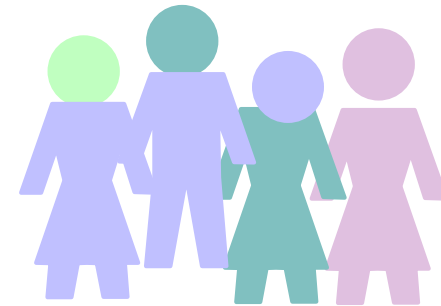


# Menge ähnlicher, aber verschiedener Objekte

## Fußballfans



## Schuhefans



### Eigenschaften

- Name
- Alter

- Lieblingsverein

### Gemeinsam- keiten

### Unterschiede

### Eigenschaften

- Name
- Alter

- Anzahl der Schuhpaare

### Verhaltensweisen

- Schlafen
- Essen

- Fußball schauen

### Gemeinsam- keiten

### Unterschiede

### Verhaltensweisen

- Schlafen
- Essen

- Schuhe kaufen



# Motivation – Analyse auf Metaebene

- Analyse der beiden Gruppen
  - ⊞ Einige Unterschiede  
=> Zusammenfassen in eine Klasse geht nicht!
  - ⊞ Viele Gemeinsamkeiten  
=> Aufspalten in zwei getrennte Klassen bewirkt hohe Redundanz
  
- Wie werden derartige Sachverhalte programmiert?
  - ⊞ Möglichst wenig Redundanz
  - ⊞ Unterschiede deutlich machen



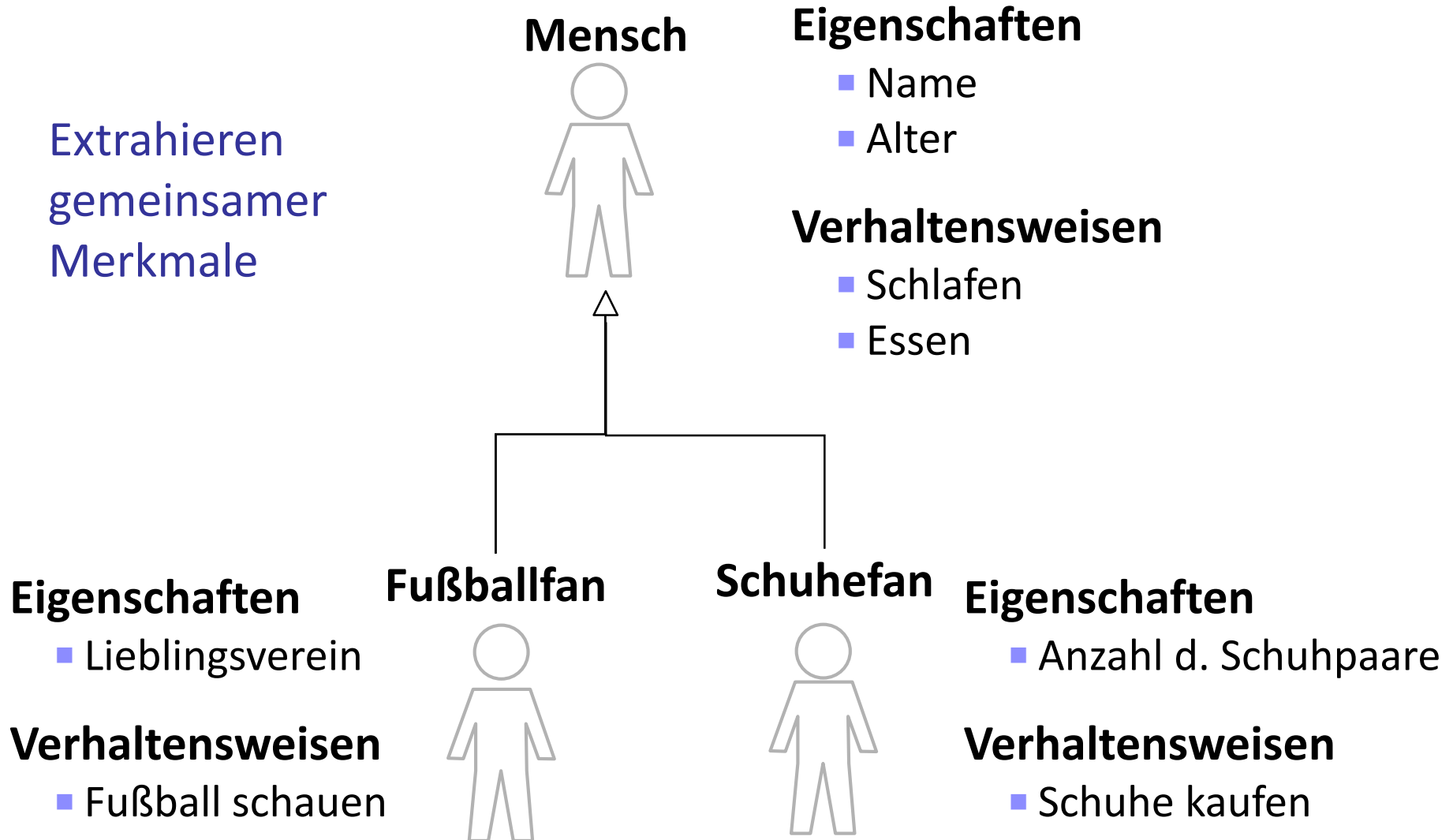
# Motivation - Lösungsidee

## ➤ Lösungsidee

- ⊞ Zentrale Definition der **Gemeinsamkeiten**  
(generalisieren – allgemeine Klasse - Oberklasse)
- ⊞ Spezialisierte Klasse (**Unterklasse**)
  - ⊞ Dokumentation der **Unterschiede**  
(zusätzliche Attribute und/oder Methoden)
  - ⊞ Gemeinsamkeiten geerbt von zentraler Definition  
(Methoden können überschrieben bzw. redefiniert werden)

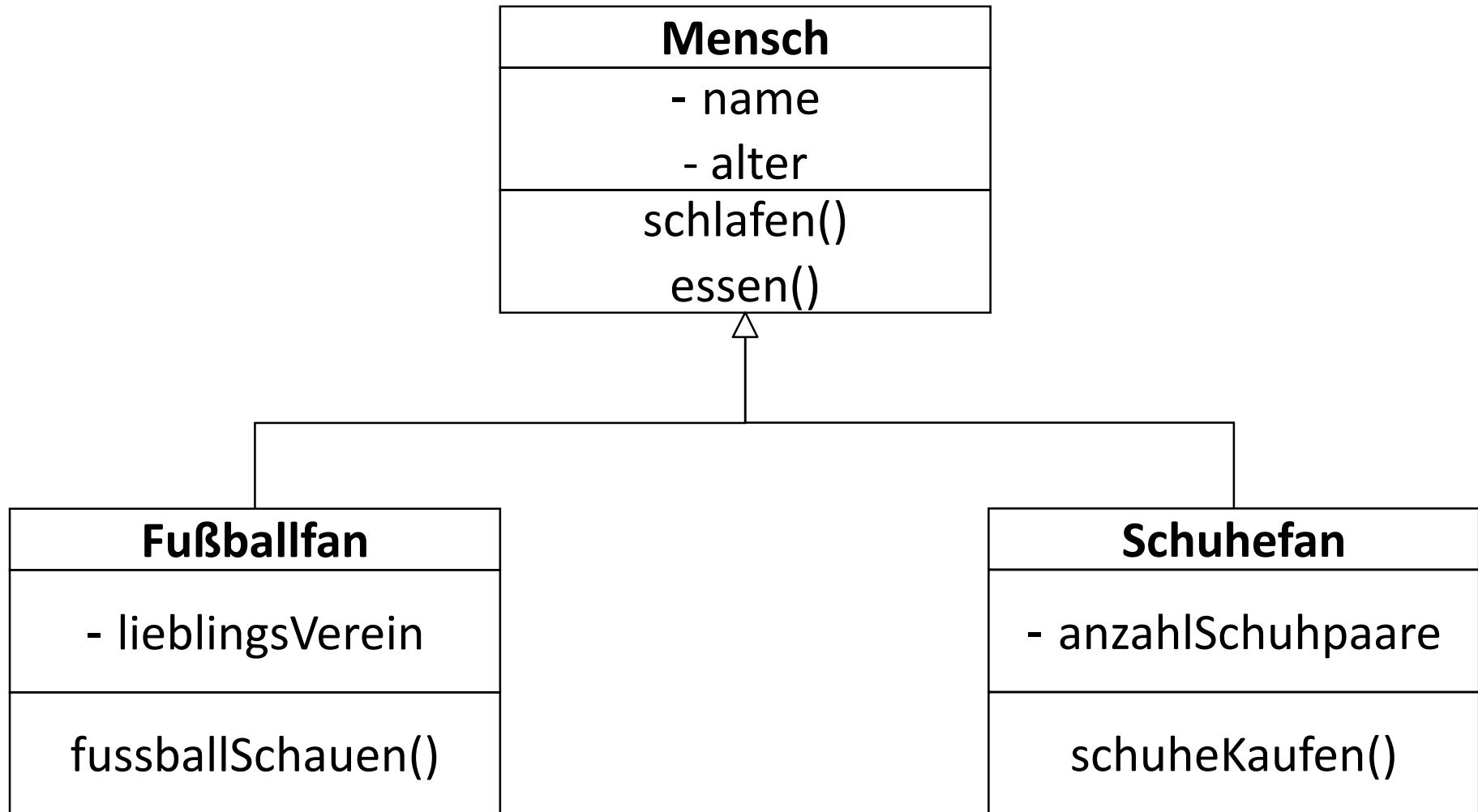


# Beispiel Lösungsidee





# Vererbung im UML-Klassendiagramm





# Bedeutung von Vererbung

## ➤ Grundidee

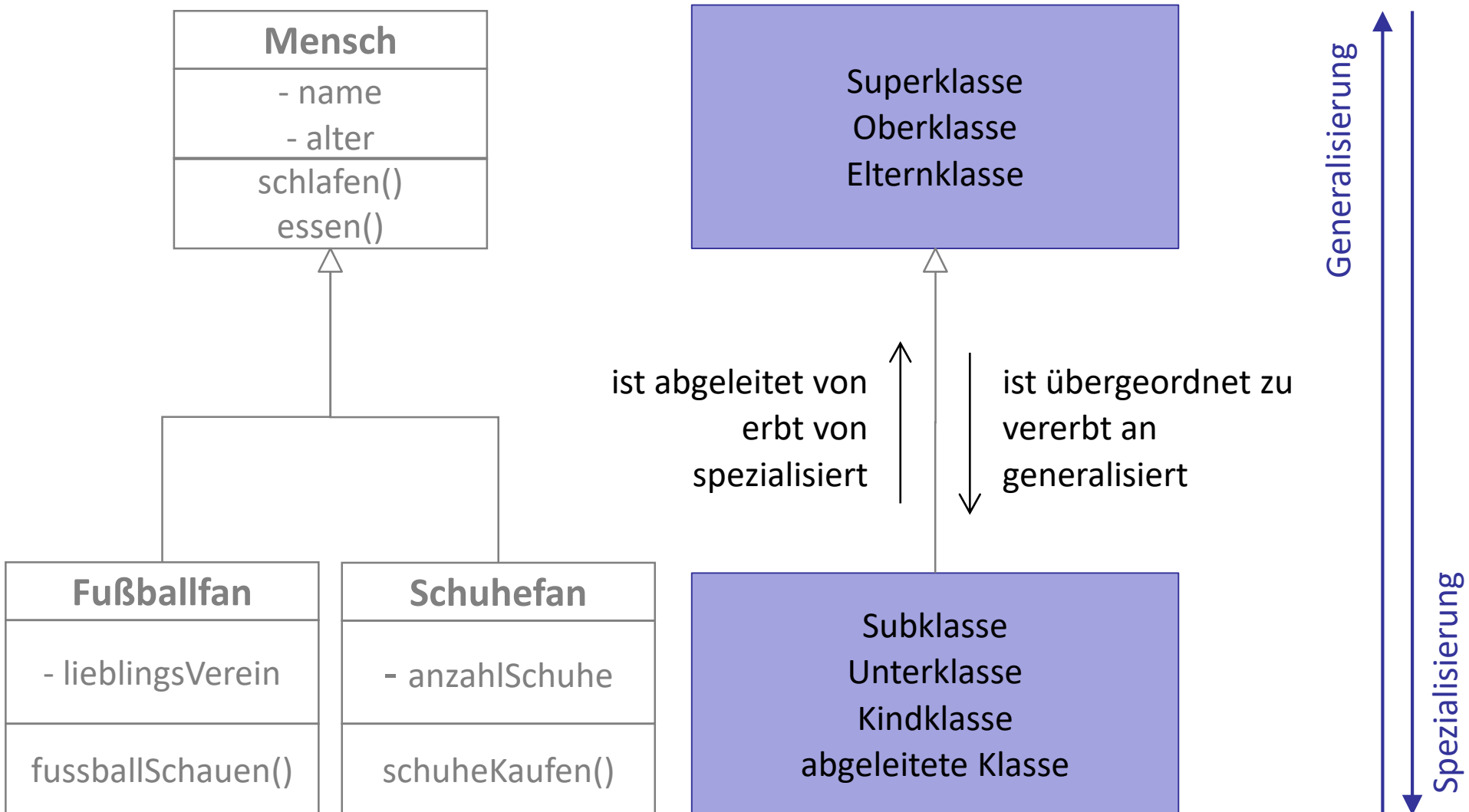
- ⊞ Beschreibt Ähnlichkeit zwischen Klassen
- ⊞ Spezialfall einer Beziehung zwischen Klassen
  - ⊞ Jedes Objekt der Unterklasse „**ist ein**“ (is a) Objekt der Oberklasse
- ⊞ Strukturiert Klassen in Hierarchie von Abstraktionsebenen
- ⊞ Ermöglicht Definition einer neuen Klasse auf Basis bereits bestehender Klassen (Wiederverwendung!)

**Wesentlicher Mechanismus**, der objektorientierte Sprachen von funktionalen/prozeduralen Sprachen unterscheidet!





# Begriffe





## Kapitel 11: Vererbung

11.1 Motivation und Begriffsdefinitionen

11.2 Vorgehensweise und Implementierung

11.3 Arten von Vererbung

11.4 Konstruktoren

11.5 Abstrakte Klasse

11.6 Verschattung

11.7 Wurzelklasse Object

11.8 Zugriffsrechte und Sichtbarkeit

11.9 Schnittstelle



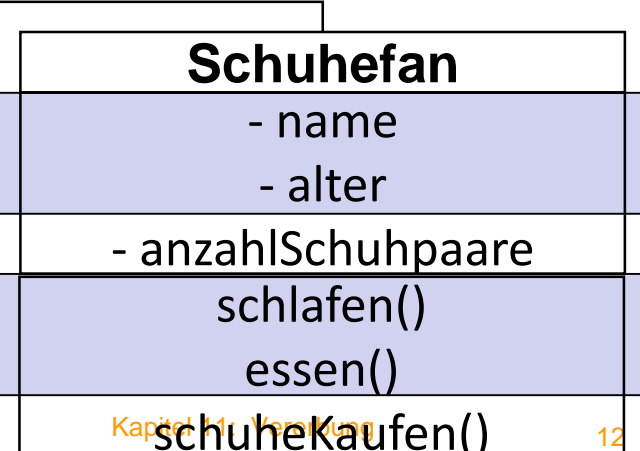
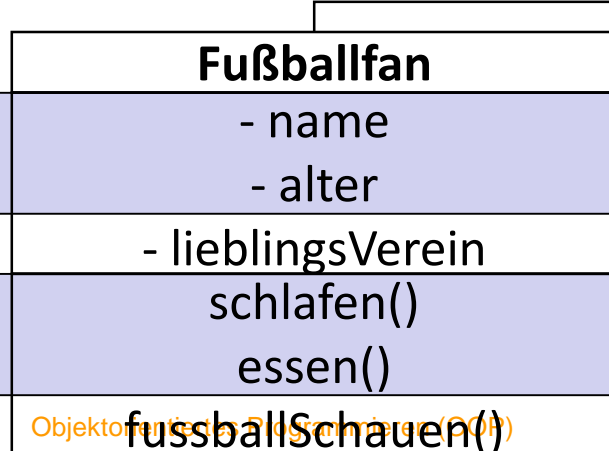
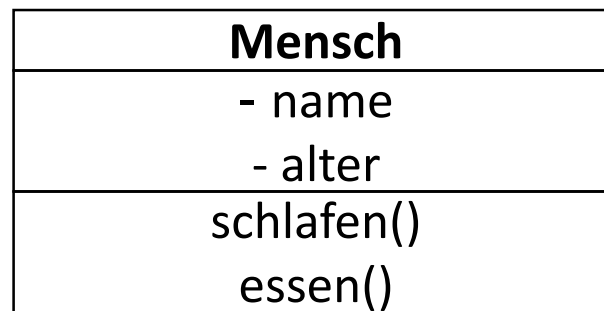
# Vorgehensweise

- Zwei mögliche Vorgehensweisen:
  - ⊞ **Bottom-up:** Vom Speziellen zum Allgemeinen
  - ⊞ **Top-down:** Vom Allgemeinen zum Speziellen
  
- Wann nimmt man was?
  - ⊞ **Bottom-up:**  
Wenn Gemeinsamkeiten erst in teilfertiger Lösung auffallen
  - ⊞ **Top-down:**  
Wenn man schon vorab weiß, dass es Gemeinsamkeiten gibt



# Vorgehensweise – Bottom-up

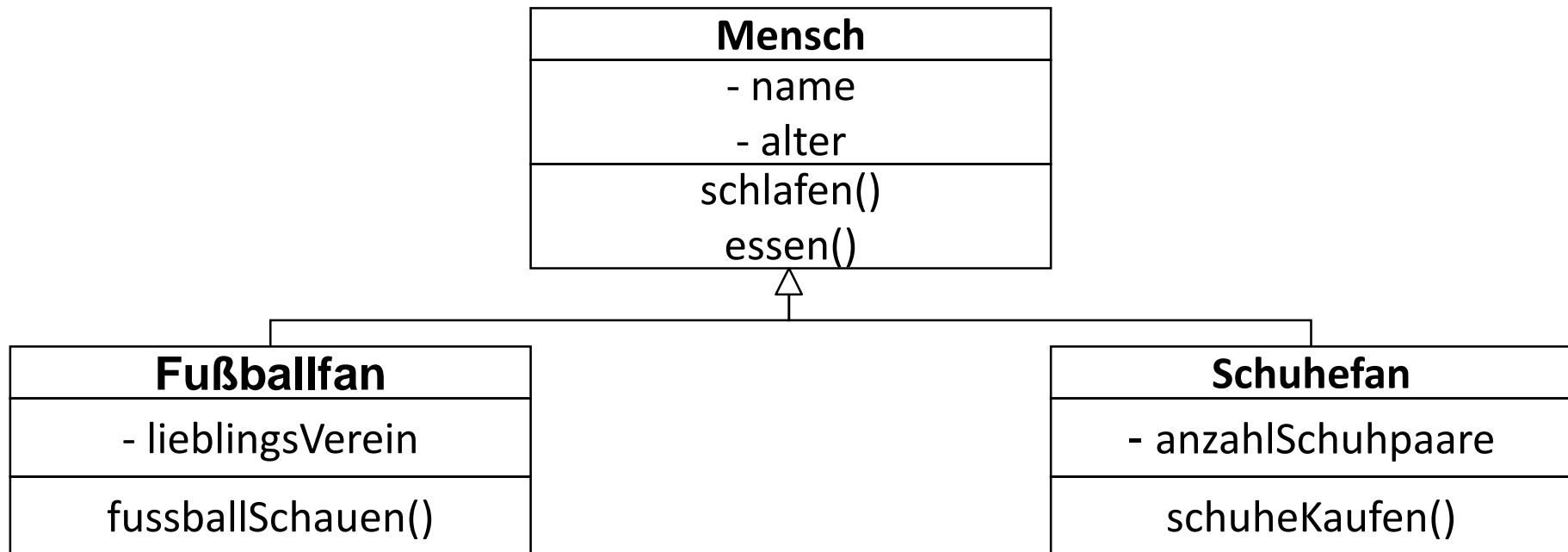
1. Zunächst einzelne Klassen modellieren
2. Redundanzen feststellen
3. Gemeinsamkeiten auslagern in Oberklasse
4. Ursprüngliche Klassen von Oberklasse ableiten und "ausmisten"





## Vorgehensweise – Top-down

1. Erst die Gemeinsamkeiten in zentraler Oberklasse definieren
2. Spezialisierende Klassen definieren, von Oberklasse ableiten
3. Dann die Spezifika der abgeleiteten Klassen definieren
4. Gegebenenfalls Zahl der abgeleiteten Klassen sukzessive erweitern





# Übung – Vererbungsstruktur entwerfen

## ➤ Live Übung

- ✚ Bearbeiten Sie **Aufgabe 1** des Blatts Live Übung „Vererbung“
- ✚ Sie haben 5 Minuten Zeit.





# Vererbung in Java

- **Unterklasse** wird durch das Schlüsselwort `extends` nach dem Klassennamen und gefolgt von dem Namen der Oberklasse spezifiziert
- Jede Klasse kann genau eine Oberklasse besitzen
- Oberklasse weiß nicht welche Unterklassen zu ihr gehören
- Konstruktoren, Methoden oder Attribute können mit dem Schlüsselwort `protected` deklariert werden
  - ⊞ von allen Unterklassen und von allen Klassen innerhalb desselben Pakets kann darauf zugegriffen werden

**Achtung:** Semantik von `protected` ist anders als in z.B. C++



# Sichtbarkeiten im Überblick

Modifizier	Klasse	Paket	Unterklasse	Welt
<code>public</code>	Ja	Ja	Ja	Ja
<code>protected</code>	Ja	Ja	Ja	Nein
<i>kein Attribut</i>	Ja	Ja	Nein	Nein
<code>private</code>	Ja	Nein	Nein	Nein

<http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>

- Attribute in der Regel `private`
  - ⊞ ...außer guter Grund für `protected` oder `public`
- Methoden in der Regel `public`
  - ⊞ ...außer guter Grund für `protected` oder `private`





# Implementierung – Definition der Oberklasse

```
public class Person {  
  
    // Gemeinsame Eigenschaften aller Unterklassen  
    private String name;  
    private int age;  
  
    // Gemeinsame Funktionalität aller Unterklassen  
    public String sleep() {  
        return "sleep: Chrrrrrr.... chrrrrr...";  
    }  
    public String eat() {  
        return "eat : Mmmh, lecker.";  
    }  
}
```



# Implementierung – Unterklasse definieren (1)

```
public class Fussballfan extends Person {  
  
    // Neues Attribut  
    private String favoriteClub;  
  
    // Neue Funktionalität  
    public String watchSoccerGame() {  
        return "play : ja... Ja... TOOOOOOOOR!!!";  
    }  
}
```



## Implementierung – Unterklasse definieren (2)

```
public class Schuhefan extends Person {  
  
    // Neues Attribut  
    private int pairsOfShoes;  
  
    // Neue Funktionalität  
    public String buyShoes() {  
        pairsOfShoes++;  
        return "shop : DIE sind ja schick..., " +  
            "Paar Nummer" + pairsOfShoes;  
    }  
}
```



# Implementierung – Hauptklasse definieren

```
public class Main {  
    public static void processPerson(Person person) {  
        person.eat();  
    }  
    public static void main(String[] args) {  
        Fussballfan ff = new Fussballfan();  
        Schuhefan sf = new Schuhefan();  
        System.out.println("Das macht Fussballfan:");  
        ff.sleep();  
        ff.watchSoccerGame();  
        processPerson(ff);  
        System.out.println();  
        System.out.println("Das macht Schuhefan:");  
        sf.sleep();  
        sf.buyShoes();  
        processPerson(sf);  
        System.out.println();  
    }  
}
```



# Implementierung – Ausgabe

## ➤ Ausgabe des Hauptprogramms

Das macht Fussballfan:

sleep: Chrrrrrr.... chrrrrr...

eat : Mmmmh, lecker.

play : Ja... JAA... T00000000R!!!

Das macht Schuhefan:

sleep: Chrrrrrr.... chrrrrr...

eat : Mmmmh, lecker.

shop : DIE sind ja schick...



# Übung – Vererbung in Java

## ➤ Live Übung

- ✚ Bearbeiten Sie **Aufgabe 2** des Blatts Live Übung „Vererbung“
- ✚ Sie haben 5 Minuten Zeit.





## Kapitel 11: Vererbung

- 11.1 Motivation und Begriffsdefinitionen
- 11.2 Vorgehensweise und Implementierung
- 11.3 Arten von Vererbung
- 11.4 Konstruktoren
- 11.5 Abstrakte Klasse
- 11.6 Verschattung
- 11.7 Wurzelklasse Object
- 11.8 Zugriffsrechte und Sichtbarkeit
- 11.9 Schnittstelle



# Arten von Vererbung

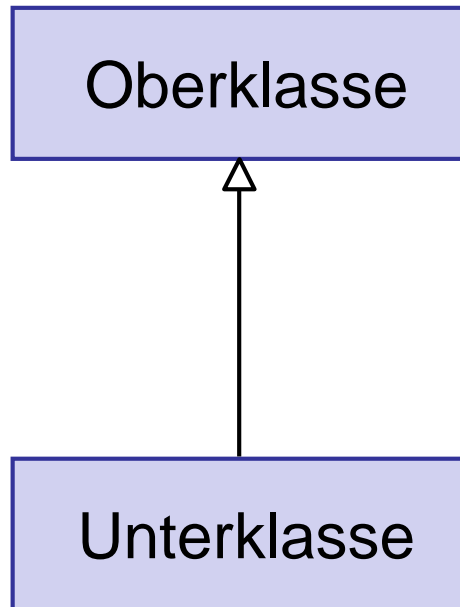
## ➤ Einfachvererbung

- ⊞ Unterklasse erbt von genau einer Oberklasse

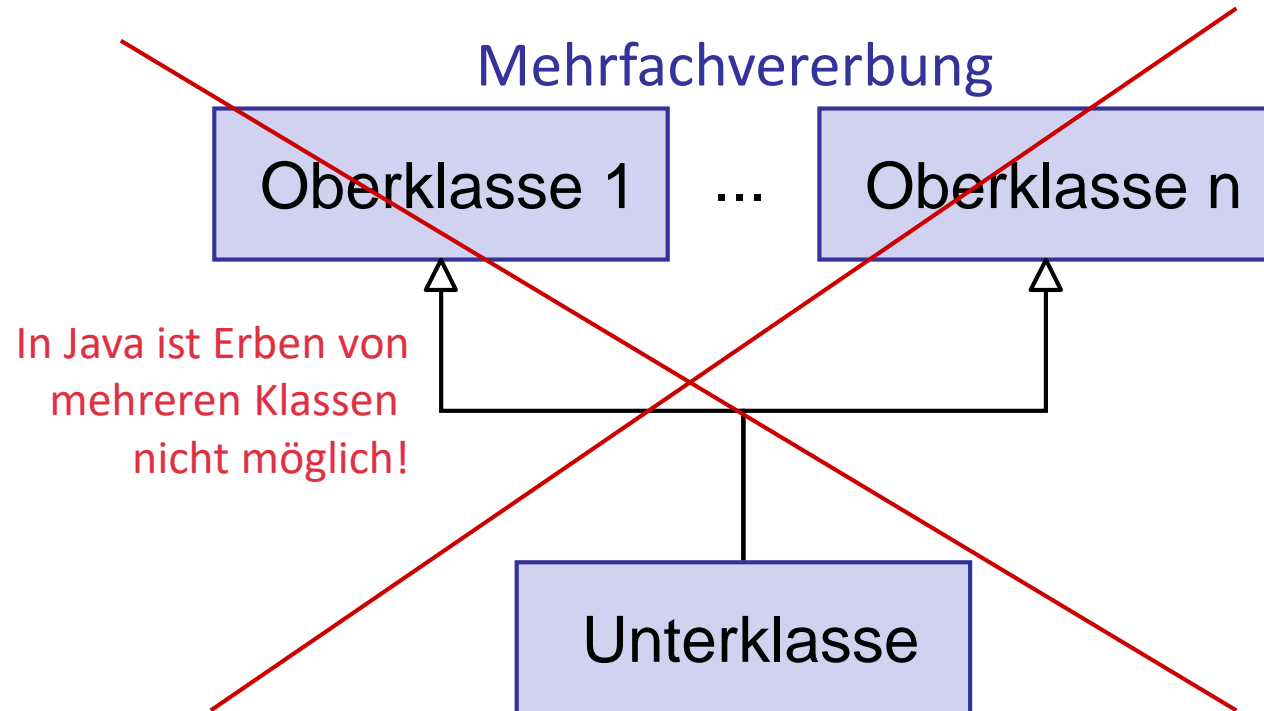
## ➤ Mehrfachvererbung

- ⊞ Unterklasse erbt von mehr als einer Oberklasse

### Einfachvererbung



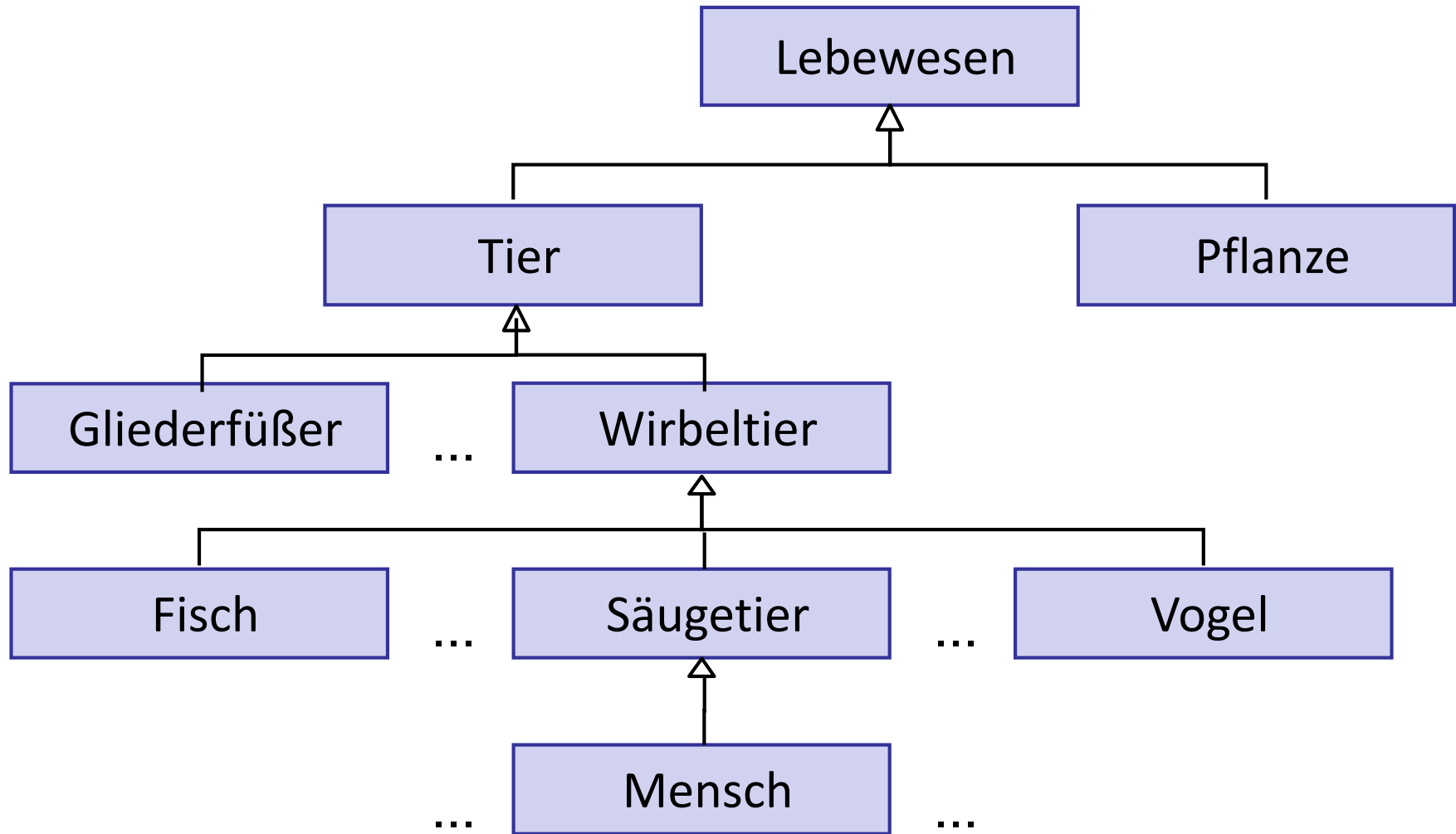
### Mehrfachvererbung





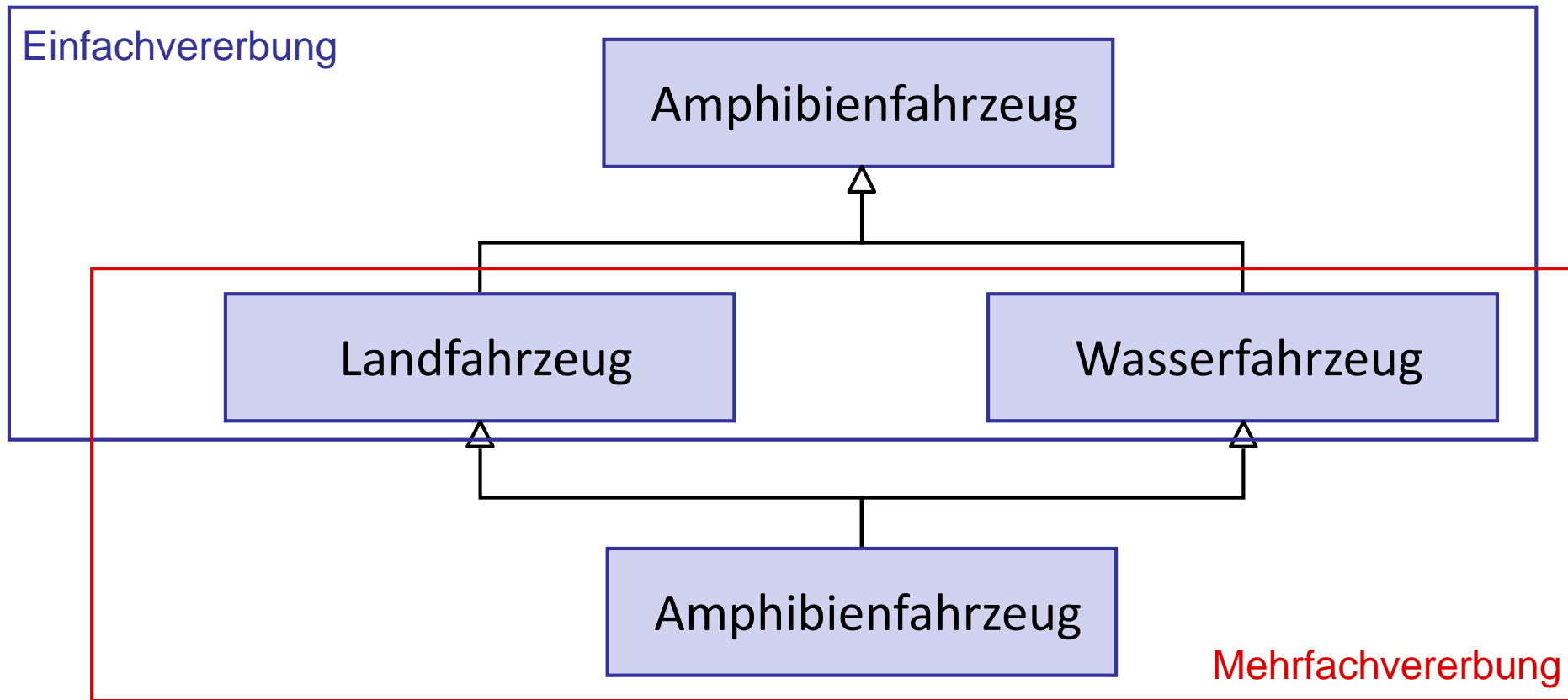


# Einfachvererbung über mehrere Stufen





# Einfach- und Mehrfachvererbung



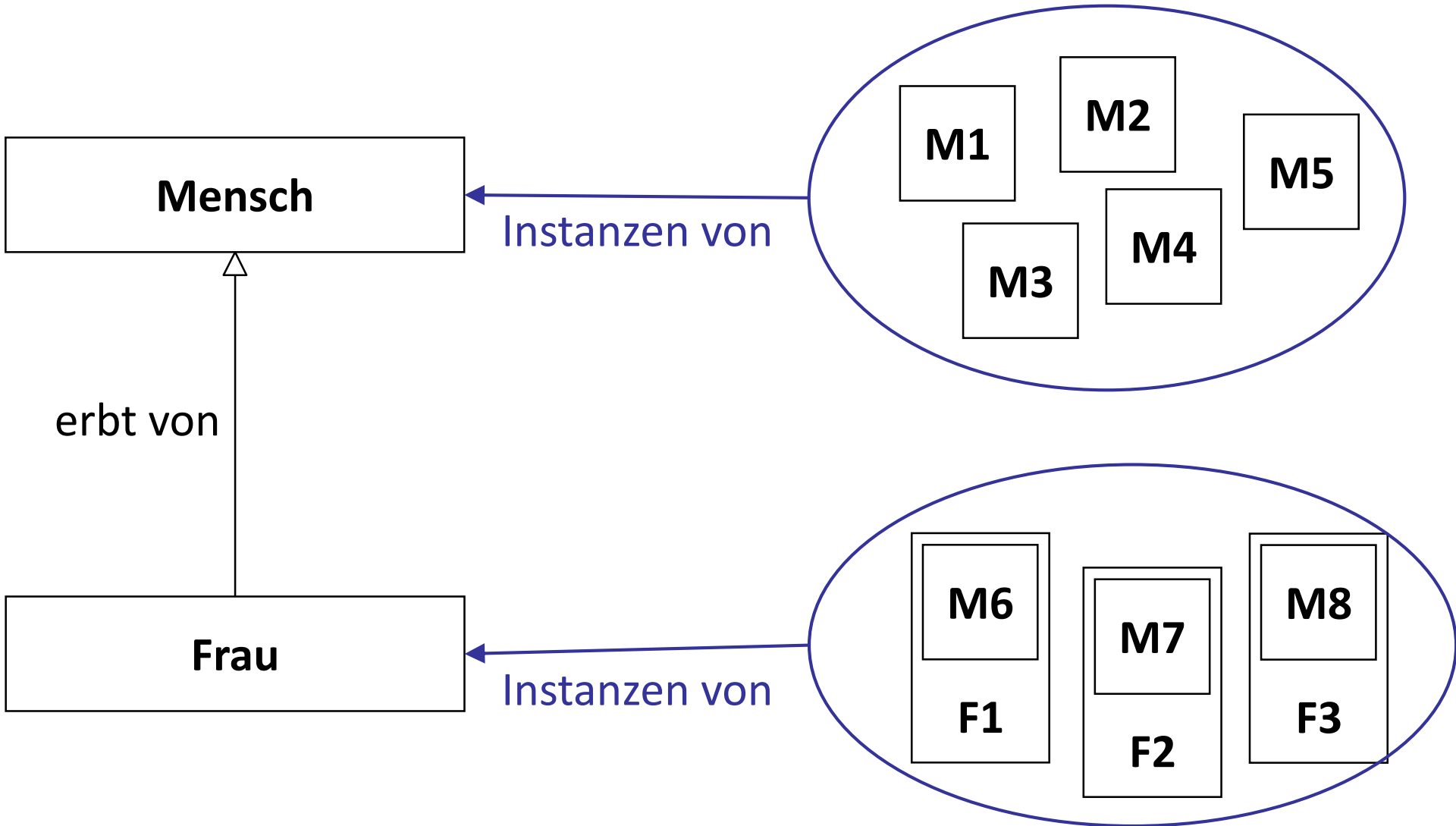


# Was wird vererbt?

- Unterklasse erbt von Oberklasse ...
  - ⊞ die Operationen (das Verhalten)
  - ⊞ die Attribute (die möglichen Zustände)
  - ⊞ die Semantik!  
(d.h. anstelle eines Objekts der Oberklasse kann immer auch ein Objekt einer beliebigen Unterklasse verwendet werden!  
=> **Substitutionsprinzip**)
  
- Beispiele in Java:
  - ⊞ `Person p = new Man();`  
`p = new Woman();`

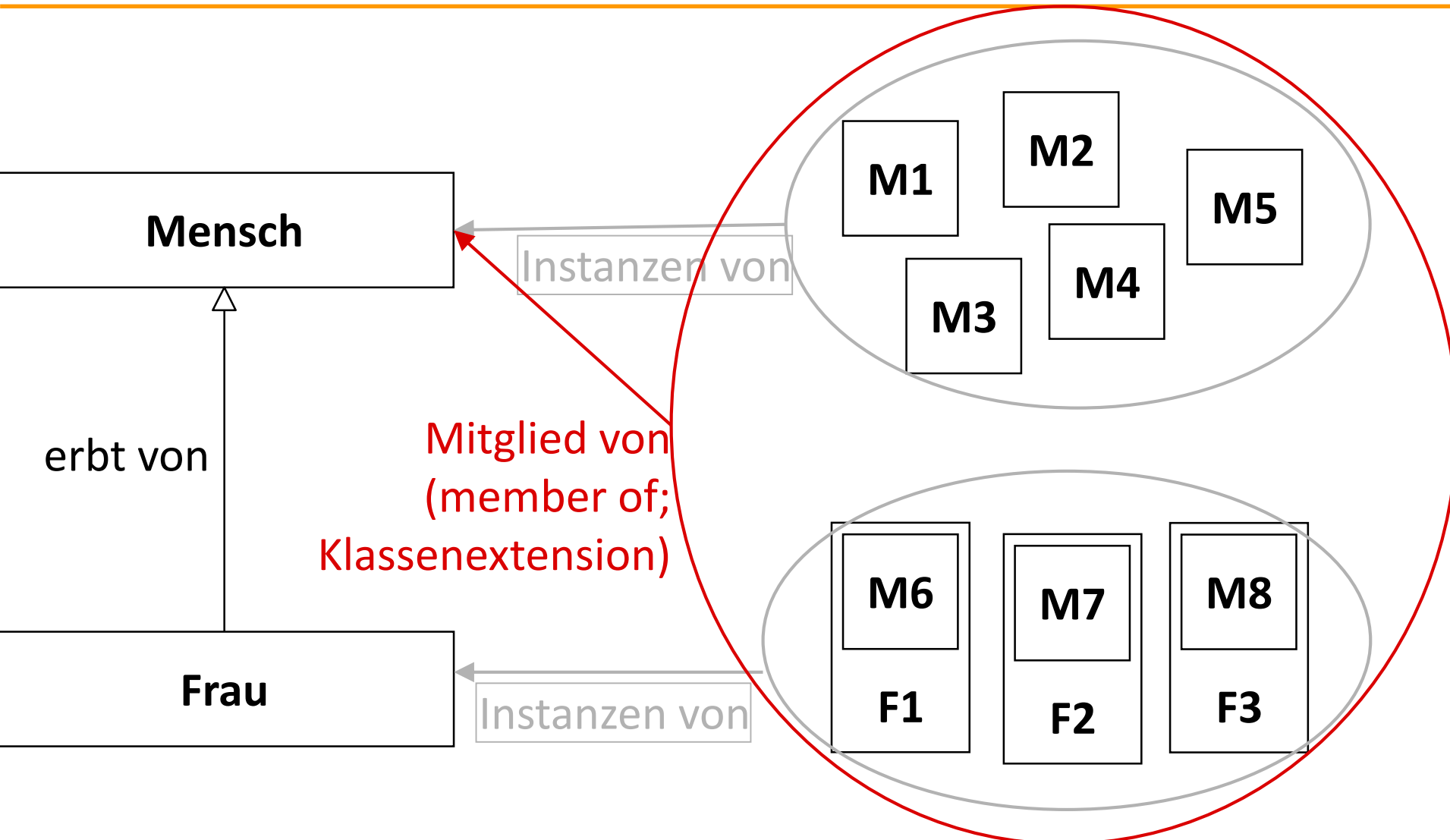


# Syntaktische Vererbung





# Semantische Vererbung





## Kapitel 11: Vererbung

- 11.1 Motivation und Begriffsdefinitionen
- 11.2 Vorgehensweise und Implementierung
- 11.3 Arten von Vererbung
- 11.4 Konstruktoren
- 11.5 Abstrakte Klasse
- 11.6 Verschattung
- 11.7 Wurzelklasse Object
- 11.8 Zugriffsrechte und Sichtbarkeit
- 11.9 Schnittstelle



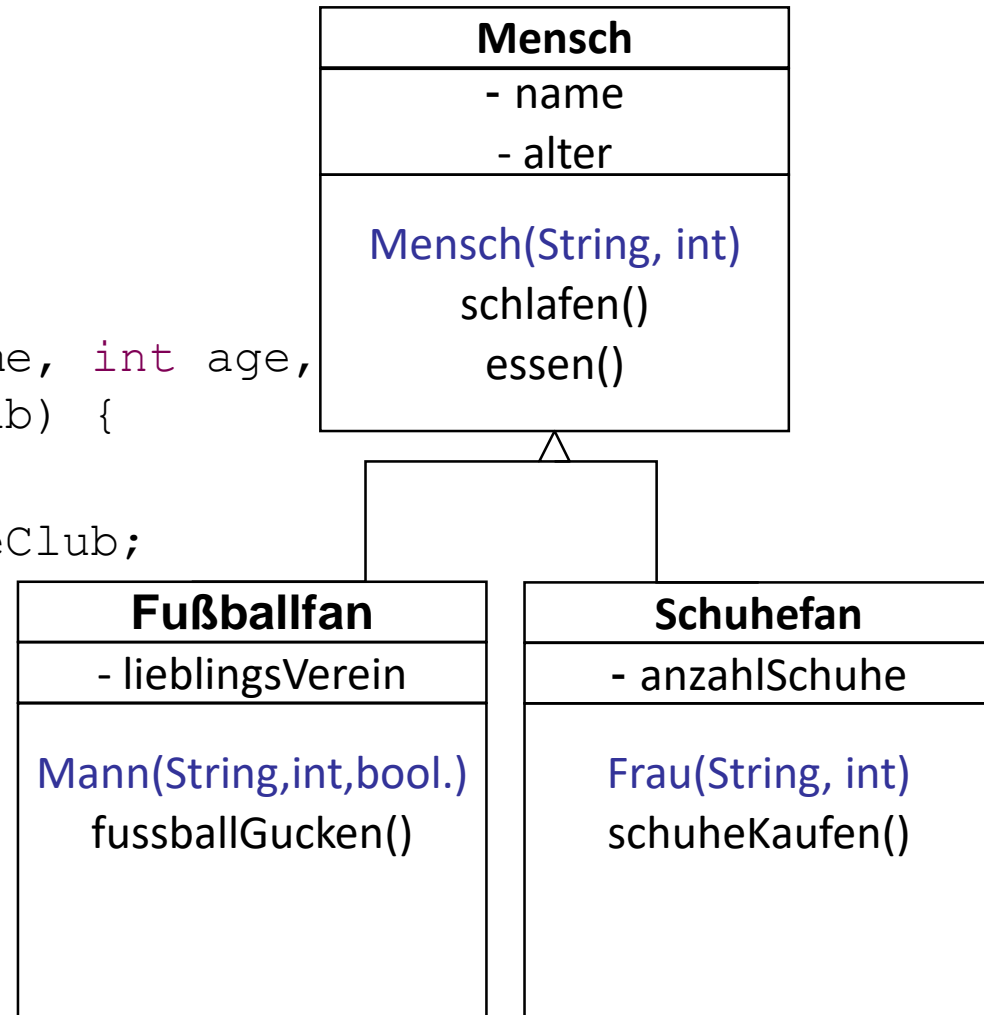
# Konstrukturen

- Folgerungen aus semantischer Vererbung:
  - ⊞ In jedem Objekt der Unterklasse steckt ein Objekt der Oberklasse
  - ⊞ Wird initialisiert über Konstruktor der Oberklasse
    - ⊞ Explizit
      - Über Aufruf des Konstruktor der Oberklasse aus dem Konstruktor der Unterklasse heraus
      - `super()` bzw. `super(name, age)`
      - Muss erste Anweisung im Konstruktor der Unterklasse sein!
    - ⊞ Implizit
      - Wenn Konstruktor der Oberklasse nicht explizit aufgerufen wird
      - Implizit eingefügter Aufruf des Standardkonstruktors der Oberklasse
      - Gleichbedeutend mit `super()` in der ersten Zeile des Konstruktors der Unterklasse



# Konstruktor mit `super()`

```
public Person (String name,  
               int age) {  
    this.name = name;  
    this.age = age;  
}  
  
public Fussballfan (String name, int age,  
                   String favoriteClub) {  
    super (name, age);  
    this.favoriteClub = favoriteClub;  
}  
  
public Schuhefan (String name,  
                 int age) {  
    super (name, age);  
    pairsOfShoes = 0;  
}
```







# Konstruktor mit `this()`

## ➤ Zur Erinnerung

- ⌘ Aufruf eines anderen Konstruktor der gleichen Klasse: `this()`
- ⌘ Muss als erste Anweisung im Konstruktorrumpf stehen
- ⌘ Nützlich, um Redundanzen in den Konstruktoren zu vermeiden

## ➤ Beispiel:

```
⌘ public Schuhefan (String name, int pairsOfShoes) {  
    this.name = name;  
    this.pairsOfShoes = pairsOfShoes;  
}
```

```
⌘ public Schuhefan (String name) {  
    this (name, 0);  
}
```



# Übung – Konstruktoren

## ➤ Live Übung

- ✚ Bearbeiten Sie **Aufgabe 3** des Blatts Live Übung „Vererbung“
- ✚ Sie haben 5 Minuten Zeit.

