GENERALISIERUNG / SPEZIALISIERUNG

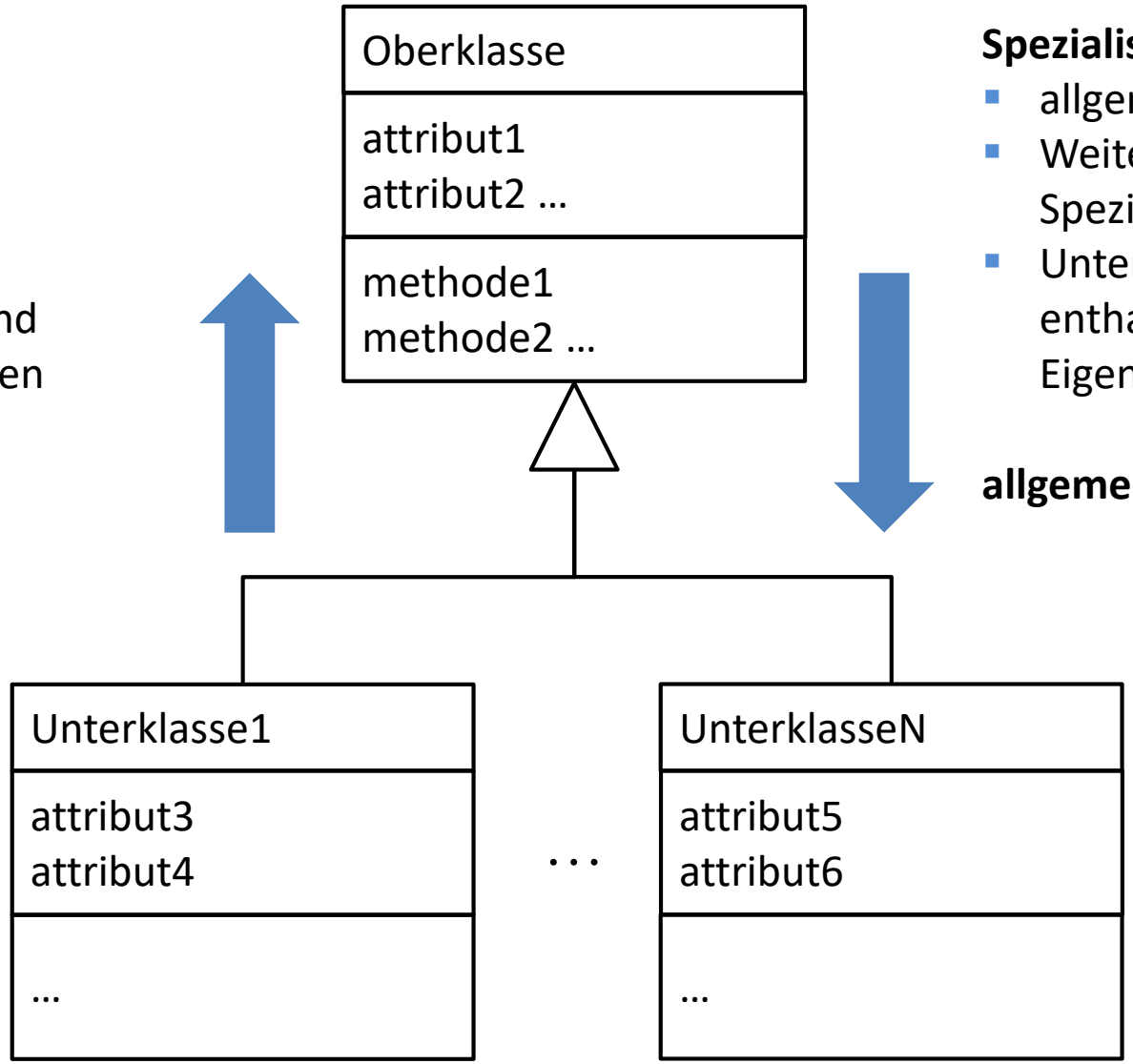
Übersicht

- Generalisierung / Spezialisierung
- Beispiel Fahrzeuge
- extends
- Überschreiben
- Überladen
- Typabstraktion
- Weitere Beispiele für Vererbung

Generalisierung / Spezialisierung /1

Generalisierung
- spezielle Unterklassen sind vorhanden
- Gemeinsamkeiten werden gesucht
- Oberklasse wird gebildet und enthält die Gemeinsamkeiten

speziell ➔ allgemein



Spezialisierung
- allgemeine Oberklasse ist vorhanden
- Weiterentwicklung zur Spezialisierung
- Unterklassen werden gebildet und enthalten jeweilige spezielle Eigenschaften

allgemein ➔ speziell

Praktisch: während der Softwareentwicklung stellt man fest, dass die **Unterklassen Gemeinsamkeiten** haben und dass diese für **Synergien zwischen** den **Unterklassen** modelliert werden müssten.

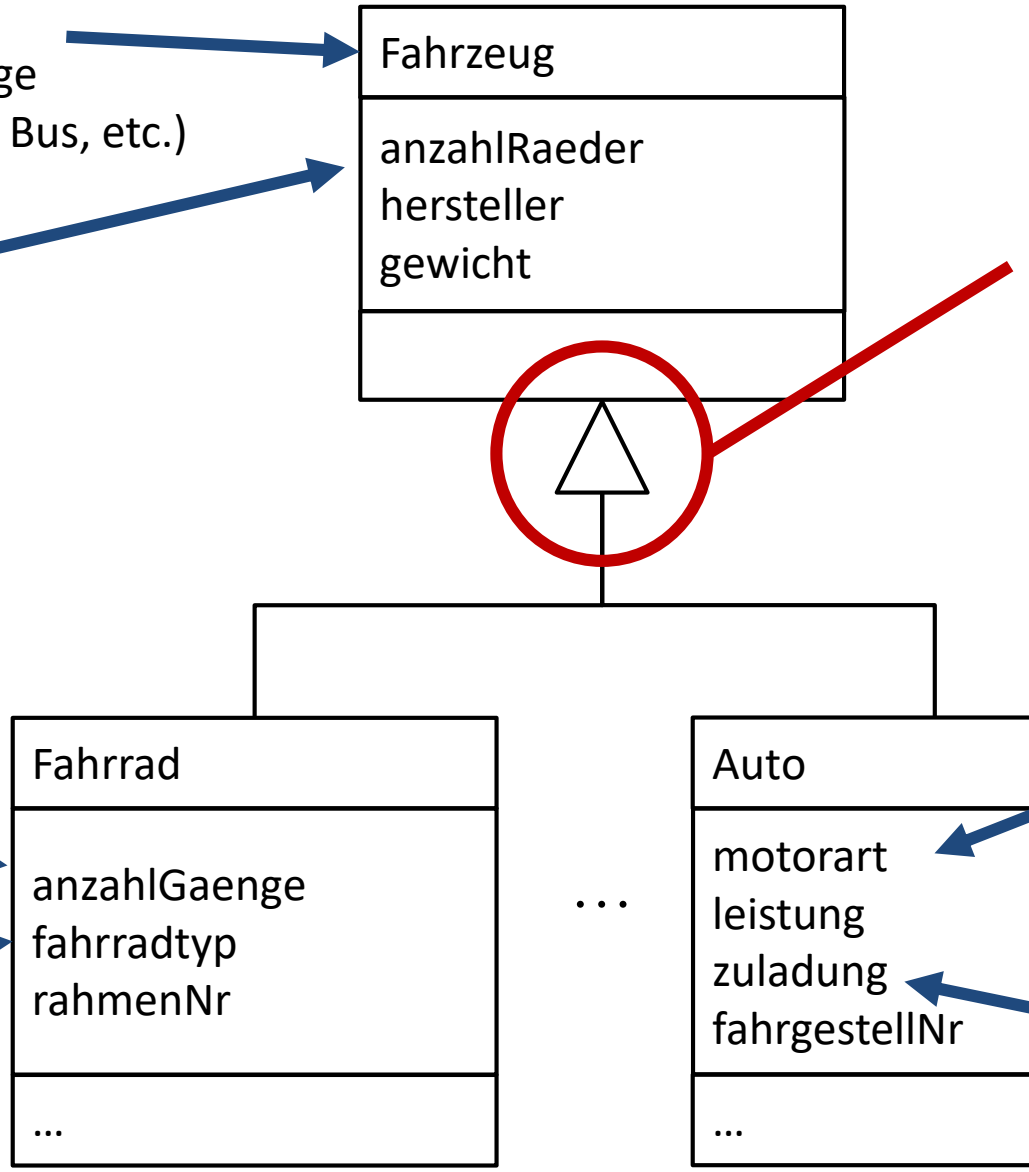
Praktisch: vor der vollständigen **Modellierung** ist schon bekannt, dass es spätere speziellere Klassen geben wird, mit **gemeinsamen Eigenschaften/Methoden**, die vorab schon benannt werden können.

Generalisierung / Spezialisierung /2

demo004.fahrzeuge

Fahrzeug ist ein Oberbegriff für verschiedene spezielle Fahrzeuge (Fahrrad, Motorrad, Auto, LKW, Bus, etc.)

Alle Fahrzeuge haben diese allgemeinen Eigenschaften



Zeichen lesen als:
Oberklasse (hier oben) hat Eigenschaften, die auch in Unterklassen zur Beschreibung der Objekte vorhanden sind

Anzahl der Gänge am Fahrrad. Spezielle Eigenschaft eines Fahrrades.

Unterschiedliche Fahrradtypen möglich (Rennrad, Mountainbike, Stadtrad, etc.)

Elektromotor, Dieselmotor, Benzinmotor, etc.

mögliche Zuladung für Transport in kg

Schlüsselwort extends

```
public class Fahrzeug {
     private int anzahlRaeder;
     private String hersteller;
     private float gewicht;
     ...
}
```

Attribute sind hier private. Die Unterklasse kann über die Get-/Set-Methoden auf die Attributwerte zugreifen, nicht direkt über this…

```
public class Fahrrad extends Fahrzeug {
     private int anzahlGaenge;
     private Fahrradtyp fahrradtyp;
     private String rahmenNr;
     ...
}
```

extends:

- Unterklasse (Fahrrad) erbt von Oberklasse (Fahrzeug)
- Unterklasse verfügt auch über die Attribute der Oberklasse
- Zugriffsrechte der Oberklasse beachten

Konstruktor bei Vererbung

```
public class Fahrzeug {
...

    public Fahrzeug(int anzahlRaeder, String hersteller,
                    float gewicht) {
        this.anzahlRaeder=anzahlRaeder;
        this.hersteller=hersteller;
        this.gewicht=gewicht;
    }
...
```

super:
Konstruktor der Oberklasse muss aufgerufen werden und führt entsprechende Initialisierung durch.

```
public class Fahrrad extends Fahrzeug {
...

    public Fahrrad(int anzRaeder, String hersteller, float gewicht, int anzGaenge,
                   Fahrradtyp fahrradtyp, String rahmenNr) {
        super(anzRaeder, hersteller, gewicht);
        this.anzahlGaenge=anzGaenge;
        this.fahrradtyp=fahrradtyp;
        this.rahmenNr=rahmenNr;
    }
...
```

Überschreiben

Überschreiben von Methoden

- In Unterklasse ist andere Implementierung für Methode der Oberklasse möglich
- Überschreiben passiert bei
  - **gleichem Methodennamen und**
  - **gleicher Signatur** (Parameter bei Aufruf und Rückgabe)
- Aufruf der überschriebenen Methoden möglich in Unterklasse

Fahrzeug

```
public String toString() {
    return "Raederanzahl: " + anzahlRaeder+"Hersteller: "+hersteller+"Gewicht: "+gewicht;
}
```

Fahrrad

```
public String toString() {
    return super.toString() + ", Ganganzahl: "+anzahlGaenge+", Fahrradtyp: "+fahrradtyp+", RahmenNr: "+rahmenNr;
}
```

Methode toString() in Oberklasse kann aufgerufen werden

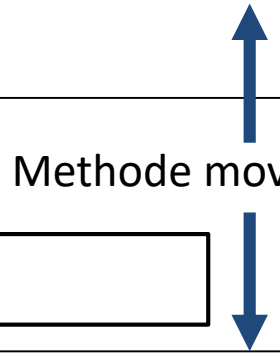
Überladen

Überladen von Methoden

- In Unterklasse ist andere Implementierung für Methode der Oberklasse möglich
- Überladen passiert bei
  - **gleichem Methodennamen** und
  - **unterschiedlicher Signatur** (Parameter bei Aufruf und Rückgabe)
- mehrfaches Überladen möglich

Fahrzeug

```
public void move(int meter) {
    ...
}
```



Methode move(…) wird überladen

Fahrrad

```
public void move(int gang, int umdrehung) {

}
public void move(int gang, int umdrehung, float kraft) {

}
```

Typabstraktion /1

Typabstraktion

- Variable mit Datentyp der Oberklasse kann Objekt der Unterklasse referenzieren
- aufgerufene Methoden werden dynamisch, typabhängig gebunden

Beispiel:

Oberklasse Unterklasse

```
Fahrzeug einFahrzeug = new Auto(4, "CarProducer", 1500.0f, Motorart.ELEKTRO, 90.0f, 500.0f, "Car201");

System.out.println(einFahrzeug.toString());
```

Beim Aufruf der Methode toString() wird die Methode der Unterklasse **Auto** ausgeführt.

Typabstraktion /2

Typabstraktion

- Oberklasse kann auf konkreten Typ einer Unterklasse geprüft werden
- **`instanceof`** prüft konkreten Typ
- differenzierte Logik in Abhängigkeit von Unterklasse möglich

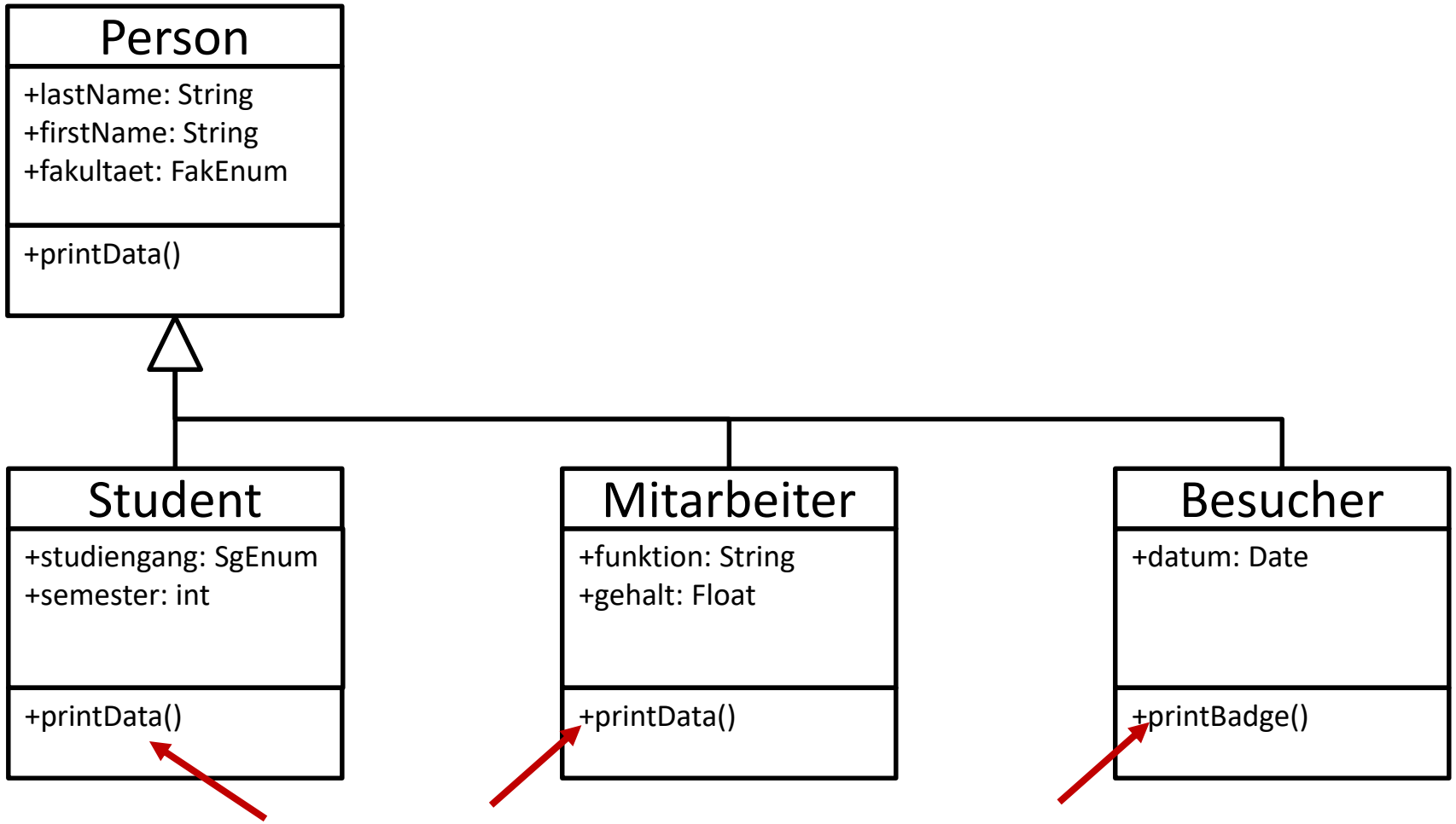
Beispiel:

Liefert **true**, falls die konkrete Unterklasse die Klasse Fahrrad ist, ansonsten **false**

```
if(einFahrzeug instanceof Fahrrad) {
     System.out.println("Variable einFahrzeug ist vom Typ Fahrrad");
     System.out.println(einFahrzeug);
     // ggf. weitere spezielle Logik für Fahrrad
} else if(einFahrzeug instanceof Auto) {
     System.out.println("Variable einFahrzeug ist vom Typ Auto");
     System.out.println(einFahrzeug);
     // ggf. weitere spezielle Logik für Auto
}
```

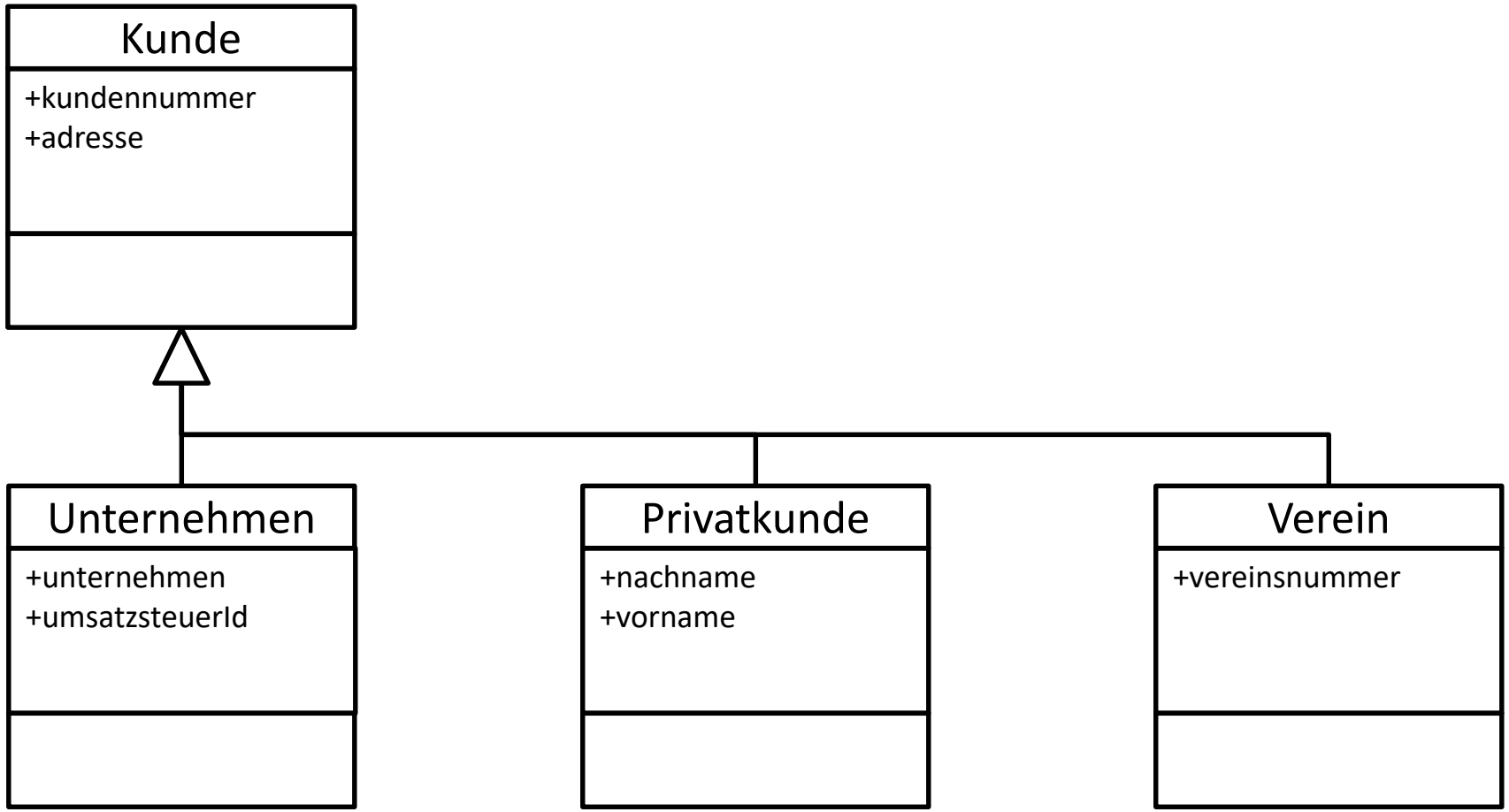
BEISPIELE

Beispiel Vererbung

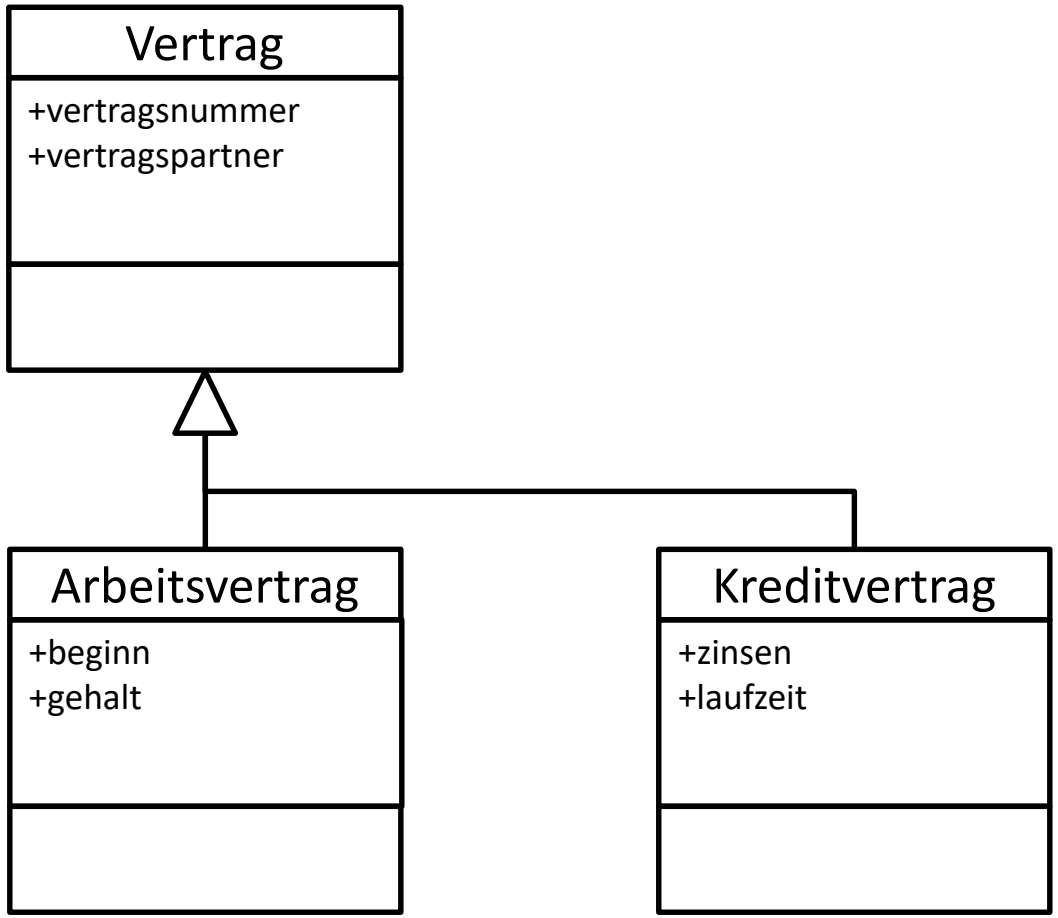


printData() überschreiben, um auch erweiterte Attribute auszugeben. printBadge() muss individuell implementiert werden.

Beispiel Vererbung



Beispiel Vererbung



Zusammenfassung

Man sagt: Die Unterklasse erbt von der Oberklasse

- alle Attribute
- alle Methoden
- Zugriffsrechte beachten

Unterklasse

- implementiert **zusätzliche** Attribute und Methoden
- kann Methoden **überschreiben**
- Methoden können **überladen** werden

Typabstraktion

- Variable mit Datentyp der Oberklasse kann Objekt der Unterklasse referenzieren
- Prüfung auf konkreten Typ möglich mit **`instanceof`**