Universität Bremen

Praktische Informatik 1
Bessere Struktur durch Vererbung 1

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

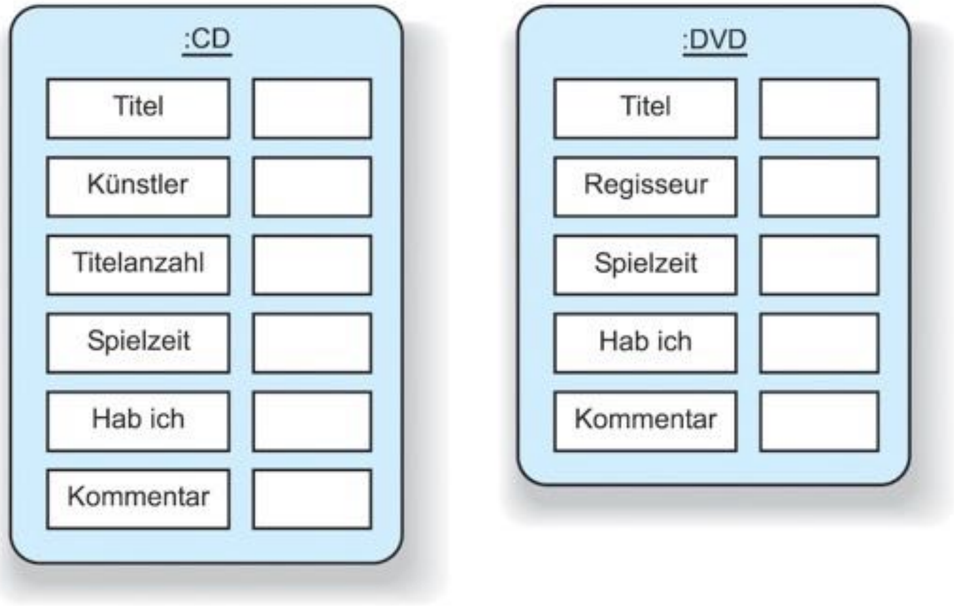
Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

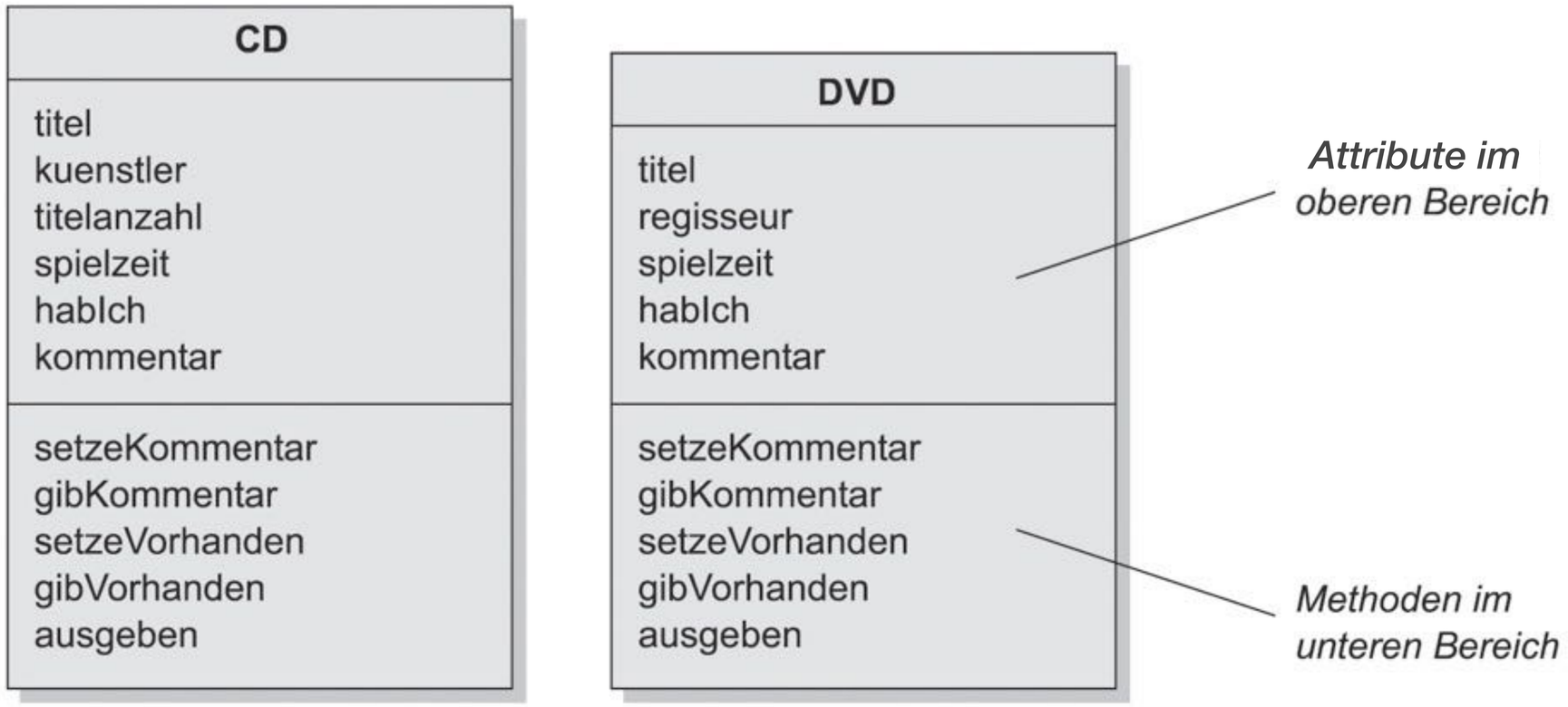
DoME: Database of Multimedia Entertainment

- Speichert Details zu CDs und DVDs
  - **CD**: Titel, Künstler, Titelanzahl, Spielzeit, Besitz (hab ich), Kommentar
  - **DVD**: Titel, Regisseur, Spielzeit, Besitz (hab ich), Kommentar
- Ausgabe von Listen
- Suche

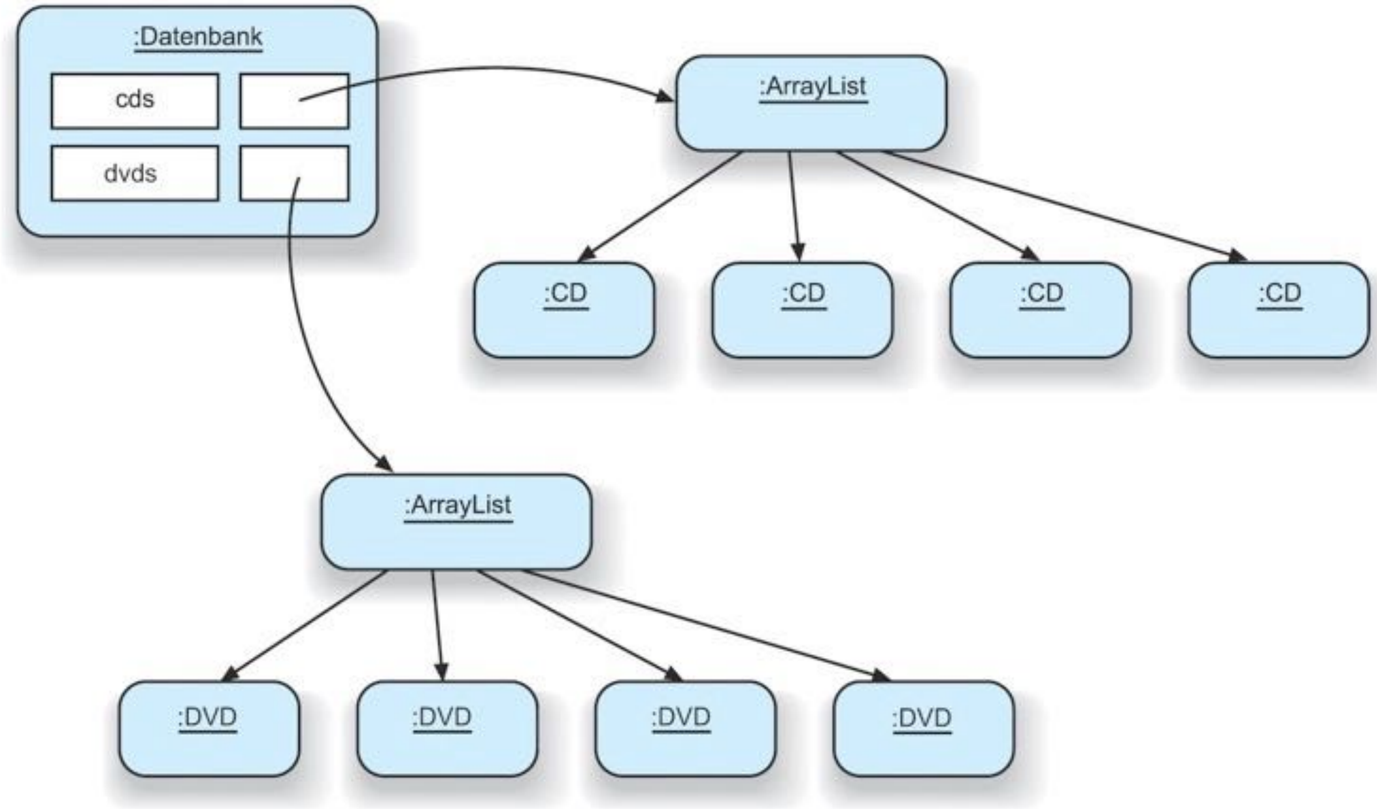


Klassen von DoME

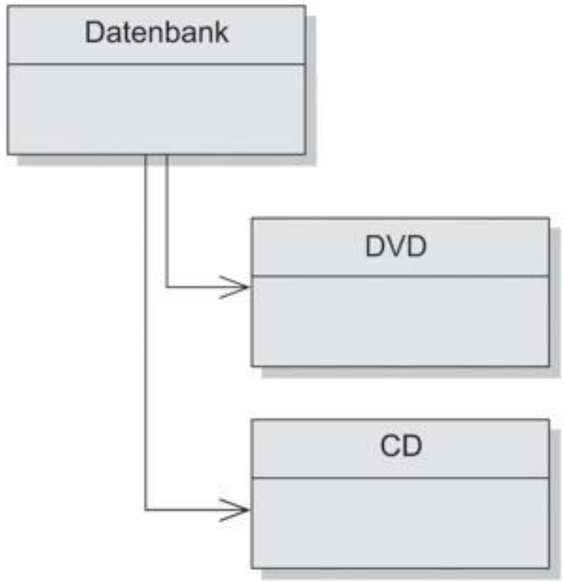
- Notation orientiert sich an UML (Unified Modeling Language)



DoME: Entwurf



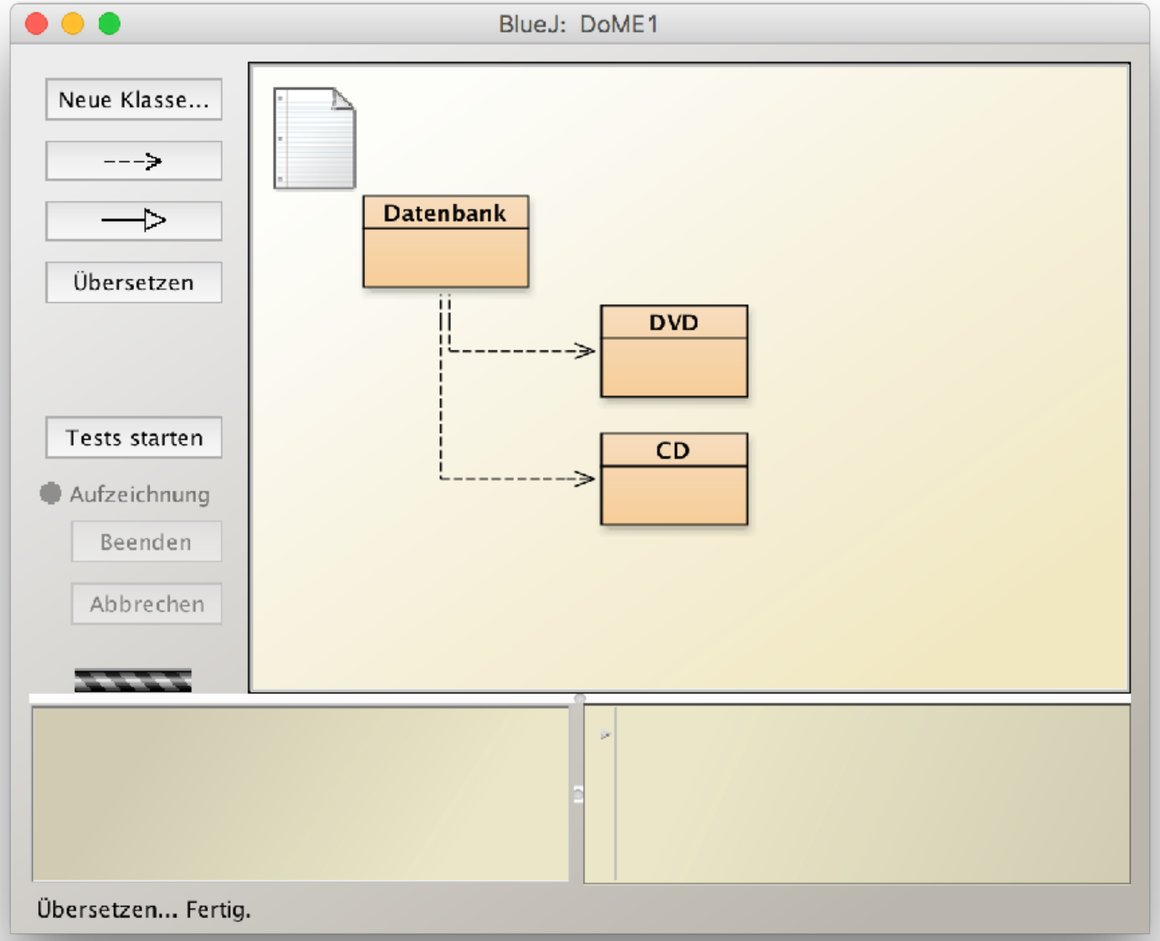
Objektdiagramm



Klassendiagramm

Universität Bremen

DoME: Demo



Universität Bremen

Kritik am Entwurf von DoME

- Code-Duplizierung
  - Klassen **CD** und **DVD** sehr ähnlich
  - Duplizierter Code in Klasse **Datenbank**
- Eine Lösung: Gemeinsamkeiten in neuer Klasse zusammenfassen, von der **CD**s und **DVD**s jeweils eine Instanz enthalten
- Bessere Lösung: **Vererbung**

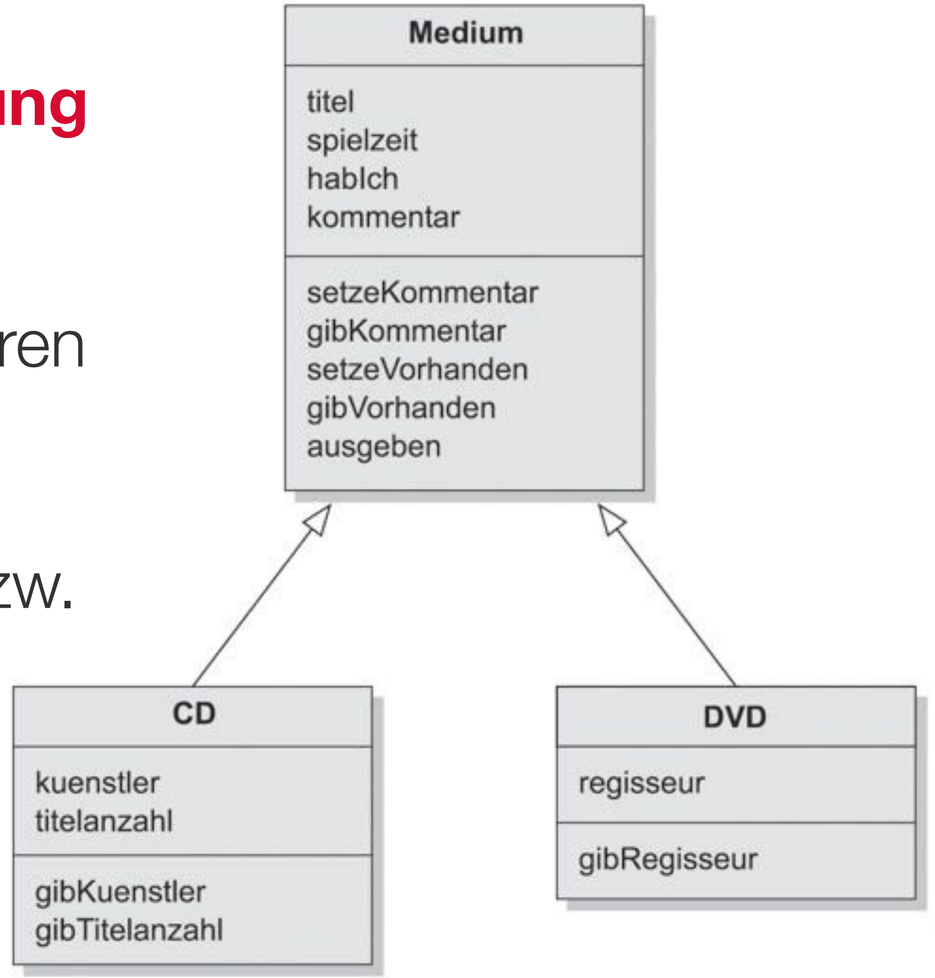
```
public CD(final String titel, final String künstler,
        final int titelanzahl, final int spielzeit)
public DVD(final String titel, final String regisseur,
       final int spielzeit)
{
    this.titel = titel;
    this.künstler = künstler;
    this.regisseur = regisseur;
    this.titelanzahl = titelanzahl;
    this.spielzeit = spielzeit;
    habIch = false;
    kommentar = "<kein Kommentar>";
}

/**
 * Setze einen Kommentar für diese CD.
 * Setze einen Kommentar für diese DVD.
 * @param kommentar der einzutragende Kommentar.
 */
public void setzeKommentar(final String kommentar)
{
    this.kommentar = kommentar;
}

/**
 * @return den Kommentar für diese CD.
 * @return den Kommentar dieser DVD.
 */
public String gibKommentar()
{
    return kommentar;
}
```

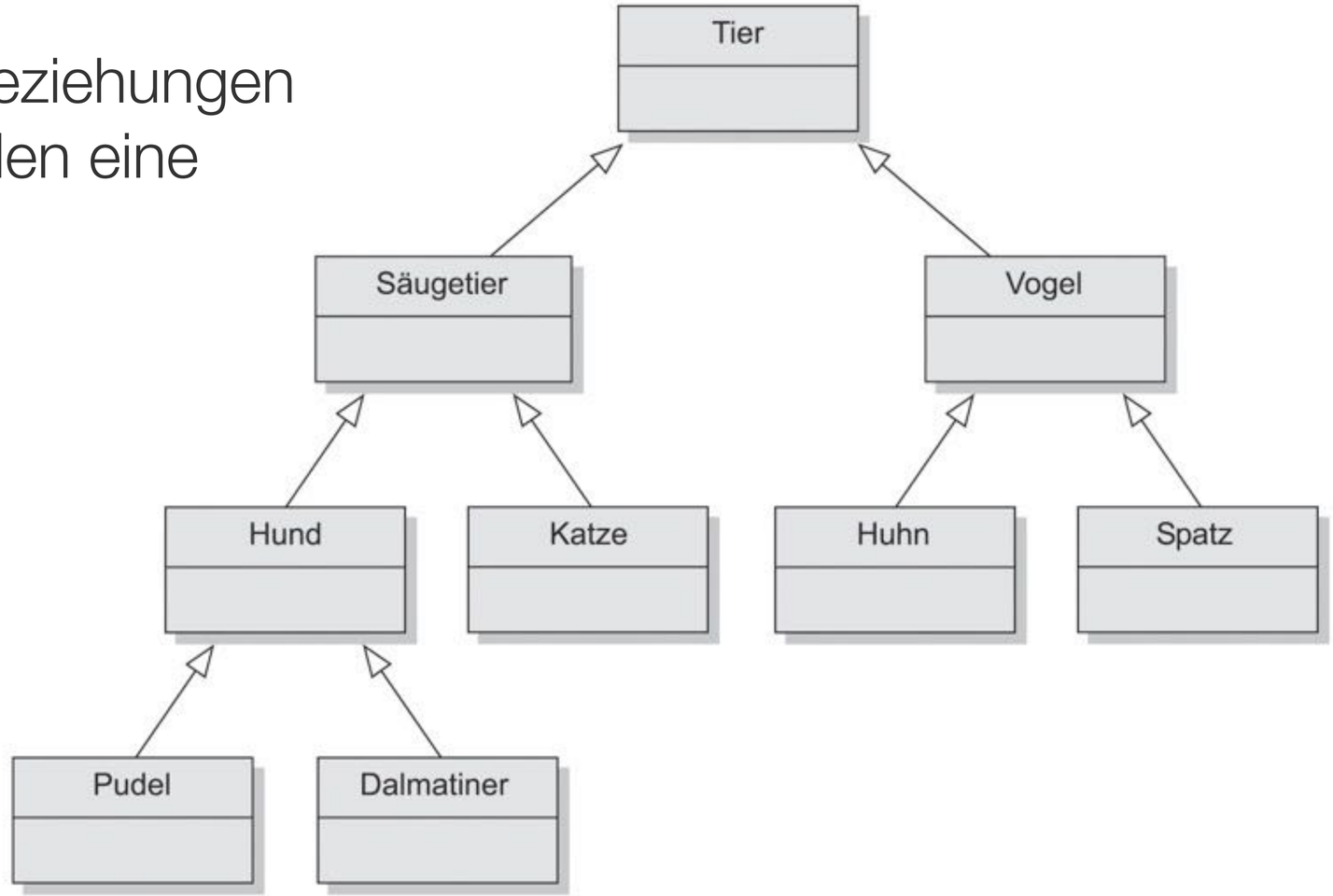
Vererbung

- Vererbung erlaubt es, eine Klasse als **Erweiterung** einer anderen zu definieren
- Eine **Superklasse** ist eine Klasse, die von anderen Klassen erweitert wird
- Eine **Subklasse** erweitert eine andere Klasse bzw. erbt von dieser
- Sie erbt alle **Methoden** und **Attribute** ihrer Superklasse

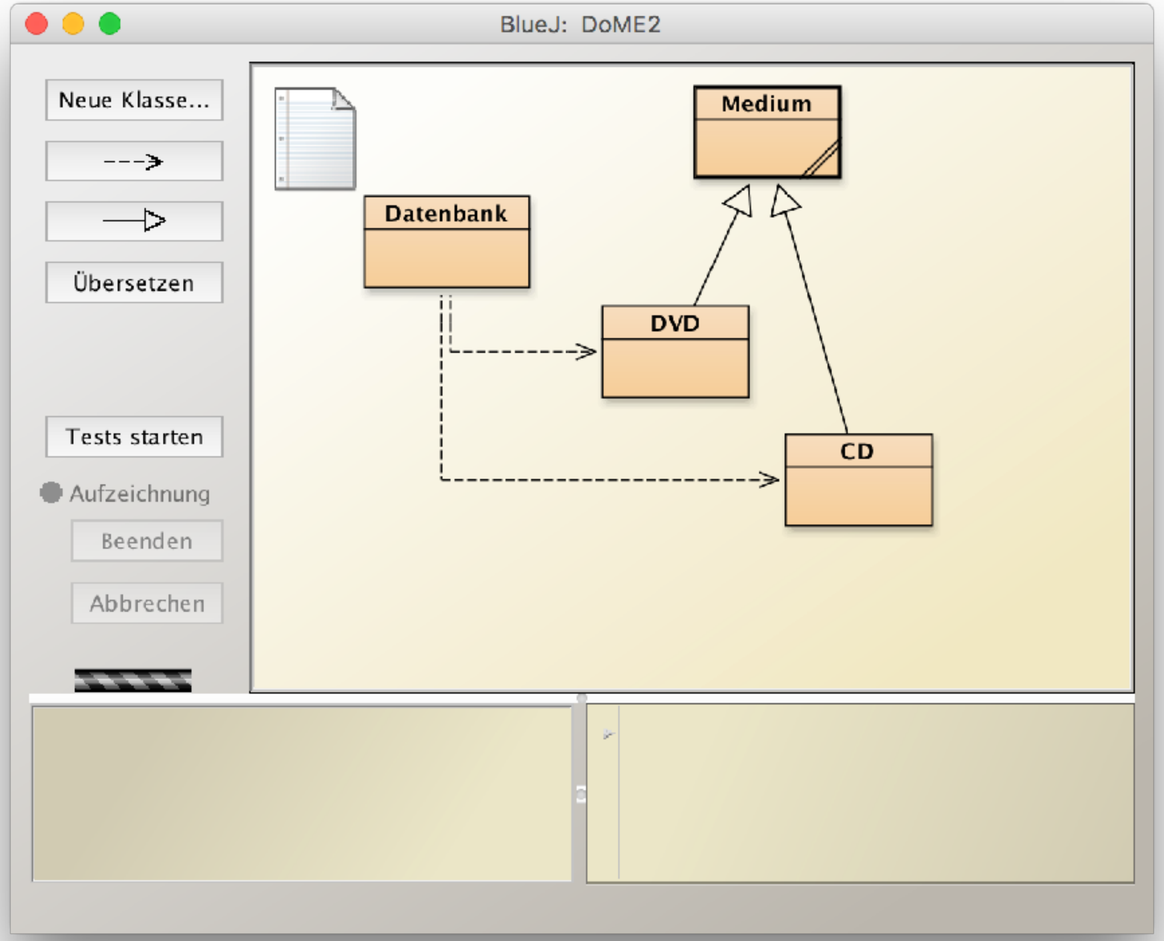


Vererbungshierarchien

- Klassen, die über Vererbungsbeziehungen miteinander verknüpft sind, bilden eine **Vererbungshierarchie**



Vererbung: Demo



Universität Bremen

Vererbung in Java

- Eine Klasse **K** erbt von einer Superklasse **S**:
  class K extends S

- Jede Klasse kann nur eine unmittelbare Superklasse haben

- Eine Subklasse erbt zwar alle **Methoden** und **Attribute** ihrer Superklasse, kann aber nur die direkt verwenden, die nicht **private** sind

  - Weitere Einschränkungen gelten, wenn Sub- und Superklasse nicht im selben Paket

```
class Medium
{
    private final String titel;
    private final int spielzeit;
    private boolean hablch;
    private String kommentar;
                    :
```

```
class CD extends Medium
{
    private final String künstler;
    private final int titelanzahl;
                    :
```

```
class DVD extends Medium
{
    private final String regisseur;
                    :
```

Universität Bremen

Vererbung und Initialisierung

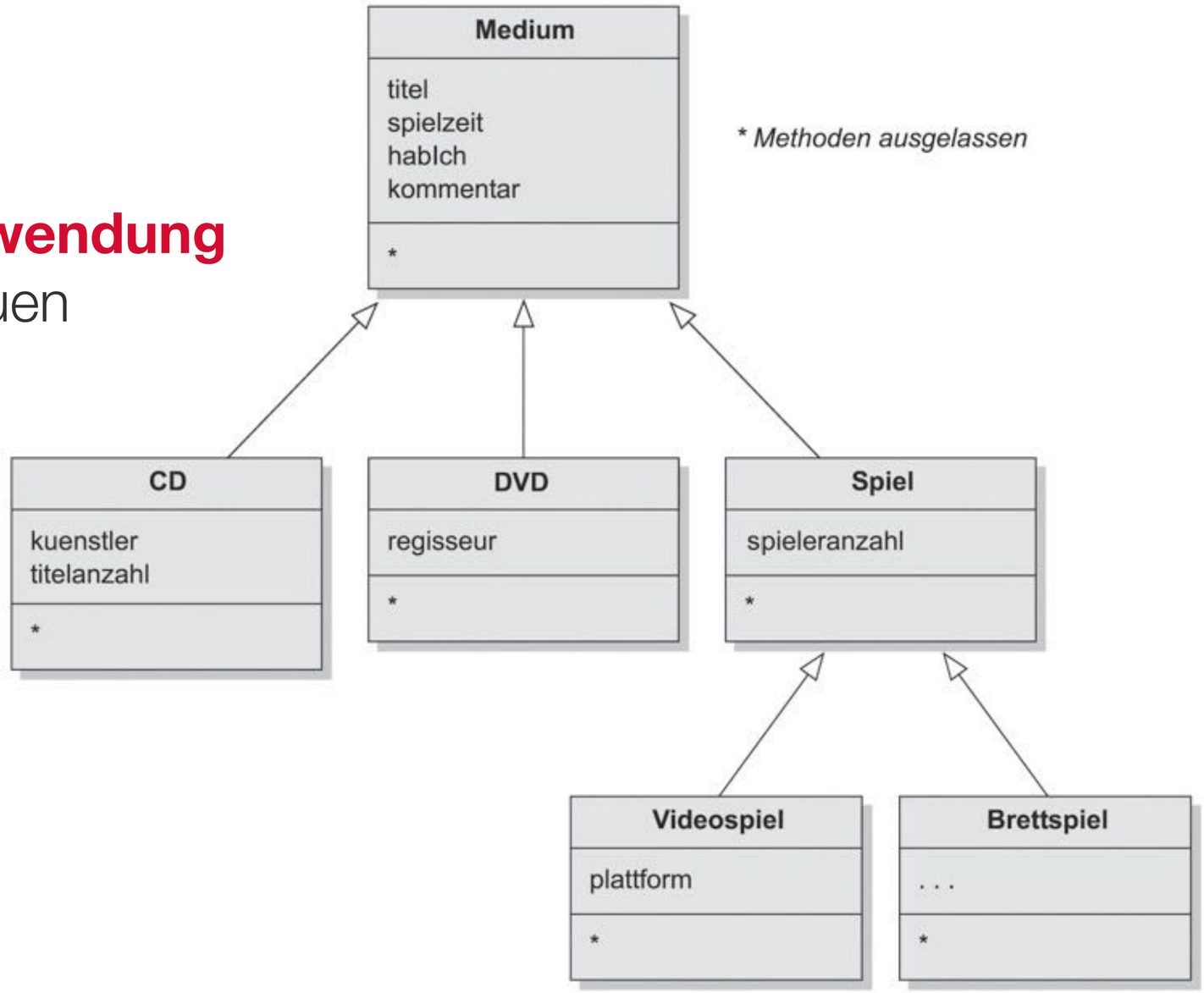
- Jeder Konstruktor einer Subklasse ruft zu Beginn den Konstruktor der Superklasse auf
- Aufruf kann explizit gemacht und somit gewählt werden, welcher Konstruktor der Superklasse aufgerufen wird (als **erste Anweisung**)
- Wird kein Konstruktoraufruf angegeben, ruft Java automatisch **super()** auf
  - Übersetzungsfehler, wenn die Superklasse keinen Konstruktor ohne Parameter hat

```
public Medium(
        final String titel,
        final int spielzeit)
{
    this.titel = titel;
    this.spielzeit = spielzeit;
    habIch = false;
    kommentar = "";
}
```

```
public CD(final String titel,
        final String künstler,
        final int titelanzahl,
        final int spielzeit)
{
    super(titel, spielzeit);
    this.künstler = künstler;
    this.titelanzahl = titelanzahl;
}
```

Wiederverwendung

- Vererbung erlaubt **Wiederverwendung** bereits erstellter Klassen in neuen Zusammenhängen



Universität Bremen

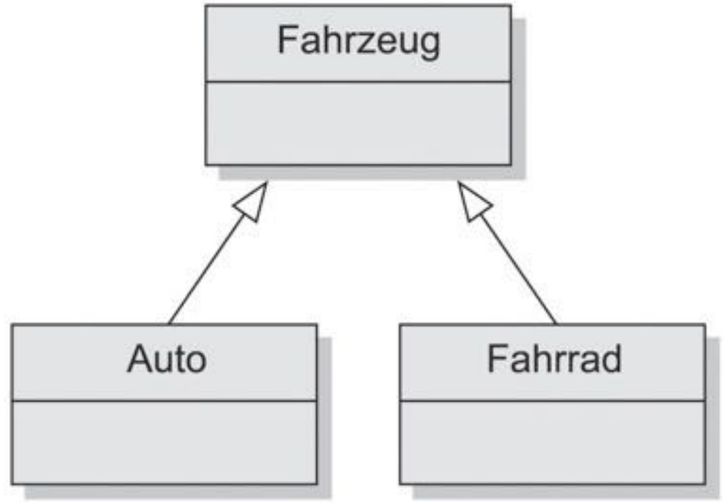
Vorteile von Vererbung (bis jetzt)

- Vermeidung von **Code-Duplizierung**
- **Wiederverwendung** von Quelltext
- **Einfachere Wartung**
- **Erweiterbarkeit**

Universität Bremen

Subtyping

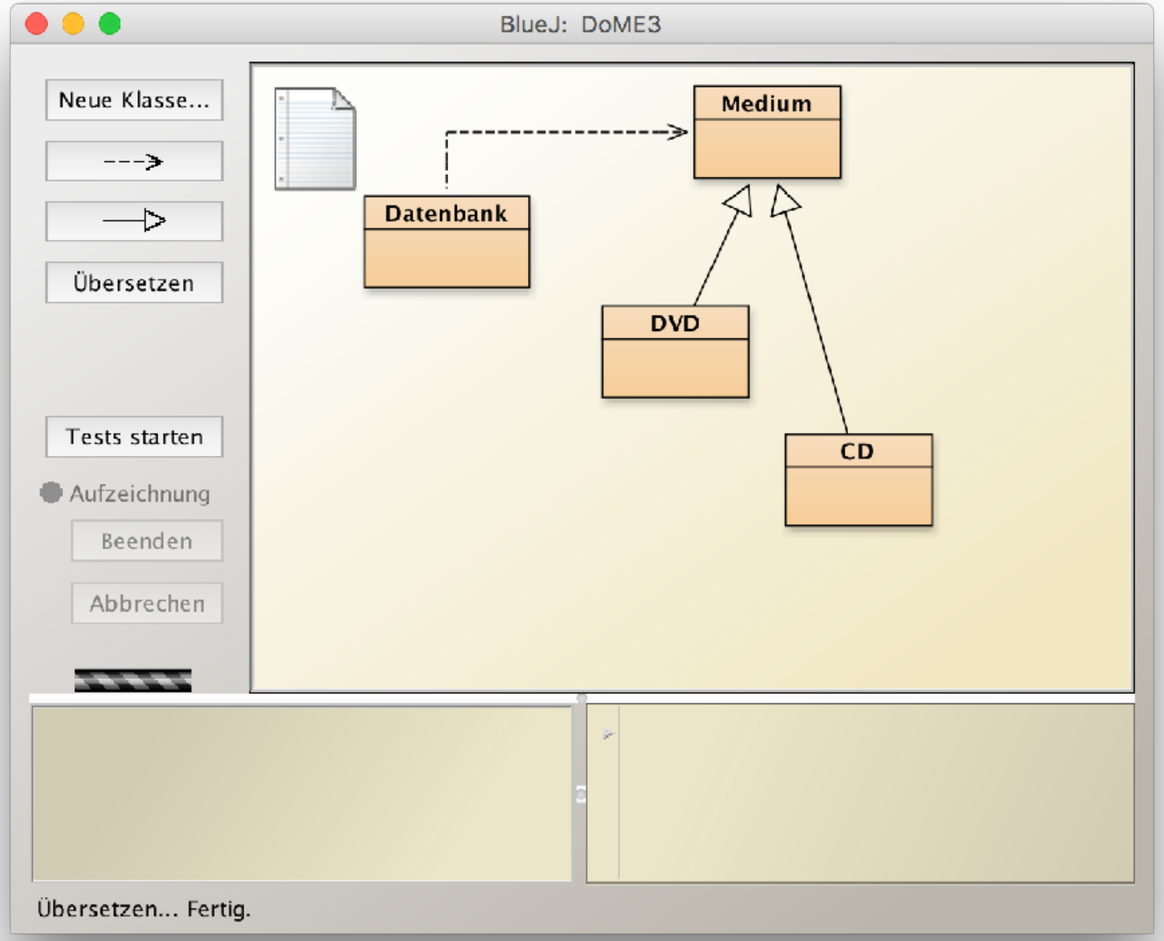
- **Klassen** definieren **Typen**
- **Subklassen** definieren **Subtypen**
- Analog zur **Klassenhierarchie** existiert eine **Typhierarchie**
- **Ersetzbarkeit**: Objekte von Subtypen sind verwendbar an Stellen, an denen ein Supertyp erwartet wird



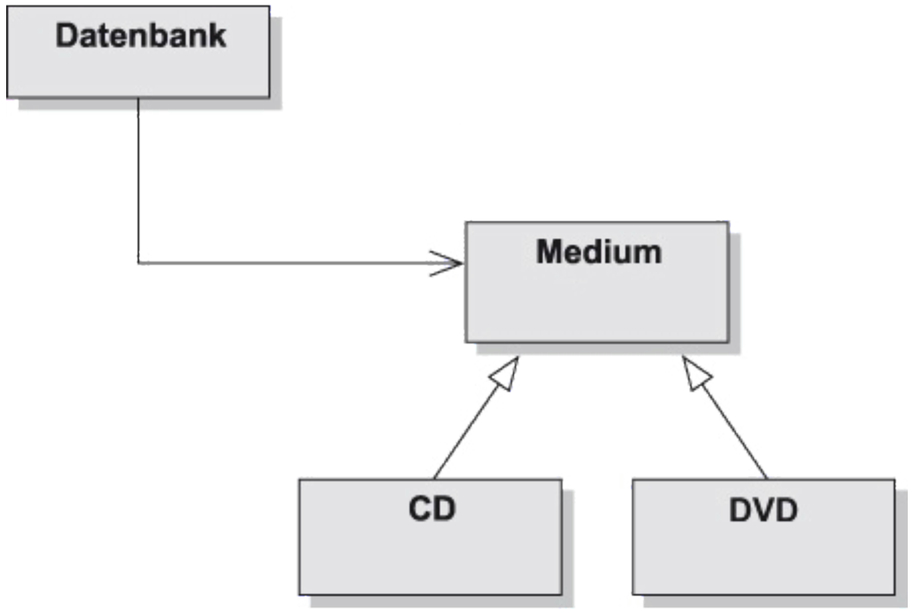

```
Fahrzeug f1 = new Fahrzeug();
Fahrzeug f2 = new Auto();
Fahrzeug f3 = new Fahrrad();
```

```
Auto a = new Fahrzeug(); // Fehler
```

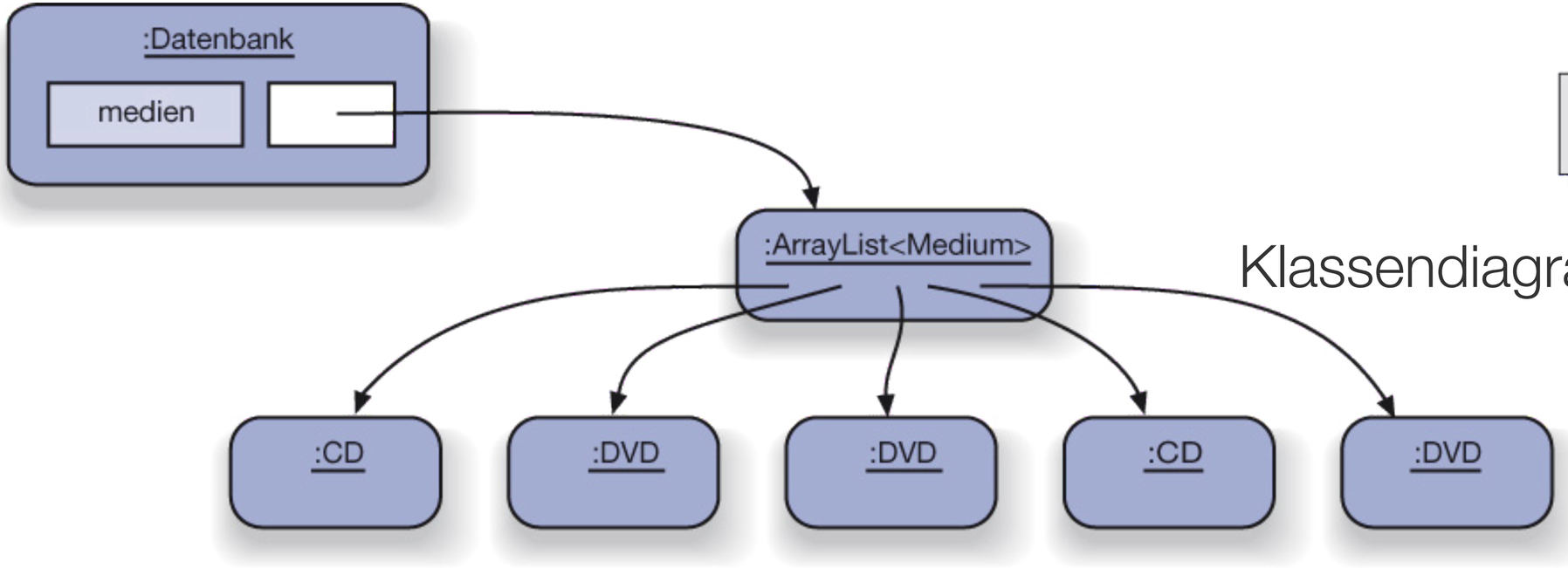
Polymorphie: Demo



DoME: Entwurf mit Vererbung



Klassendiagramm



Objektdiagramm

Universität Bremen

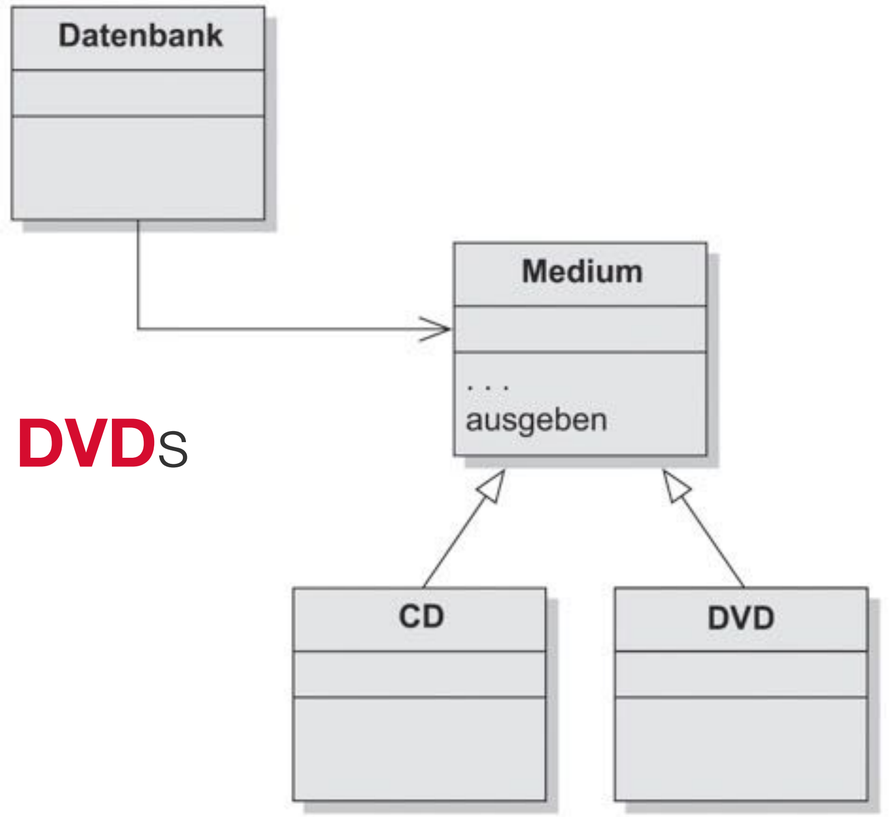
Polymorphe Variablen

- Objektvariablen sind in Java **polymorph** (vielgestaltig)
- Sie können Objekte des deklarierten Typs oder von Subtypen enthalten
- Z.B. sind Variablen vom Typ **Medium** polymorph: sie können **DVD**s oder **CD**s enthalten

```
for (final Medium medium : medien) {
    medium.ausgeben();
    System.out.println();
}
```

Problem in DoME nach Refactoring

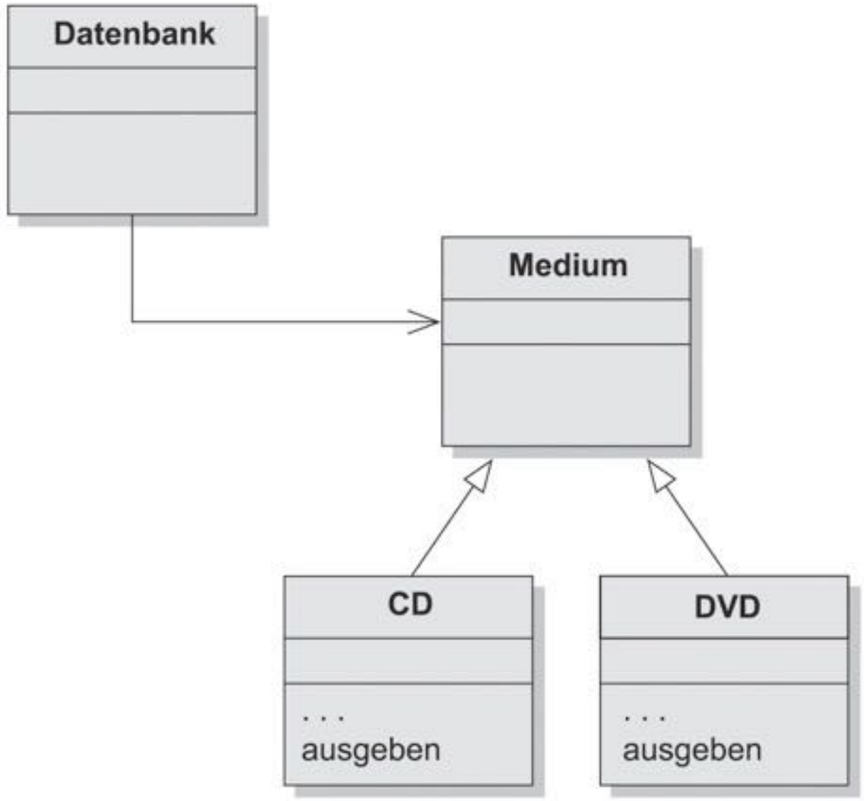
- **ausgeben** in Klasse **Medium**
  - Gibt nur gemeinsame Attribute aus
  - Also keine spezifischen Attribute von **CD**s und **DVD**s
- Vererbung ist Einbahnstraße
  - Subklasse erbt Attribute der Superklasse
  - Superklasse kennt Attribute der Subklasse nicht



Universität Bremen

Lösungsversuch

- Verschieben von **ausgeben** nach **CD** und **DVD**
- Probleme
  - Attribute von **Medium private**
  - **Datenbank** findet in **Medium** kein **ausgeben**



Universität Bremen

Statischer und dynamischer Typ

- Welchen Typ hat **dvd**?

```
DVD dvd = new DVD(...);
```

- Welchen Typ hat **medium**?

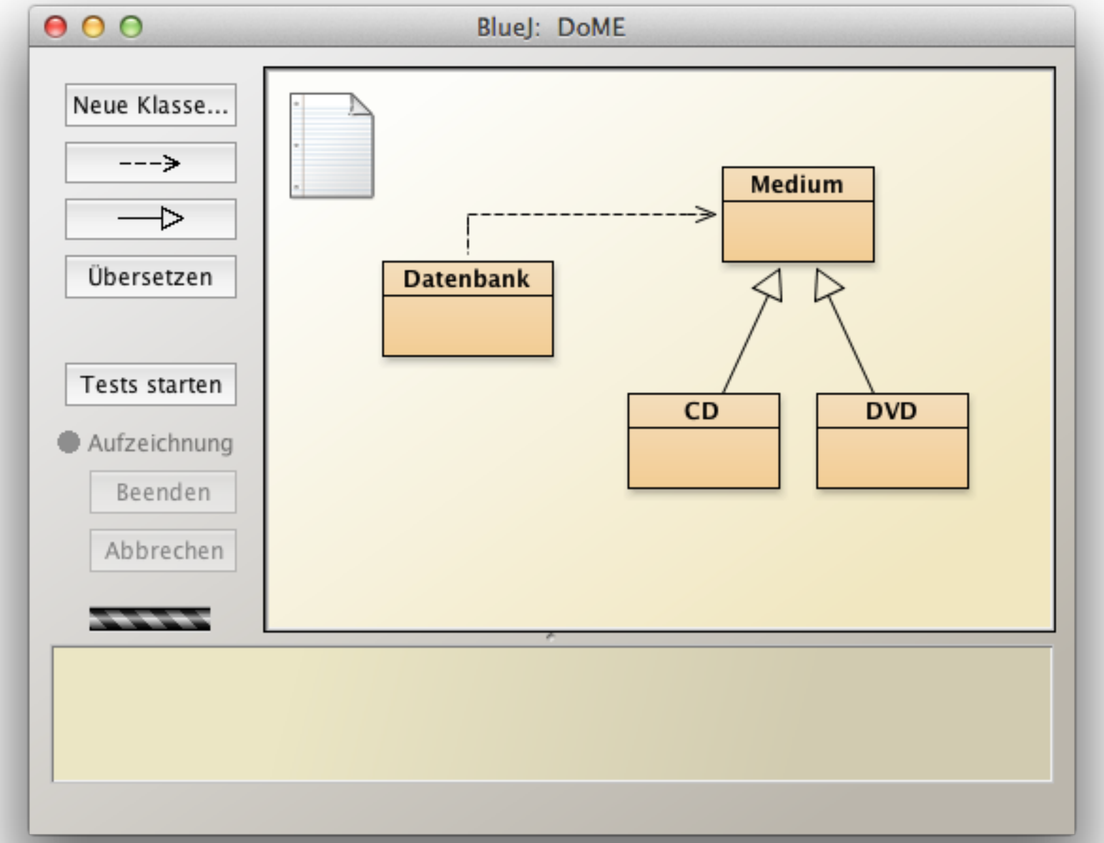
```
Medium medium = new DVD(...);
```

- **Statischer Typ**: Deklarierter Typ im Quelltext
  - Wird beim Übersetzen geprüft

- **Dynamischer Typ**: Typ des Objekts, das gerade in Variable gespeichert ist
  - Ist erst zur Laufzeit bekannt

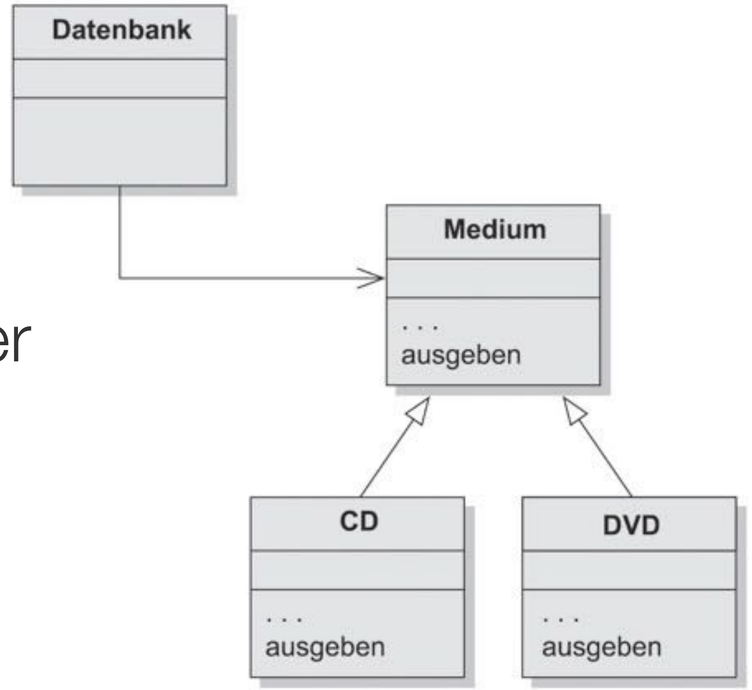
Universität Bremen

Überschreibung: Demo



Überschreiben von Methoden

- Deklaration einer Methode in einer Subklasse mit **identischer Signatur** wie in der Superklasse
- Beim Aufruf wird die Methode in der Subklasse statt der Methode in der Superklasse verwendet
- Statische und dynamische Typprüfungen funktionieren
  - Was kann aufgerufen werden: **statischer Typ**
  - Was wird aufgerufen: **dynamischer Typ**



Universität Bremen

Überschreiben von Methoden

- Überschreibungen dürfen Zugriffsrechte **erweitern**, aber nicht einschränken
- Das Überschreiben einer **private**n Methode führt eine neue, unabhängige Methode ein
- Ist eine Methode als **final** deklariert, kann sie nicht überschrieben werden
- Steht **@Override** vor einer Methode, überprüft der Compiler, ob wirklich eine Überschreibung vorliegt

```
class Medium
{
    void ausgeben() ...
}
```

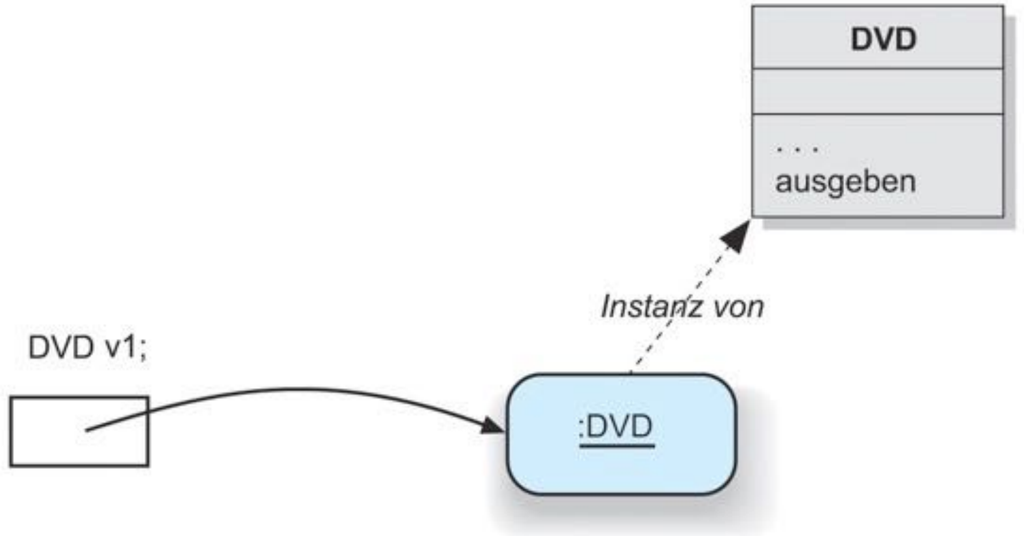
```
class CD extends Medium
{
    // erlaubt
    public void ausgeben() ...
}
```

```
class DVD extends Medium
{
    // Fehler!
    private void ausgeben() ...
}
```

Dynamische Methodensuche

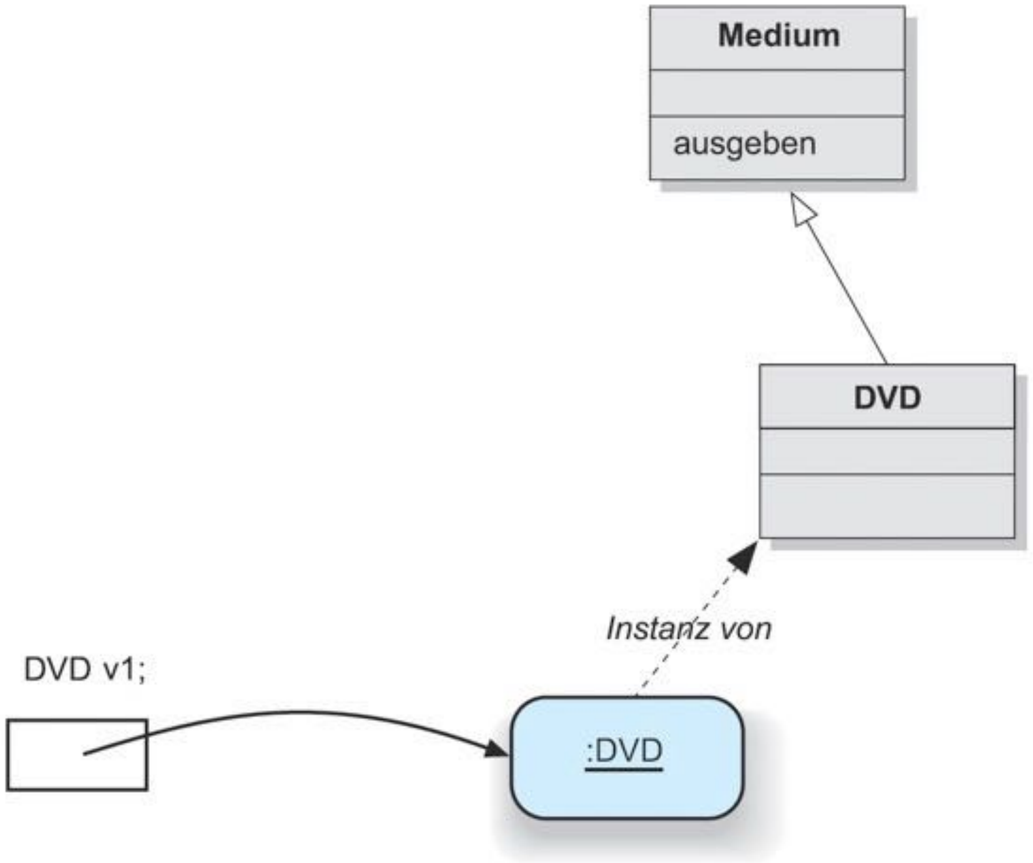
- Die Typprüfung berücksichtigt den **statischen Typ**
- Zur Laufzeit werden die Methoden des **dynamischen Typs** ausgeführt
- Beispiel: Ursprüngliche Version von DoME
  - Keine Vererbung oder Polymorphie
  - Naheliegende Methode wird ausgeführt

```
DVD v1 = new DVD(...);
v1.ausgeben();
```



Dynamische Methodensuche

- Beispiel: DoME nach erstem Refactoring
  - Vererbung, aber keine Überschreibung
  - Vererbungshierarchie hinaufwandern

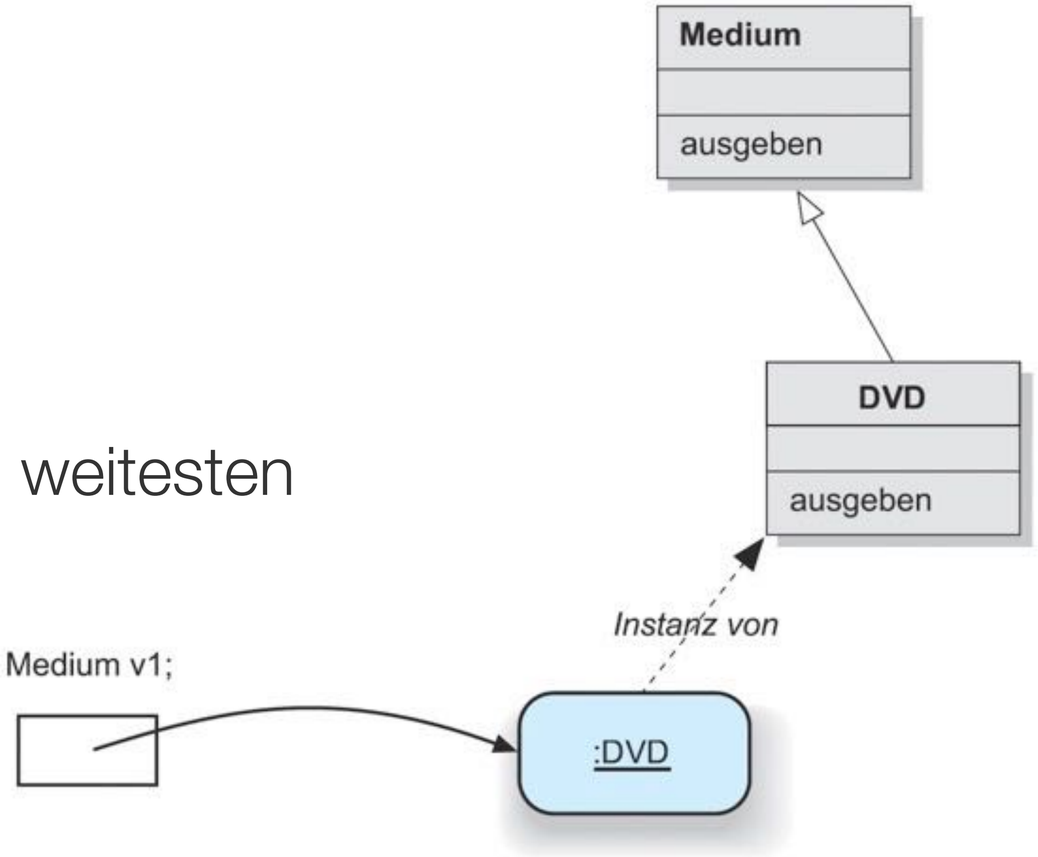



```
DVD v1 = new DVD(...);
v1.ausgeben();
```

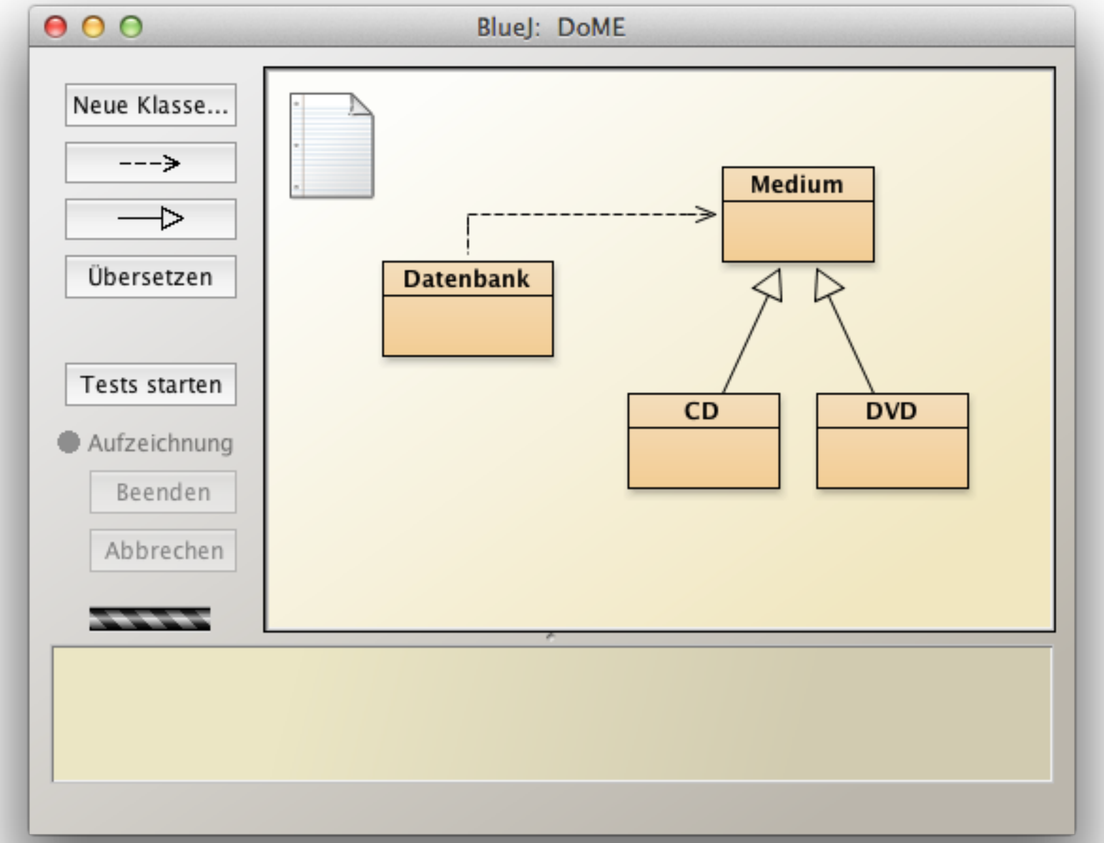
Dynamische Methodensuche

- Beispiel: DoME jetzt
  - Polymorphie und Überschreibung
  - Zuerst gefundene Version wird ausgeführt
- Es wird immer die Überschreibung in der am weitesten abgeleiteten Klasse ausgeführt

```
Medium v1 = new DVD(...);
v1.ausgeben();
```



Aufruf der überschriebenen Methode: Demo



Universität Bremen

super-Aufrufe in Methoden

- Über **super** kann explizit eine Methode der Superklasse aufgerufen werden
  - Typischerweise benutzt, um Methode aufzurufen, die überschrieben wurde
- Aufrufe über **super** sind **statisch gebunden**, d.h es ist zur Übersetzungszeit bekannt, welche Überschreibung aufgerufen wird
- Aufrufe über **this** bedeuten **nicht** „diese Klasse“ sondern „dieses Objekt“ und sind wie andere Aufrufe **dynamisch gebunden**

```
void ausgeben()
{
    super.ausgeben();
    System.out.println("     " + künstler);
    System.out.println("     " + titelanzahl);
}
```

Universität Bremen

Statisches und dynamisches Binden

- Methodenaufrufe werden dynamisch gebunden (außer über **super** oder wenn sie **private** sind)
- Zugriffe auf Attribute werden statisch gebunden

```
A a = new B();
int i = a.value; // 1
int j = a.getValue(); // 2
```

```
class A
{
    int value = 1;
    int getValue()
    {
        return value;
    }
}
```

```
class B extends A
{
    int value = 2;
    int getValue()
    {
        return value;
    }
}
```

Universität Bremen

Zusammenfassung der Konzepte

- **Vererbung** und **Vererbungshierarchie**
- **Superklasse** und **Subklasse**
- **super**
- **Subtyp** und **Ersetzbarkeit**
- **Polymorphie**
- **Statischer** und **dynamischer Typ**
- **Überschreiben** von Methoden