Universität Bremen

Praktische Informatik 1 Bessere Struktur durch Vererbung 2

Thomas Röfer

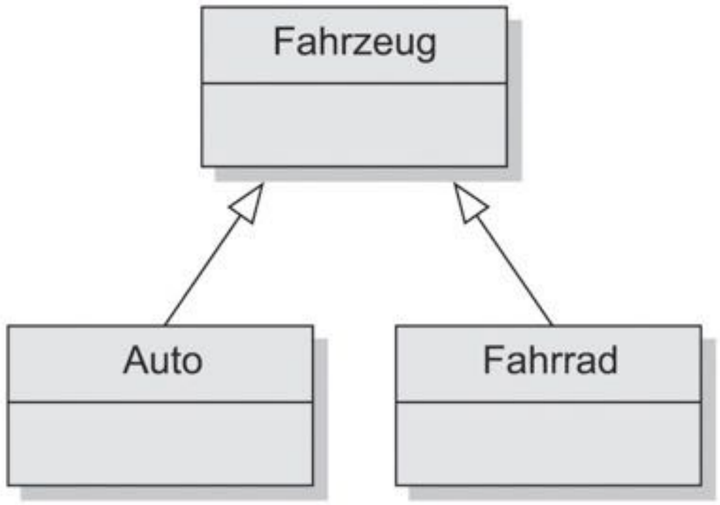
Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

Typumwandlung



- Zuweisung von Subtyp an Supertyp möglich, umgekehrt nicht

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

- **(***Typ***)**: Explizite Typumwandlung

```
a = (Auto) f; // ok
```

  - Gültigkeitsprüfung erst zur Laufzeit (außer, wenn generell unmöglich); wenn nicht gültig, dann Fehler

```
if (f instanceof Auto) {
    a = (Auto) f;
}
```

- *Objekt* **instanceof** *Klasse*: Typtest zur Laufzeitzeit, ob Objekt Instanz einer Klasse (oder deren Subklassen) ist

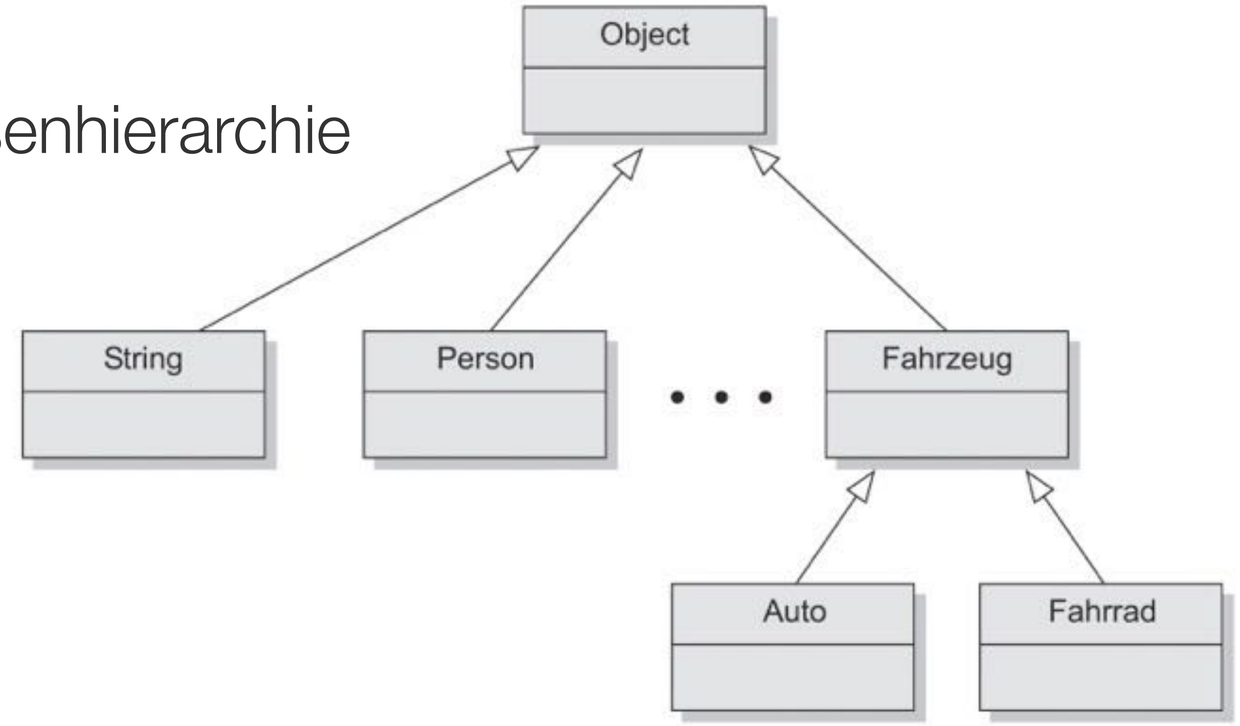
  - *Objekt* **instanceof** *Klasse Name* deklariert gleich Synonym vom getesteten Typ (**Pattern Matching**)

```
if (f instanceof Auto b) {
    a = b;
}
```

Universität Bremen

Die Klasse **Object**

- Wenn für eine Klasse keine Superklasse definiert wird, erbt sie von **Object**
  - Java ergänzt **extends Object**
- **Object** ist die **Wurzel** der Java-Klassenhierarchie
- Somit ist **Object** die einzige Klasse ohne Superklasse



Universität Bremen

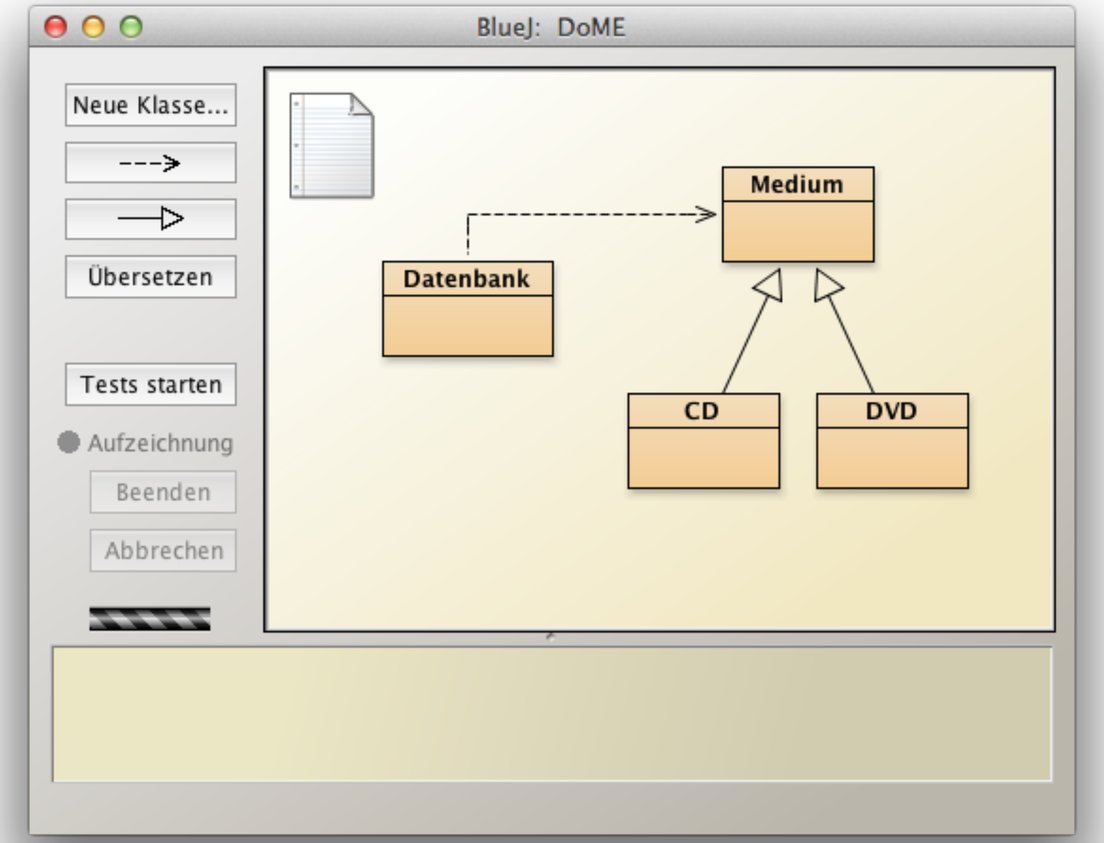
Methoden aus **Object**: **toString**

- Methoden von **Object** sind in allen Klassen verfügbar (und sind alle **public**)
- Ohne Überschreibung sind einige davon aber selten sinnvoll einsetzbar
- Beispiel: **toString**
  - Soll eine Repräsentation des Objektzustands als Zeichenkette liefern
  - Implementierung in **Object** liefert lediglich "Klassenname@hashCode"

```
class CD extends Medium
{
                :
    public String toString()
    {
        return super.toString() +
                künstler + "\n"
                titelanzahl + " Titel\n";
    }
}
```

Universität Bremen

DoME mit **toString**: Demo



Universität Bremen

Methoden aus **Object**: **equals**

- **==** prüft Referenzgleichheit
  - **obj1 == obj2** gdw. **obj1** dasselbe Objekt ist wie **obj2**
- **equals** soll Inhaltsgleichheit prüfen
  - **obj1.equals(obj2)** gdw. **obj1** den gleichen Zustand hat wie **obj2**
  - Auch muss gelten: **obj1.equals(obj2)** gdw. **obj2.equals(obj1)**
  - Muss dazu überschrieben werden, denn die Version in **Object** prüft auch nur Referenzgleichheit

Universität Bremen

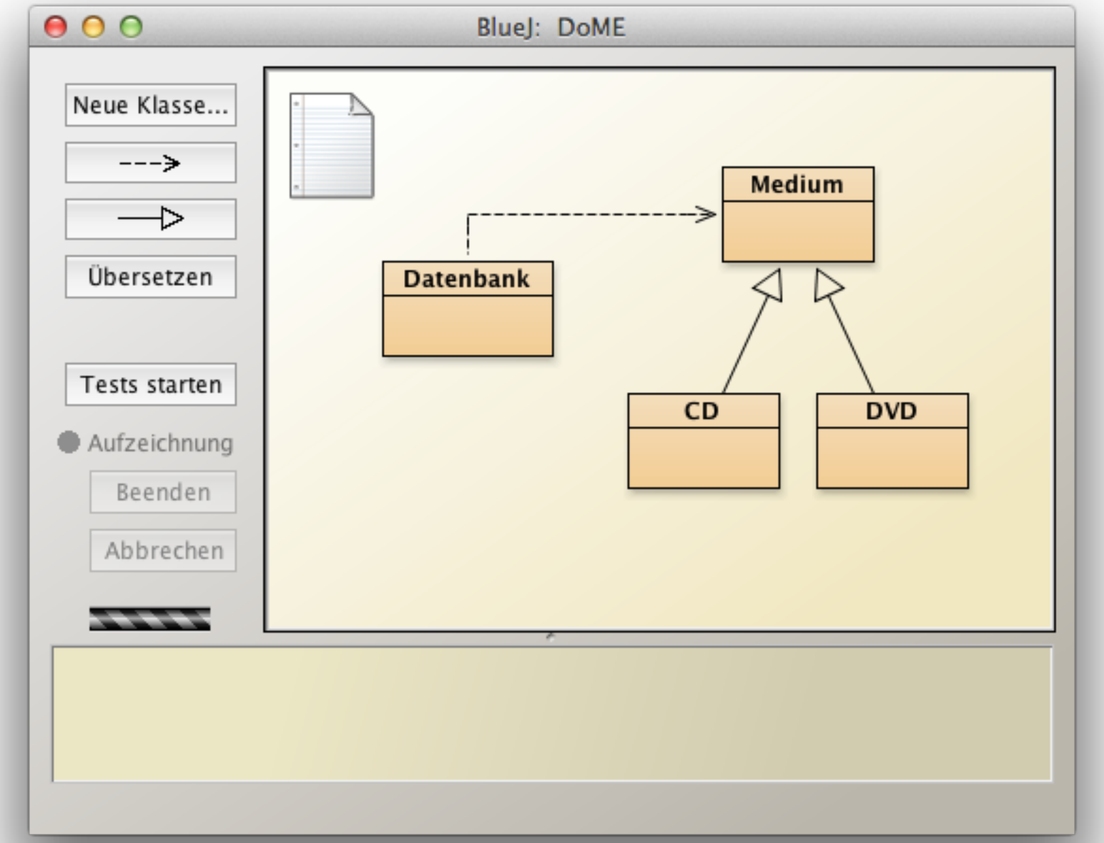
Inhaltsgleichheit

- Gleich, wenn alle Attribute gleich sind
- **==** für primitive Datentypen
- **equals** für Referenzen (**null** beachten)
- **super.equals** aufrufen (nur, wenn auch implementiert)
- Der **hashCode** eines Objekts muss aus denselben Attributen berechnet werden, die auch für **equals** verwendet werden (→ 2. Semester)

Für garantierte Kommutativität:
&& obj.getClass() == getClass()

```
public boolean equals(final Object obj)
{
    return this == obj
            || obj instanceof CD andere
                && super.equals(andere)
                && titelanzahl == andere.titelanzahl
                && (künstler == null
                    ? andere.künstler == null
                    : künstler.equals(andere.künstler));
}
```

Abstrakte Klasse: Demo



Universität Bremen

Abstrakte Methoden

- Die Definition einer **abstrakten Methode** beginnt mit **abstract** und hat keinen Rumpf (nur **;**)
- Ist eine **Verpflichtung**, die Methode in einer Subklasse zu implementieren
- Ist eine **Zusicherung**, dass diese Methode in Instanzen dieses Typs vorhanden sein wird
- Enthält eine Klasse eine **abstrakte Methode**, muss diese Klasse auch abstrakt sein

```
abstract void gibBeschreibung();
```

Universität Bremen

Abstrakte Klassen

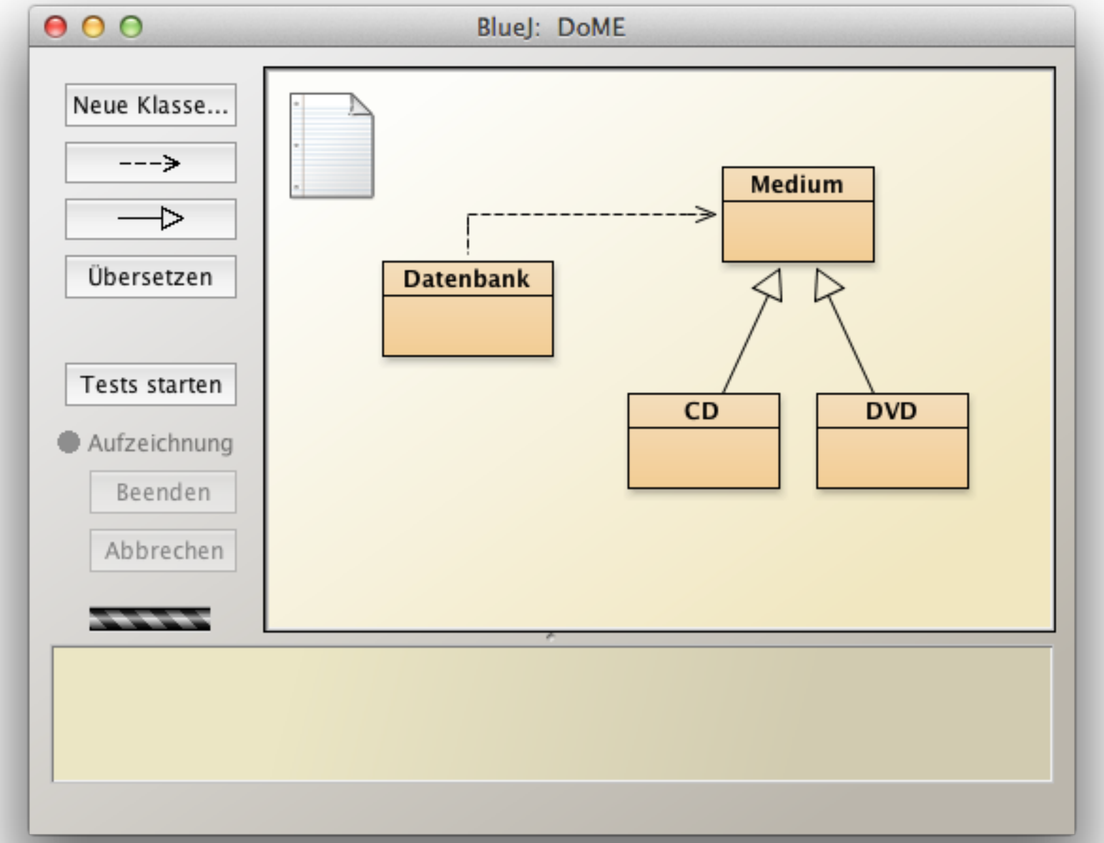
```
abstract class Medium
{
                    :
```

- Klassen, von denen keine Instanzen erzeugt werden können
- Dienen ausschließlich als Superklassen für andere Klassen
- **Abstrakte Klassen** dürfen **abstrakte Methoden** anbieten
- Solange eine Subklasse einer abstrakten Klasse nicht alle abstrakten Methoden implementiert, bleibt sie abstrakt
- Variablen können vom Typ einer abstrakten Klasse sein

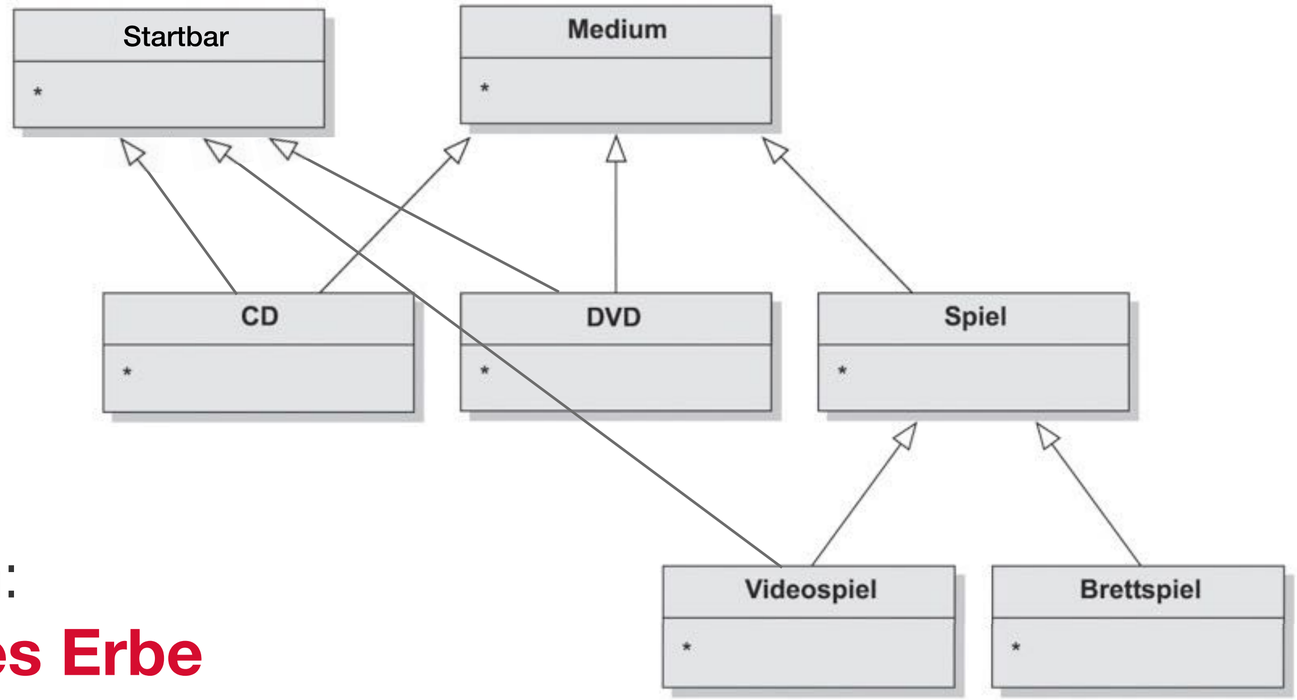
```
for (final Medium medium : medien) {
```

Universität Bremen

Schnittstelle: Demo



Multiple Vererbung



- Eine Klasse erbt von mehr als einer Superklasse
- Falsch übersetzt aus dem Englischen: **multiple inheritance → mehrfaches Erbe**
- Immer nur eine Superklasse pro Klasse in Java
- Aber eine Klasse kann mehrere Schnittstellen (**Interfaces**) implementieren

```
for (final Medium medium : medien) {
    if (medium.gibKommentar().contains(suchText)
            && medium instanceof Startbar startbar) {
        startbar.starte();
    }
}
```

Universität Bremen

Interfaces

- Im Kopf wird statt **class** das Schlüsselwort **interface** verwendet
- Sie enthalten **keine Objektattribute** und haben deshalb auch **keine Konstruktoren**
- Alles ist automatisch **public**
- Methodensignaturen sind automatisch **abstract**
- Implementierte Methoden (seit Java 8) müssen entweder **static** (Klassenmethoden) sein oder mit **default** beginnen
- Attribute sind automatisch **static** und **final**

```
interface Startbar
{
    void starte();
}
```

```
interface Ausgebbar
{
    String gibBeschreibung();

    default void ausgeben()
    {
        System.out.println(
            gibBeschreibung());
    }
}
```

Universität Bremen

Multiple Vererbung für Interfaces

- Eine Klasse kann nur eine Superklasse haben
- Sie kann aber beliebig viele Interfaces implementieren
- Dadurch ist multiple Vererbung möglich, aber es können nie **Objektattribute** von mehr als einer Klasse geerbt werden

```
class CD extends Medium implements Startbar
{ ...
```

```
class DVD extends Medium implements Ausgebbar, Startbar, Iterable<String>
{ ...
```

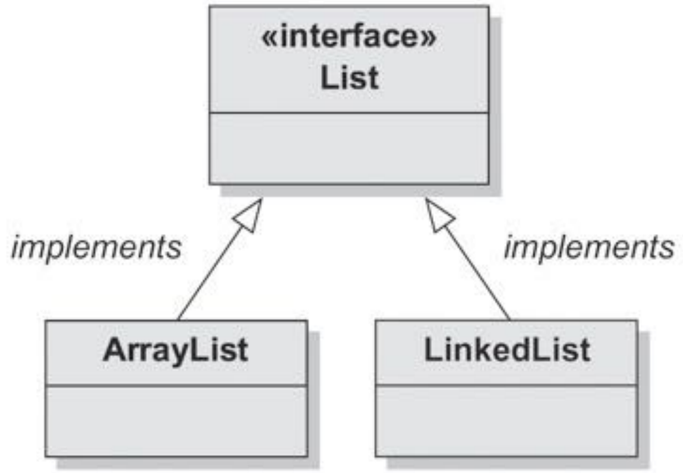
Universität Bremen

Interfaces als Typen

- Interfaces definieren Typen (wie Klassen)
- Eine Variable vom Typ eines Interfaces kann Instanzen von Klassen enthalten, die dieses Interface implementieren
- Über den statischen Typ können alle Methoden aufgerufen werden, die das Interface deklariert

```
final List<Medium> medien
        = new ArrayList<>();
                :
final List<Ausgebbar> ausgabe
        = new ArrayList<>();
for (final Medium m : medien) {
    if (m instanceof Ausgebbar a) {
        ausgabe.add(a);
    }
}
                :
for (final Ausgebbar a : ausgabe) {
    a.ausgeben();
}
```

Interfaces als Spezifikation



- Interfaces werden oft als Spezifikation benutzt, insbesondere über Paketgrenzen hinweg
  - Wie muss ein Objekt beschaffen sein, damit es von einer Methode oder Klasse verwendet werden kann?
- Verschiedene Java-Sammlungen implementieren die gleichen Schnittstellen
- Dadurch ist die Verwendung unabhängig von der Implementierung
- Z.B.: **List** (ArrayList, LinkedList), **Set** (HashSet, TreeSet), **Map** (HashMap, TreeMap)

```
List<String> notizen = new ArrayList<>();
```

```
Map<String, Raum> ausgänge = new HashMap<>();
Set<String> ausgangsnamen = ausgänge.keySet();
```

Universität Bremen

Abstrakte Klasse oder Interface?

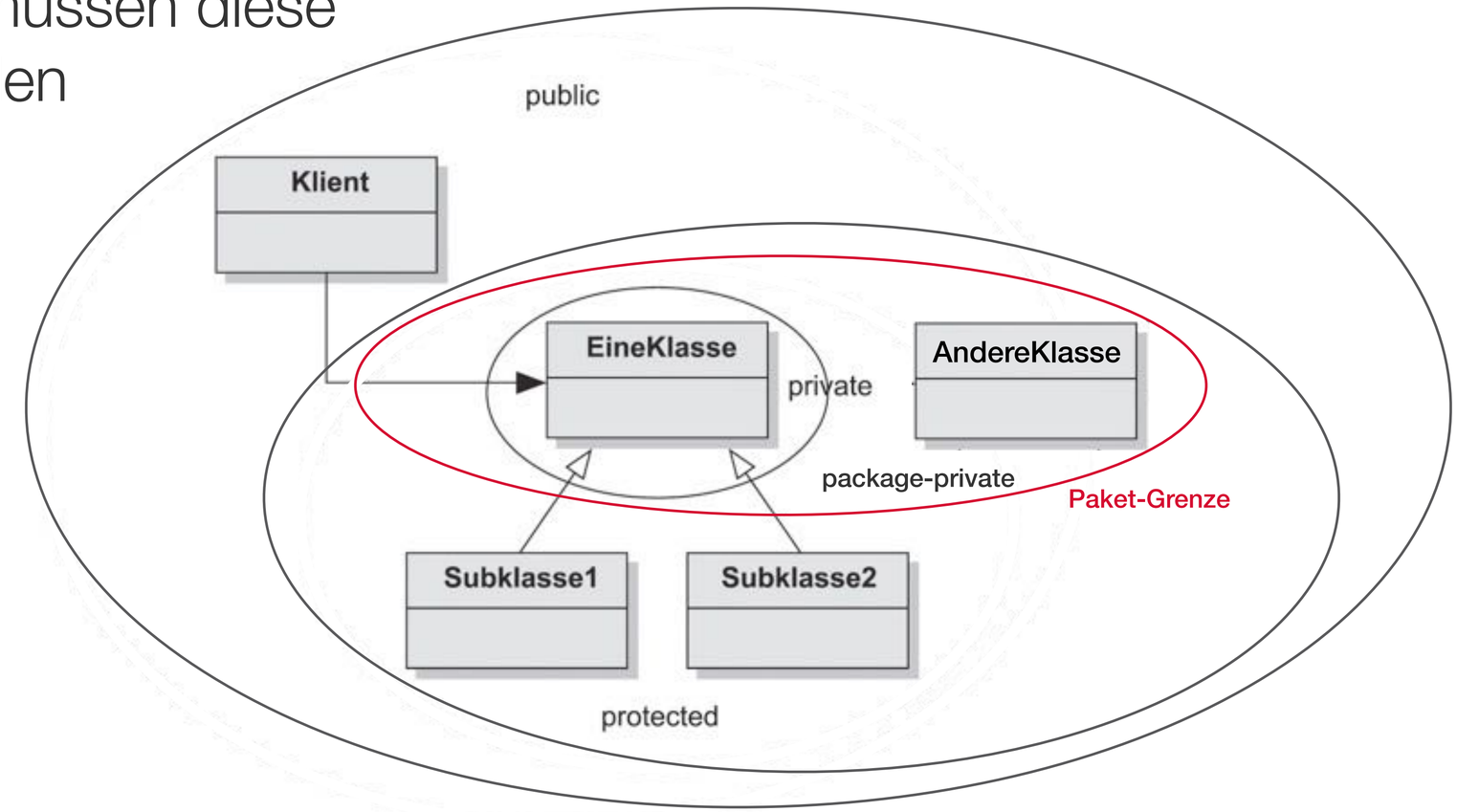
- Abstrakte Klasse
  - Sollen bereits Objektattribute definiert werden?
  - Sollen nicht alle Methoden **public** sein?
- Interface
  - In allen anderen Fällen
  - Schränkt Verwendung weniger ein (Multiple Vererbung)

Der Zugriff über **protected**

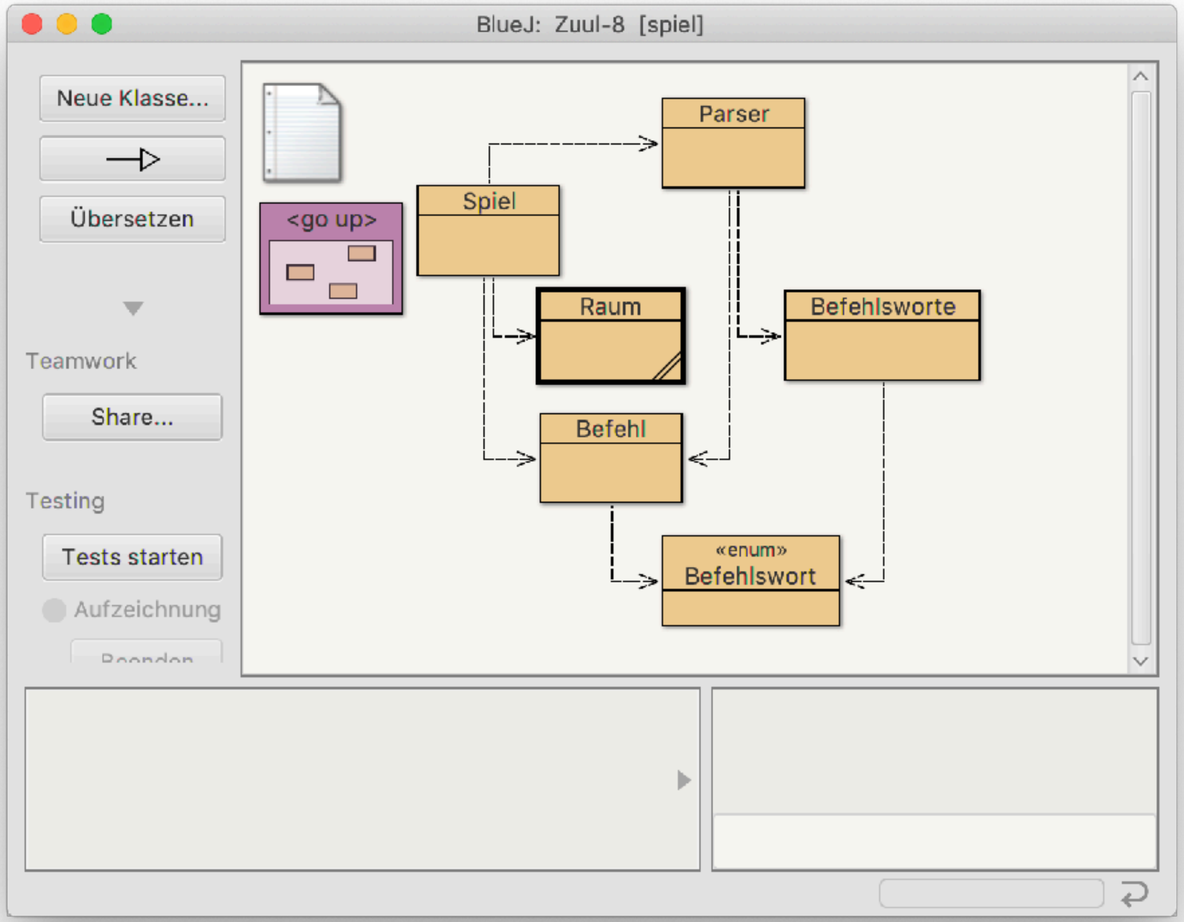
- Soll auf Methoden oder Attribute aus anderen Paketen nur von Subklassen aus zugegriffen werden können, müssen diese als **protected** deklariert werden

- Java kennt somit vier Zugriffsschutzzonen

- **protected** wird auch oft für Methoden genutzt, die überschrieben, aber nicht von außen aufgerufen werden sollen



Zuul mit Teleporter-Raum: Demo



Universität Bremen

Zusammenfassung der Konzepte

- **Typumwandlung**
- Die Klasse **Object** und die Methoden **toString** und **equals**
- **Abstrakte Methode** und **Abstrakte Klasse**
- **Multiple Vererbung** und **Interface**
- **protected**