Grundlagen der Objektorientierung

„Listenklassen“

Stand 30.05.2017

Was sind Listenklassen?

- Listenklassen verwalten einzelne Instanzen einer Klasse (Basisklasse).
- Sie bieten Such- und Zugriffs-, Erzeugungs- und Löschmethoden ebenso wie Methoden für das Laden und Speichern der gesamten Liste.

➔ **Entspricht Entity-Manager bzw. Entity-Factory für 1 Basisklasse**

Was sind Listenklassen(2)?

- Listenklassen sind frei modellierte Containerklassen, das heißt, sie müssen nicht direkt von einer bekannten Containerklasse erben
  (*Java: Vector, HashMap, HashSet, LinkedList, …*)
  bzw. keine bekannten Container-Interfaces implementieren
  (*Java: Collection, Set, List, Map, …*).

Vorteil der Vererbung / Implementierung:

Die Listenklassen können wie die „Elternklassen“ verwendet werden.

Nachteil der Vererbung / Implementierung:

- Es müssen evtl. alle (Interface-) Methoden implementiert werden.
- Die angebotenen Methoden reichen im Allgemeinen nicht aus

Methoden einer Listenklasse

Wesentliche Methoden für eine Basisklasse XYZ:

- createNewXYZ(AttributeList)
- deleteXYZ(YXZ, …)
- setXYZs(XYZs) (optional)

- getXYZ(…)
- getXYZs(…)
- getAllXYZ()

- addXYZ(XYZ) (optional)

- contains(YXZ)
- contains(KeyAttribute)

- printout()

<u>Optional, aber sinnvoll:</u>

- loadFromFile(…)
- saveToFile(…)
- loadFromDB()
- saveInDB()

Methoden einer Listenklasse(2)

Beispiel Filmverwaltung, Listenklasse FilmListe

- createNewFilm(String filmName,…)
- deleteFilm(Film film, …)
- setFilme(XYZs) (optional)

- getFilm(int filmID)
- getFilm(String filmName)
- getFilme(Darsteller darsteller)
- getFilme(Regisseur regisseur)
- getFilme(String wildCardName)
- getAlleFilme()

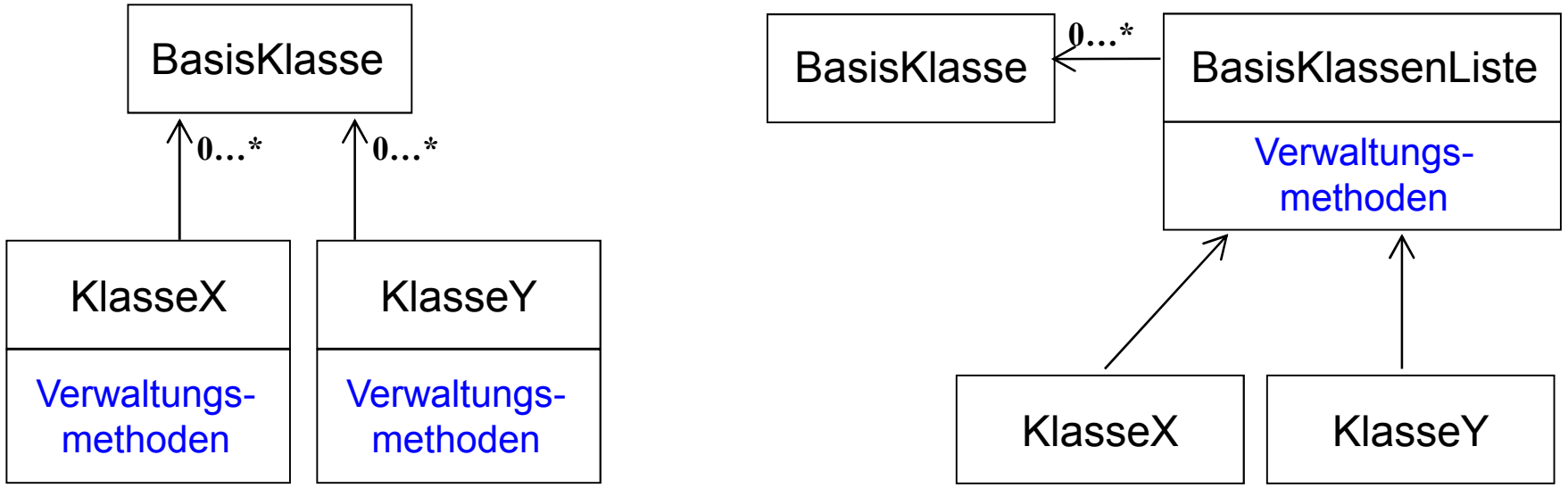
- contains(Film film)

- printout()

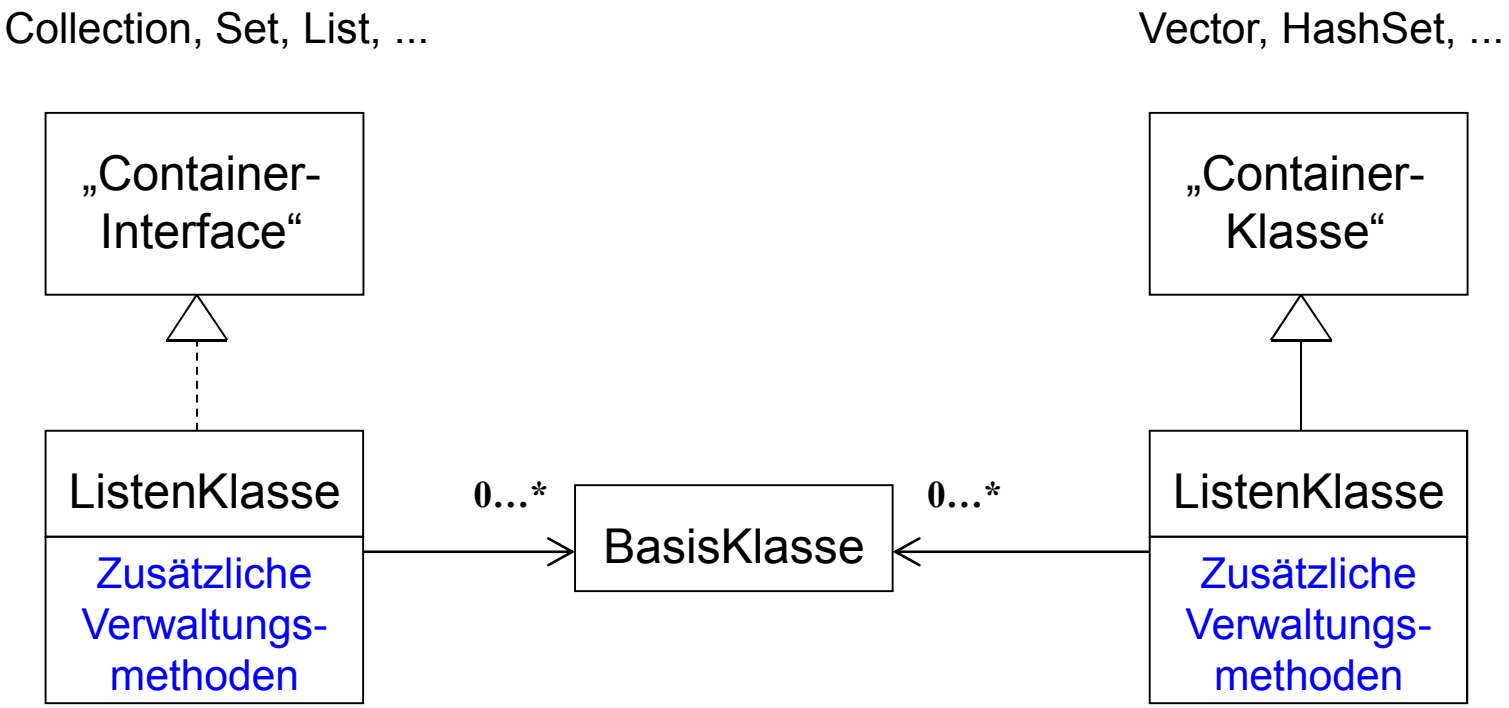
- loadFromFile(String fName, …)
- saveToFile(String fName)
- loadFromDB()
- saveInDB()

Vorteile von Listenklassen

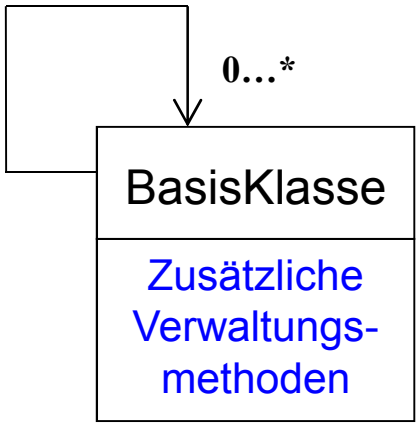
- Der gesamte „Overhead“ der Verwaltung der Basisklassen ist zentral vorhanden.
- Die Listenklassen können überall dort verwendet werden, wo die Basisklassen mehrfach referenziert werden (0...*, 1…*, N>1).



Realisierungsvarianten (1)

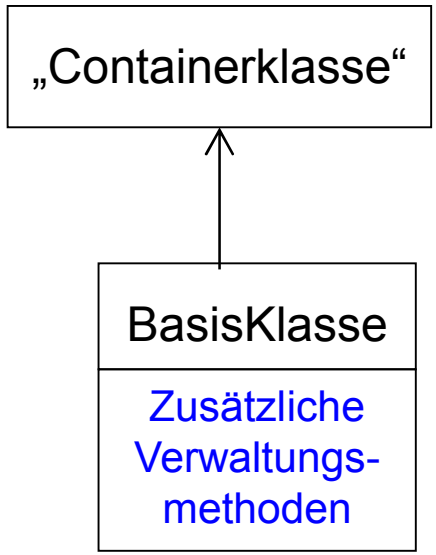


Realisierungsvarianten (2)



realisiert durch:

Vector, HashSet, ...



Implementierungsbeispiel:

```
public class BasisKlasse{
   …
   private static Vector<BasisKlasse> alleBasisKlassen;

   private BasisKlasse( Attributliste  ){ ... } // Konstruktor

   public static BasisKlasse getBasisKlasse( Attributliste  ){
      // häufig auch getInstance( Attributliste )verwendet
      wenn "BasisKlasse für gegebene Attribute vorhanden"
      dann
           diese BasisKlasse zurückgeben
      sonst
           neue BasisKlasse erzeugen, in Vector eintragen und zurückgeben
   }
   …
}
```