Konzepte der Datenbanktechnologie

Prof. Dr. U. Hoffmann
FH Wedel

Hibernate 3
Objekt–Relationales–Mapping

das letzte Mal

Entwicklungsrichtungen

- Top Down
- Bottom Up
- Middle Out
- Meet in the Middle

APIs
Mapping (Überblick)
Konfiguration und Tools

Gliederung

Object–Relational Mapping

- Collections
- N:M–Assioziationen
- Typsystem

Abbildung von Collections

- Entity–Klasse hat nicht nur einzelne Value-Type–Objekte sondern ganze Sammlungen (Collections)
- Darstellung im relationalen Modell durch eine zusätzliche Tabelle (Collection–Tabelle)
- Verwendung der Collection–Interfaces (Set, Map, Collection, List, ...) als Attribute im Objekt–Modell:

```
private <<Interface>> a = new <<Implementation>>();
...
// Getter and setter methods
```

```
private Set<String> a = new HashSet<String>();
...
// Getter and setter methods
```

Abbildung von Mengen

Beispiel (Artikel hat Menge von Bildernamen)

```
class Item {

  private int id;
  private String name;

  private Set images = new HashSet();
  ...

  public Set getImages() {
      return this.images;
  }
  public void setImages(Set images) {
      this.images = images;
  }
  ...
}
```

Abbildung von Mengen

Beispiel (Artikel hat Menge von Bildernamen)

Mapping:

```
<class name="Item" table="ITEM">
...
<set name="images" table="ITEM_IMAGE">
    <key column="ITEM_ID"/>
    <element type="string" column="FILENAME"
             not-null="true"/>
</set>
...
</class>
```

Tabellenstruktur und Beispieldaten:

ITEM

ITEM_ID	NAME
1	Foo
2	Bar
3	Baz

ITEM_IMAGE

ITEM_ID	FILENAME
1	fooimage1.jpg
1	fooimage2.jpg
2	barimage1.jpg

Abbildung von Multimengen (Bags)

- Java hat standardmäßig keine Implementierung für Bags (Multimengen).
- Interface `java.util.collection` hat aber Multimengen–Semantik (Reihenfolge nicht erhalten, Element können mehrfach vorkommen).
- Spezielle Hibernate–Unterstützung für Kombination `Collection a = new ArrayList();` und `<idbag>`

Beispiel (Artikel hat MultiMenge von Bildernamen)

```
private Collection images = new ArrayList();
...
public Collection getImages() {
    return this.images;
}
public void setImages(Collection images) {
    this.images = images;
}
```

Abbildung von Multimengen (Bags)

Beispiel (Artikel hat MultiMenge von Bildernamen)

Mapping:

```
<idbag name="images" table="ITEM_IMAGE">
  <collection-id type="long" column="ITEM_IMAGE_ID">
    <generator class="sequence"/>
  </collection-id>
  <key column="ITEM_ID"/>
  <element type="string" column="FILENAME"
           not-null="true"/>
</idbag>
```

Tabellenstruktur und Beispieldaten:

ITEM

ITEM_ID	NAME
1	Foo
2	Bar
3	Baz

ITEM_IMAGE

ITEM_IMAGE_ID	ITEM_ID	FILENAME
1	1	fooimage1.jpg
2	1	fooimage1.jpg
3	3	barimage1.jpg

Abbildung von Listen

- Reihenfolge erhalten: Listen

Beispiel (Artikel hat Liste von Bildernamen)

```
private List images = new ArrayList();
...
public List getImages() {
    return this.images;
}
public void setImages(List images) {
    this.images = images;
}
```

Abbildung von Listen

Beispiel (Artikel hat Liste von Bildernamen)

Mapping:

```
<list name="images" table="ITEM_IMAGE">
    <key column="ITEM_ID"/>
    <list-index column="POSITION"/>
    <element type="string" column="FILENAME"
             not-null="true"/>
</list>
```

Tabellenstruktur und Beispieldaten:

ITEM

ITEM_ID	NAME
1	Foo
2	Bar
3	Baz

ITEM_IMAGE

ITEM_ID	POSITION	FILENAME
1	0	fooimage1.jpg
1	1	fooimage2.jpg
1	2	foomage3.jpg

Abbildung von Java–Maps

- Zugriff nicht über Index sondern über Namen: Java–Maps

Beispiel (Artikel hat Map)

```
private Map images = new HashMap();
...
public Map getImages() {
    return this.images;
}
public void setImages(Map images) {
    this.images = images;
}
```

Abbildung von Java–Maps

Beispiel (Artikel hat Map)

Mapping:

```
<map name="images" table="ITEM_IMAGE">
    <key column="ITEM_ID"/>
    <map-key column="IMAGENAME" type="string"/>
    <element type="string" column="FILENAME"
             not-null="true"/>
</map>
```

Tabellenstruktur und Beispieldaten:

ITEM

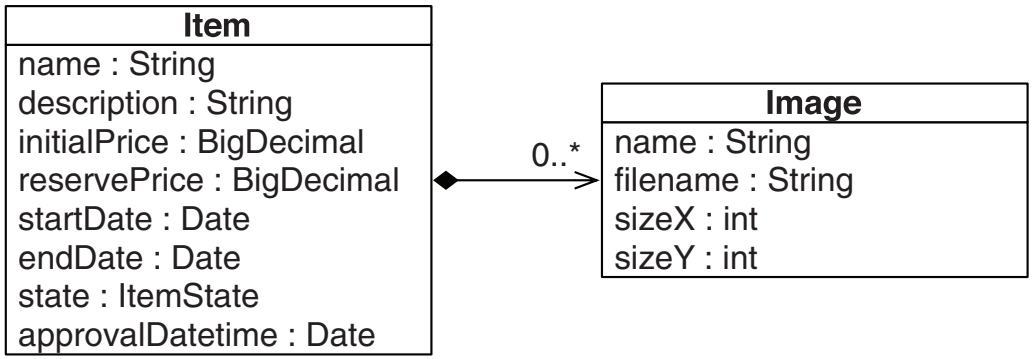
ITEM_ID	NAME
1	Foo
2	Bar
3	Baz

ITEM_IMAGE

ITEM_ID	IMAGENAME	FILENAME
1	Image One	fooimage1.jpg
1	Image Two	fooimage2.jpg
1	Image Three	foomage3.jpg

Collections von Komponenten

- Ganze Strukturen in Collections ablegen



- Image nicht nur durch den Filenamen sondern auch durch zusätzliche Attribute beschrieben.
- Implementierung der Komponentenklasse als POJO (Muss `equals()` und `hashCode()`) definieren.

Collections von Komponenten

Beispiel (Artikel hat Menge von Komponenten)

```
<set name="images" table="ITEM_IMAGE">
 <key column="ITEM_ID"/>
 <composite-element class="Image">
   <property name="name" column="IMAGENAME"
             not-null="true"/>
   <property name="filename" column="FILENAME"
             not-null="true"/>
   <property name="sizeX" column="SIZEX"
             not-null="true"/>
   <property name="sizeY" column="SIZEY"
             not-null="true"/>
 </composite-element>
</set>
```

ITEM

ITEM_ID	ITEM_NAME
1	Foo
2	Bar
3	Baz

ITEM_IMAGE

ITEM_ID	IMAGENAME	FILENAME	SIZEX	SIZEY
1	Foo	Foo.jpg	123	123
1	Bar	Bar.jpg	420	80
2	Baz	Baz.jpg	50	60

Collections von Komponenten

- Null–Werte in den Attributen der Komponente: Multimenge und Mapping via `<idbag>`

Beispiel (Multimenge von Komponenten)

```
private Collection images = new ArrayList();
...
public Collection getImages() {
    return this.images;
}
public void setImages(Collection images) {
    this.images = images;
}
```

Collections von Komponenten

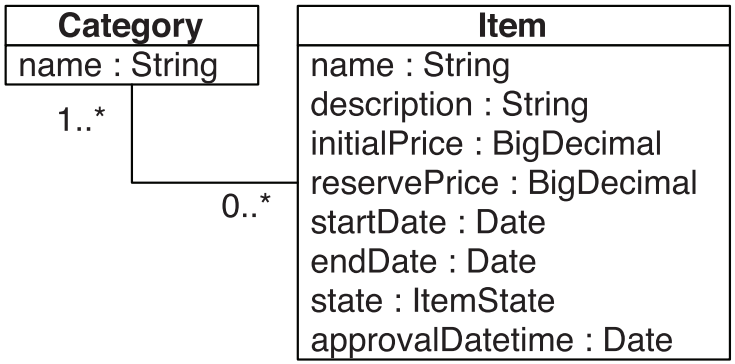
Beispiel (Multimenge von Komponenten)

```
<idbag name="images" table="ITEM_IMAGE">
  <collection-id type="long" column="ITEM_IMAGE_ID">
    <generator class="sequence"/>
  </collection-id>
  <key column="ITEM_ID"/>
  <composite-element class="Image">
    <property name="name" column="IMAGENAME"/>
    <property name="filename" column="FILENAME"
              not-null="true"/>
    <property name="sizeX" column="SIZEX"/>
    <property name="sizeY" column="SIZEY"/>
  </composite-element>
</idbag>
```

ITEM_IMAGE

ITEM_IMAGE_ID	ITEM_ID	IMAGENAME	FILENAME	SIZEX	SIZEY
1	1	Foo	Foo.jpg	123	123
2	1	Bar	Bar.jpg	420	80
3	2	Baz	Baz.jpg	NULL	NULL

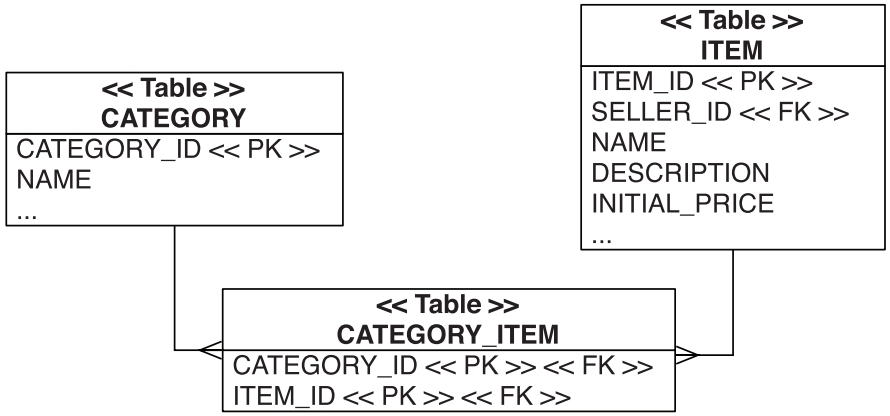
N:M–Assioziationen



- Häufig deuten N:M–Assoziationen auf Modellierungsprobleme hin.
- In der Praxis oft bessere Alternative:
  - Zwischenklasse zur Darstellung der Assoziation
  - Zwei 1:N–Assoziationen zur Zwischenklasse

- Hier: Mapping *echter* N:M–Beziehnungen

Einfache unidirektionale N:M–Assoziation



Beispiel (unidirektionale N:M–Assoziation)

```
<class name="Category" table="CATEGORY">
...
<set name="items"
     table="CATEGORY_ITEM"
     cascade="save-update">
    <key column="CATEGORY_ID"/>
    <many-to-many class="Item" column="ITEM_ID"/>
</set>
...
</class>
```

- Zugriff: `aCategory.getItems().add(anItem)`

Einfache unidirektionale N:M–Assoziation

Alternatives Mapping:

- Mit `<idbag>`
- generiert zusätzliche Primärschlüssel–Spalte

Beispiel (unidirektionale N:M–Assoziation)

```
<idbag name="items"
       table="CATEGORY_ITEM"
       cascade="save-update">
  <collection-id type="long"
                 column="CATEGORY_ITEM_ID">
    <generator class="sequence"/>
  </collection-id>
  <key column="CATEGORY_ID"/>
  <many-to-many class="Item" column="ITEM_ID"/>
</idbag>
```

- Zugriff bleibt identisch:
  `aCategory.getItems().add(anItem)`

bidirektionale N:M–Assoziation

Beispiel (bidirektionale N:M–Assoziation Teil 1)

```
<class name="Category" table="CATEGORY">
 ...
 <set name="items"
      table="CATEGORY_ITEM"
      cascade="save-update">
      <key column="CATEGORY_ID"/>
      <many-to-many class="Item" column="ITEM_ID"/>
 </set>
```

- Zugriff:
  ```
  aCategory.getItems().add(anItem);
  anItem.getCategories().add(aCategory);
  ```

bidirektionale N:M–Assoziation

Beispiel (bidirektionale N:M–Assoziation Teil 2)

```
<class name="Item" table="ITEM">
 ...
 <set name="categories"
      table="CATEGORY_ITEM"
      inverse="true"
      cascade="save-update">
   <key column="ITEM_ID"/>
   <many-to-many class="Category" column="CATEGORY_ID"/>
 </set>
</class>
```

Abbildung der primitiven Java Typen

Mapping type	Java type	Standard SQL built-in type
integer	int oder java.lang.Integer	INTEGER
long	long oder java.lang.Long	BIGINT
short	short oder java.lang.Short	SMALLINT
float	float oder java.lang.Float	FLOAT
double	double oder java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte oder java.lang.Byte	TINYINT
boolean	boolean oder java.lang.Boolean	BIT
yes_no	boolean oder java.lang.Boolean	CHAR(1) ('Y' oder 'N')
true_false	boolean oder java.lang.Boolean	CHAR(1) ('T' oder 'F')
date	java.util.Date oder java.sql.Date	DATE
time	java.util.Date oder java.sql.Time	TIME
timestamp	java.util.Date oder java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE
binary	byte[]	VARBINARY
text	java.lang.String	CLOB
clob	java.sql.Clob	CLOB
blob	java.sql.Blob	BLOB
serializable	Jede Klasse, die java.io.Serializable implementiert	VARBINARY
class	java.lang.Class	VARCHAR
locale	java.util.Locale	VARCHAR
timezone	java.util.TimeZone	VARCHAR
currency	java.util.Currency	VARCHAR