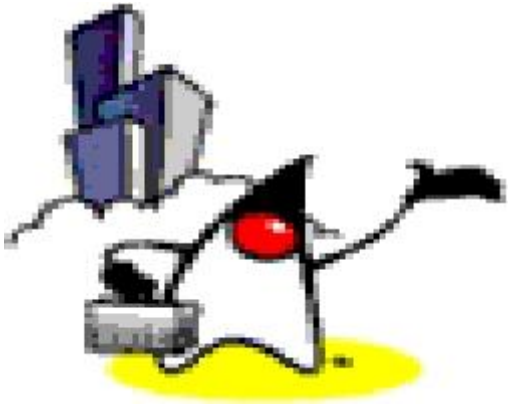
Mapping
Cardinality & Inheritance
Relationships

Contents

- Mapping cardinality relationship
  - One-To-One
  - One-To-Many
  - Many-To-Many
  - One-has-Map
- Mapping inheritance relationship
  - One table per each concrete class implementation
  - One table for each subclass
  - One table for each class hierarchy

Mapping Cardinality Relationship



Mapping Cardinality Relationships

- one-to-one
- many-to-one
- one-to-many
- many-to-many

One to One Relationship

- Expresses a relationship between two classes where each instance of the first class is related to a single instance of the second or vice versa
- Can be expressed in the database in two ways
  - Giving each of the respective tables the same primary key values
  - Using foreign key constraint from one table onto a unique identifier column of the other

One-To-Many Relationship

- <set>
- <list>
- <array>
- <bag>

One-to-Many relationship: Using <set> in mapping file

- An event has many speakers and attendees
- *Event.hbm.xml*

```
<class name="Event" table="events">
<id name="id" column="uid" type="long" unsaved-value="null">
<generator class="increment"/>
</id>
<property name="name" type="string" length="100"/>


<set name="speakers" cascade="all">
    <key column="event_id"/>
    <one-to-many class="Speaker"/>
</set>
<set name="attendees" cascade="all">
    <key column="event_id"/>
    <one-to-many class="Attendee"/>
</set>
</class>
```

One to Many relationship: Using Set in Domain Class

- An event has many speakers and attendees

```
public class Event {
        private Long id;
        private String name;
        private Date startDate;
        private int duration;
        // Event has one-to-many relationship with Speaker
        private Set speakers;

        // Event has one-to-many relationship with Attendee
        private Set attendees;

        // ...
```

One to Many relationship: Creating Object Instance

- An event has many speakers and attendees

```
// Create an Event object which has one to many relationship
 // with Speaker objects.

Event event = new Event();
event.setName("Java Conference");
event.setSpeakers(new HashSet());
event.getSpeakers().add(new Speaker("Sang", "Shin"));
event.getSpeakers().add(new Speaker("Dave", "Smith"));
event.getSpeakers().add(new Speaker("Bill", "Clinton"));

session.saveOrUpdate(event);
```

2. One to Many relationship: Using <list> in mapping file

- Group has many stories
- *Group.hbm.xml*

```
<class name="Group" table="grouptable">
        <id name="id" unsaved-value="0">
                <generator class="increment" />
        </id>

        <list name="stories" cascade="all">
                <key column="parent_id" />
                <!-- index in a single list -->
                <index column="idx" />
                <one-to-many class="Story" />
        </list>
        <property name="name" type="string" />
</class>
```

One to Many relationship: Using List in Domain Class

- Group has many stories

```
public class Group {

    private int id;
    private String name;

    private List stories;

    public void setStories(List l) {
        stories = l;
    }

    public List getStories() {
        return stories;
    }
    // ...
```

One to Many relationship: Creating Object Instances

- Group has many stories

```
ArrayList list = new ArrayList();
list.add(new Story("Tom Jones"));
list.add(new Story("Beatles"));
list.add(new Story("Elvis"));

Group sp = new Group("Singers");
sp.setStories(list);

ArrayList list2 = new ArrayList();
list2.add(new Story("Bill Clinton"));
list2.add(new Story("Ronald Reagan"));

Group sp2 = new Group("Politicians");
sp2.setStories(list2);
```

One to Many relationship: Using <list> in the mapping file

- Tables

********** Table: grouptable *********

ID	NAME
1	Singers
2	Politicians

********** Table: story *********

ID	INFO	IDX	PARENT_ID
1	Tom Jones	0	1
2	Beatles	1	1
3	Elvis	2	1
4	Bill Clinton	0	2
5	Ronald Reagan	1	2

3. One to Many relationship: Using <array> in mapping file

- Group has many stories
- *Group.hbm.xml*

```
<class name="Group" table="grouptable">
        <id name="id" unsaved-value="0">
                <generator class="increment"/>
        </id>

        <array name="stories" cascade="all">
                <key column="parent_id"/>
                <index column="idx"/>
                <one-to-many class="Story"/>
        </array>
        <property name="name" type="string"/>
</class>
```

One to Many relationship: Using an array in Domain Class

- Group has many stories

```
public class Group {

    private int id;
    private String name;

    // Group object has an array of Story objects
    private Story[] stories;

    public void setStories(Story[] l) {
        stories = l;
    }

    public Story[] getStories() {
        return stories;
    }

    // ...
```

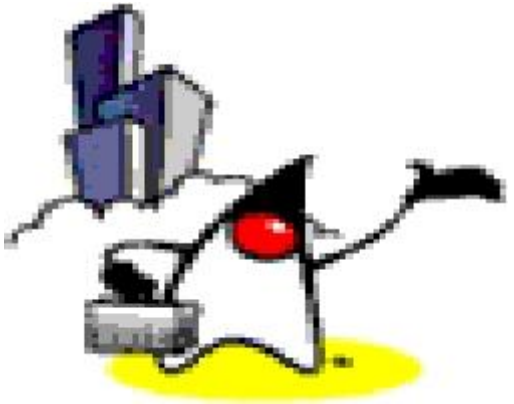
One to Many relationship: Creating an Object Instance

- Group has many stories

```
// Create an Group object which has one to many relationship
// with Story objects.

Group sp = new Group("Group Name");
sp.setStories(new Story[]{new Story("Story Name 1"), new
   Story("Story Name 2")});
```

One-To-Many:
Using <bag>



4. One to Many relationship: Using <bag> in mapping file

- Group has many stories
- *Group.hbm.xml*

```
<class name="Group" table="grouptable">

        <id name="id" unsaved-value="0">
            <generator class="increment"/>
        </id>

        <bag name="stories" cascade="all">
            <key column="parent_id"/>
            <one-to-many class="Story"/>
        </bag>

        <property name="name" type="string"/>

</class>
```

One to Many relationship: Using an List in Domain Class

- Group has many stories

```
public class Group {

        private int id;
        private String name;
        private List stories;

        public void setStories(List l) {
                stories = l;
        }

        public List getStories() {
                return stories;
        }

        // ...
```

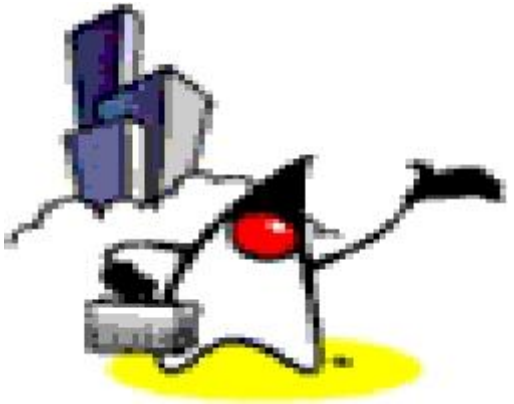
One to Many relationship: Creating an Object Instance

- Group has many stories

```
// Create an Group object which has one to many relationship
// with Story objects.

ArrayList list = new ArrayList();
list.add(new Story("Story Name 1"));
list.add(new Story("Story Name 2"));
Group sp = new Group("Group Name");
sp.setStories(list);
```

Mapping Cardinality Relationship:
Many-To-Many



Many to Many relationship

- Speakers speak in many events and Event has many speakers
- *SpeakerManyToMany.hbm.xml*

```
<class name="SpeakerManyToMany" table="m_speakers">
        <id name="id" column="uid" type="long">
            <generator class="increment"/>
        </id>
        <property name="firstName" type="string" length="20"/>
        <property name="lastName" type="string" length="20"/>


        <set name="events" table="event_speakers" cascade="all">
            <key column="speaker_id"/>
            <many-to-many class="EventManyToMany"/>
        </set>


</class>
```

Many to Many relationship

- Event has many speakers and speakers speak in many events
- *EventManyToMany.hbm.xml*

```
<class name="EventManyToMany" table="m_events">
<id name="id" column="uid" type="long" unsaved-value="null">
<generator class="increment"/>
</id>
<property name="name" type="string" length="100"/>
<property name="startDate" column="start_date" type="date"/>
<property name="duration" type="integer"/>

<set name="speakers" table="event_speakers" cascade="all">
    <key column="event_id"/>
    <many-to-many class="SpeakerManyToMany"/>
</set>
</class>
```

- Event has many speakers

```
public class EventManyToMany {

    private    Long id;
    private    String name;
    private    Date startDate;
    private    int duration;
    private    Set speakers;
    private    Set attendees;

    public void setSpeakers(Set speakers) {
        this.speakers = speakers;
    }

    public Set getSpeakers() {
        return speakers;
    }

    // ...
```

Many to Many relationship:

- A speaker speaks in many events

```
public class SpeakerManyToMany {
        private Long id;
        private String firstName;
        private String lastName;
        private Set events;
        public Set getEvents() {
                return this.events;
        }

        public void setEvents(Set events) {
                this.events = events;
        }
        // ...
```

Many to Many relationship: Creating object instances

- Event has many to many relationship with Speaker

```
// Event has Many-To-Many relationship with Speaker
EventManyToMany event = new EventManyToMany();
event.setName("JavaOne conference");
event.setSpeakers(new HashSet());
event.getSpeakers().add(new SpeakerManyToMany("Sang",
    "Shin", event));
event.getSpeakers().add(new SpeakerManyToMany("Joe",
    "Smith", event));
event.getSpeakers().add(new SpeakerManyToMany("x",
    "Man", event));
// Save event
session.save(event);
```

Many to Many relationship

******** Table: m_events *******

UID	NAME	START_DATE	DURATION	LOCATION_ID
1	JavaOne conference		0	
2	Passion Conference		0	

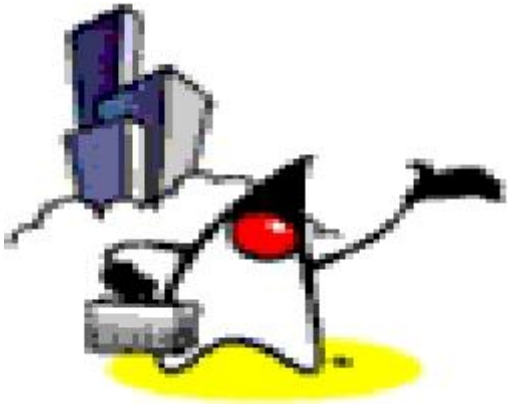
******** Table: m_speakers *******

UID	FIRSTNAME	LASTNAME
1	Joe	Smith
2	John	Smith
3	Sang	Shin
4	Sang	Shin
5	Diane	Woon
6	Shelly	Lumm

********** Table: event_speakers *********

ELT	EVENT_ID	SPEAKER_ID
1	1	
2	1	
3	1	
1		1
1		2
1		3
4	2	
5	2	
6	2	
2		4
2		5
2	6	

Mapping Cardinality Relationship: Using <map>



One-Has-Collection relationship: Using <map> in mapping file

- SupportProperty class has Collection
- *SupportProperty.hbm.xml*

```
<class name="SupportProperty" table="supportproperty">
        <id name="id">
                <generator class="increment"/>
        </id>


        <map name="properties">
                <key column="id"/>
                <index column="property_name" type="string"/>
                <element column="property_value" type="string"/>
        </map>


        <property name="name" type="string"/>
</class>
```

One-Has-Collection relationship: Domain Class

- Group has many stories

```
public class SupportProperty {
    private int id;
    private String name;
    private Map properties;

    public void setProperties(Map m) {
        properties = m;
    }

    public Map getProperties() {
        return properties;
    }
    // ...
```

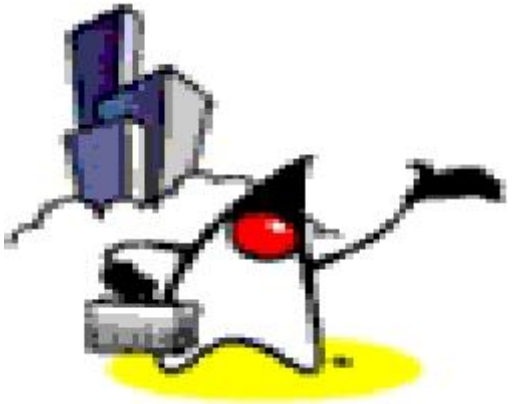
One-Has-Collection relationship: Creating an Object Instance

- Group has many stories

```
// Create Domain object, SupportProperty object has
   a Map
// object.
SupportProperty sp = new SupportProperty();
sp.setName("Joe");

HashMap p = new HashMap();
p.put("color", "blue");
p.put("lnf", "mac");
sp.setProperties(p);
```

Mapping Inheritance:



Inheritance Relationship Representations

- 3 different ways
  - One table for each class hierarchy
  - One table for each subclass
  - One table per each concrete class implementation
- Each of these techniques has different costs and benefits

One Table per Class Hierarchy

- A single table for the whole class hierarchy
- Discriminator column contains key to identify the base type
- Advantages
  - Offers best performance even for in the deep hierarchy since single select may suffice
- Disadvantages
  - Changes to members of the hierarchy require column to be altered, added or removed from the table

One Table per Class Hierarchy

- How to define the mapping
  - Use <*subclass*> element with *extends* and *discriminator-value* attributes

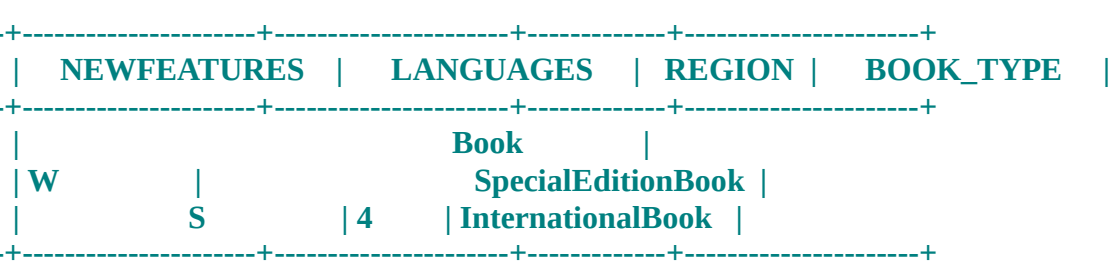
One Table per Class Hierarchy

```
<hibernate-mapping>
<subclass name="SpecialEditionBook"
extends="Book"

discriminator-value="SpecialEditionBook">

<property name="newfeatures" type="string"
   />
</subclass>
</hibernate-mapping>
```

One Table per Class Hierarchy



One Table per Subclass

- One table for each class in the hierarchy
  - Foreign key relationship exists between common table and subclass tables
- Advantages
  - Does not require complex changes to the schema when a single parent class is modified
  - Works well with shallow hierarchy
- Can result in poor performance – as hierarchy grows, the number of joins required to construct a leaf class also grows

One Table per Subclass

- How to define the mapping
  - Use *<joined-subclass>* element with *extends* attribute in the mapping file of the subclass

Example: One Table per Subclass

```
<hibernate-mapping>

    <joined-subclass name="SpecialEditionBook"
                                 extends="Book"
                                 table="secd">
        <key column="id" />
        <property name="newfeatures" type="string" />
    </joined-subclass>

</hibernate-mapping>
```

One Table per Subclass

********** Table: Book *********

ID	TITLE	ARTIST	PURCHASEDATE	COST
1	Book	R	2008-04-11	9.99
2	sBook	R	2008-04-11	9.99
3	IBook	R	2008-04-11	9.99

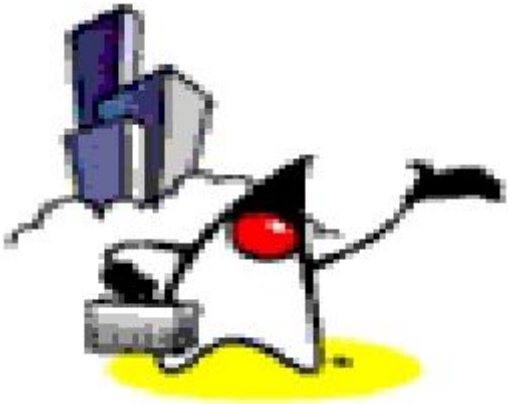
********** Table: SpecialEditionBook *********

ID	NEWFEATURES
2	W

********** Table: InternationalBook *********

ID	LANGUAGES	REGION
3	S	4

Mapping Inheritance:
1 Table for Concrete Class



One Table per Concrete Class

- Map each of the concrete classes as normal persistent class
- Pros
  - Easiest to implement
- Cons
  - Data belonging to a parent class is scattered across a number of different tables, which represent concrete classes
  - A query in terms of parent class is likely to cause a large number of select operations
  - Changes to a parent class can touch large number of tables
  - This scheme is not recommended for most cases

One Table per Concrete Class

- How to define the mapping
  - The mapping of the subclass repeats the properties of the parent class

One Table per Concrete Class

```
<hibernate-mapping>
        <class name="Book" table="cd" discriminator-value="cd">
                <id name="id" type="integer" unsaved-value="0">
                        <generator  class="increment"/>
                </id>
                <property name="title"/>
                <property name="artist"/>
                <property name="purchasedate" type="date"/>
                <property name="cost" type="double"/>
        The mapping of the subclass repeats the properties of the parent class
        <class  name="SpecialEditionBook" table="secd">
                <id name="id" type="integer" unsaved-value="0">
                        <generator  class="increment"/>
                </id>
                <property name="title"/>
                <property name="artist"/>
                <property name="purchasedate" type="date"/>
                <property name="cost" type="double"/>
                <property name="newfeatures" type="string"/>
        </class>
</hibernate-mapping>
```

One Table per Concrete Class

```
******** Table: Book *******
+------------+--------------------+--------------------+-------------+-----------------------+
|   ID  |    TITLE    |    ARTIST    | PURCHASEDATE |    COST     |
+------------+--------------------+--------------------+-------------+-----------------------+
| 1     | Book        | R          | 2008-04-11  | 9.99        |
+------------+--------------------+--------------------+-------------+-----------------------+

******** Table: SpecialEditionBook *******
+------------+--------------------+--------------------+-------------+-----------------------+--------------------+
|   ID  |    TITLE    |    ARTIST    | PURCHASEDATE |    COST     |   NEWFEATURES  |
+------------+--------------------+--------------------+-------------+-----------------------+--------------------+
| 1     | sBook       | R          | 2008-04-11  | 9.99        | W          |
+------------+--------------------+--------------------+-------------+-----------------------+--------------------+

******** Table: InternationalBook *******
+------------+--------------------+--------------------+-------------+-----------------------+--------------------+------------+
|   ID  |    TITLE    |    ARTIST    | PURCHASEDATE |    COST     |   LANGUAGES   | REGION |
+------------+--------------------+--------------------+-------------+-----------------------+--------------------+------------+
| 1     | IBook       | R          | 2008-04-11  | 9.99        | S          | 4     |
| 2     | IBook       | R          | 2008-04-11  | 100.9        | T          | 3     |
+------------+--------------------+--------------------+-------------+-----------------------+--------------------+------------+
```

The End!