**Usage of mappedBy in Spring JPA Hibernate – 2022**

mappedBy indicates that the entity in this side is the inverse of the relationship, and the owner resides in the "other" entity. This also means that you can access the other table from the class which you've annotated with "mappedBy" (fully bidirectional relationship). **In hibernate terms,it means “I am not the owner of the relationship, ownership is governed by other entity'**.

For a bidirectional relationship, we usually define:

* the owning side
* inverse or the referencing side

The @JoinColumn annotation helps us specify the column we'll use for joining an entity association or element collection. On the other hand, the mappedBy attribute is used to define the referencing side (non-owning side) of the relationship.

Examples are given below.

In case of **One To One Bidirectional Mapping**

@Entity(name = "Child")

@Table(name = "child")

@Data

**public** **class** Child {

@Id

@GeneratedValue

**private** **long** id;

@Column(name="name")

**private** String name;

/\*

\* By writting mappedBy = child,

\* Here Parent is the owner

\* and parent table will have

\* child id as reference

\*/

@OneToOne(mappedBy = "child")

**private** Parent parent;

}

@Entity(name = "Parent")

@Table(name = "parent")

@Data

**public** **class** Parent {

@Id

@GeneratedValue

**private** **long** id;

@Column(name="name")

**private** String name;

@OneToOne(cascade = CascadeType.***ALL***)

**private** Child child;

}

**Database Table Design**

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated

\*\*\* **Here Parent is the owner**.

@Entity(name = "Company")

@Table(name = "company")

@Data

**public** **class** Company {

@Id

@GeneratedValue

**private** **long** id;

@Column(name="name")

**private** String name;

/\*

\* In this case Branch class

\* is the owning side.

\* Whenever you use mappedBy,

\* the opposite side class

\* will be the owner.

\*/

@OneToOne(cascade = CascadeType.***ALL***,

mappedBy = "company")

**private** Branch branch;

}

@Entity(name = "Branch")

@Table(name = "branch")

@Data

**public** **class** Branch {

@Id

@GeneratedValue

**private** **long** id;

@Column(name="name")

**private** String name;

@OneToOne

**private** Company company;

}

**\*\*\* Here Branch is the owner**

**Database Table Design**

Graphical user interface, table

Description automatically generated Table

Description automatically generated

**Usage of mappedBy in case of One To Many Bidirectional Mapping**

@Entity(name = "Teacher")

@Table(name = "teacher")

//@Data // Do not use @Data, it creates StackOverflow Error

@Getter

@Setter

**public** **class** Teacher {

@Id

@GeneratedValue

**private** Long id;

@Column(name = "name")

**private** String name;

@OneToMany(mappedBy = "teacher", cascade = CascadeType.***ALL***)

**private** Set<Student> students = **new** HashSet<>();

**public** **void** addStudent(Student student) {

students.add(student);

student.setTeacher(**this**);

}

**public** **void** removeStudent(Student student) {

student.setTeacher(**null**);

students.remove(student);

}

}

@Entity(name="Student")

@Table(name = "student")

@Getter @Setter

**public** **class** Student {

@Id @GeneratedValue

**private** Long id;

@Column(name = "name")

**private** String name;

@ManyToOne

**private** Teacher teacher;

}

**Database Table Design**

Graphical user interface, text, application, chat or text message

Description automatically generated Graphical user interface, text, application

Description automatically generated

**Cascade By Attribute**

Cascading is the way to achieve this. **When we perform some action on the target entity, the same action will be applied/propagated to the associated entity.**

*javax.persistence.CascadeType*enum containing entries:

* *ALL*
* *PERSIST*
* *MERGE*
* *REMOVE*
* *REFRESH*
* *DETACH*

**Cascade By attribute provided by Hibernate keeps the above mechanism abstract to the developer. Developer persists/updates/deletes the top level object and changes are propagated to the associated entities depending upon the value of cascade by attribute**.

So by providing the right cascade attributes we can govern the flow of changes from parent to child entities.

class BookLibrary{  
      Long libraryId;  
      String libraryName;  
       …...  
    **@Cascade(value = { CascadeType.ALL })**       Set<Book> bookSet;  
}

### Difference between Unidirectional and Bidirectional mapping in hibernate

**Unidirectional Mapping**

**When only one of the pair of entities contains a reference to the other, the association is unidirectional**. for example, assume that there are two entities called Teacher and Course.

In unidirectional mapping, the teacher will hold a reference for the course OR course will hold a reference for the teacher.(Not Both) It is mandatory that the references should not be mapped to the both directions and there should be only one direction(unidirectional) mapping.

**In unidirectional mapping, it will provide the navigational access only to one direction.**

assumption: teacher can teach only once course and a given course can be taught only by one teacher.

**Bidirectional Mapping**

**if the association between both entities are mutual**, then it is known as bidirectional mapping. in bidirectional mapping, the lecturer should hold to a reference for the course and the same time the course should a reference to the lecturer.

**therefore in bi-directional mapping, it will provide the navigational access to the both directions.**

assumption: teacher can teach only once course and a given course can be taught only by one teacher.