Hibernate Mappings

Three most important mappings are:

* Mapping of collections,
* Mapping of associations between entity classes, and
* Component Mappings.

Collections Mappings

If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java. Hibernate can persist instances of **java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List**, and any **array** of persistent entities or values.

|  |  |
| --- | --- |
| **Sr.No.** | **Collection type & Mapping Description** |
| 1 | [java.util.Set](https://www.tutorialspoint.com/hibernate/hibernate_set_mapping.htm)  This is mapped with a <set> element and initialized with java.util.HashSet |
| 2 | [java.util.SortedSet](https://www.tutorialspoint.com/hibernate/hibernate_sortedset_mapping.htm)  This is mapped with a <set> element and initialized with java.util.TreeSet. The **sort** attribute can be set to either a comparator or natural ordering. |
| 3 | [java.util.List](https://www.tutorialspoint.com/hibernate/hibernate_list_mapping.htm)  This is mapped with a <list> element and initialized with java.util.ArrayList |
| 4 | [java.util.Collection](https://www.tutorialspoint.com/hibernate/hibernate_bag_mapping.htm)  This is mapped with a <bag> or <ibag> element and initialized with java.util.ArrayList |
| 5 | [java.util.Map](https://www.tutorialspoint.com/hibernate/hibernate_map_mapping.htm)  This is mapped with a <map> element and initialized with java.util.HashMap |
| 6 | [java.util.SortedMap](https://www.tutorialspoint.com/hibernate/hibernate_sortedmap_mapping.htm)  This is mapped with a <map> element and initialized with java.util.TreeMap. The **sort** attribute can be set to either a comparator or natural ordering. |

## Association Mappings

The mapping of associations between entity classes and the relationships between tables is the soul of ORM. Following are the four ways in which the cardinality of the relationship between the objects can be expressed. An association mapping can be unidirectional as well as bidirectional.

|  |  |
| --- | --- |
| **r.No.** | **Mapping type & Description** |
| 1 | [Many-to-One](https://www.tutorialspoint.com/hibernate/hibernate_many_to_one_mapping.htm)  Mapping many-to-one relationship using Hibernate |
| 2 | [One-to-One](https://www.tutorialspoint.com/hibernate/hibernate_one_to_one_mapping.htm)  Mapping one-to-one relationship using Hibernate |
| 3 | [One-to-Many](https://www.tutorialspoint.com/hibernate/hibernate_one_to_many_mapping.htm)  Mapping one-to-many relationship using Hibernate |
| 4 | [Many-to-Many](https://www.tutorialspoint.com/hibernate/hibernate_many_to_many_mapping.htm)  Mapping many-to-many relationship using Hibernate |

Component Mappings

It is very much possible that an Entity class can have a reference to another class as a member variable. If the referred class does not have its own life cycle and completely depends on the life cycle of the owning entity class, then the referred class hence therefore is called as the **Component class**.

The mapping of Collection of Components is also possible in a similar way just as the mapping of regular Collections with minor configuration differences. We will see these two mappings in detail with examples.

|  |  |
| --- | --- |
| **Sr.No.** | **Mapping type & Description** |
| 1 | [Component Mappings](https://www.tutorialspoint.com/hibernate/hibernate_component_mappings.htm)  Mapping for a class having a reference to another class as a member variable. |

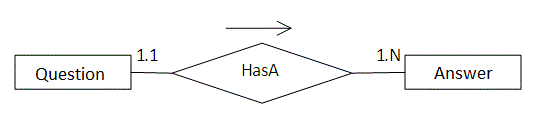
# Collection Mapping in Hibernate

We can map collection elements of Persistent class in Hibernate. You need to declare the type of collection in Persistent class from one of the following types:

* java.util.List
* java.util.Set
* java.util.SortedSet
* java.util.Map
* java.util.SortedMap
* java.util.Collection
* or write the implementation of org.hibernate.usertype.UserCollectionType

# Mapping List in Collection Mapping (using xml file)

Here, we are using the scenario of Forum where one question has multiple answers.



Let's see how we can implement the list in the mapping file:

1. **class** name="com.javatpoint.Question" table="q100">
2. ...
3. <list name="answers" table="ans100">
4. <key column="qid"></key>
5. <index column="type"></index>
6. <element column="answer" type="string"></element>
7. </list>
9. ...
10. </**class**>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**package** com.javatpoint;

* **import** java.util.List;
* **public** **class** Question {
* **private** **int** id;
* **private** String qname;
* **private** List<String> answers;//List can be of any type
* //getters and setters
* }
* **package** com.javatpoint;
* **import** java.util.List;
* **public** **class** Answer {
* **private** **int** id;
* **private** String answer;
* **private** String posterName;
* //getters and setters
* }

### Mapping collection in mapping file

There are many subelements of **<class>** elements to map the collection. They are **<list>**, **<bag>**, **<set>** and **<map>**.

Now the mapping file will be:

1. <**class** name="com.javatpoint.Question" table="q100">
2. <id name="id">
3. <generator **class**="increment"></generator>
4. </id>
5. <property name="qname"></property>
7. <list name="answers" >
8. <key column="qid"></key>
9. <index column="type"></index>
10. <one-to-many **class**="com.javatpoint.Answer" />
11. </list>
13. </**class**>

Here, List is mapped by one-to-many relation. In this scenario, there can be many answers for one question.

### Understanding key element

The key element is used to define the foreign key in the joined table based on the original identity. The foreign key element is nullable by default. So for non-nullable foreign key, we need to specify not-null attribute such as:

The attributes of the key element are column, on-delete, property-ref, not-null, update and unique.

<key

column="columnname"

on-delete="noaction|cascade"

not-**null**="true|false"

property-ref="propertyName"

update="true|false"

unique="true|false"

/>

### Indexed collections

The collection elements can be categorized in two forms:

* **indexed** ,and
* **non-indexed**

The List and Map collection are indexed whereas set and bag collections are non-indexed. Here, indexed collection means List and Map requires an additional element **<index>**.

### Collection Elements

The collection elements can have value or entity reference (another class object). We can use one of the 4 elements

* **element**
* **component-element**
* **one-to-many**, or
* **many-to-many**

The element and component-element are used for normal value such as string, int etc. whereas one-to-many and many-to-many are used to map entity reference.

Need of association mapping:

Hibernate mappings are one of the key features of [Hibernate](http://www.hibernate.org/). They establish the relationship between two database tables as attributes in your model. That allows you to easily navigate the associations in your model and Criteria queries.

You can establish either unidirectional or bidirectional i.e you can either model them as an attribute on only one of the associated entities or on both. It will not impact your database mapping tables, but it defines in which direction you can use the relationship in your model and Criteria queries.

The relationship that can be established between entities are-

* **One to One** — It represents the one to one relationship between two tables.
* **One to Many/Many to One** — It represents the one to many relationship between two tables.
* **Many to Many** — It represents the many to many relationship between two tables.
* There is the ***One to One*** relationship between *Address*. *Address*must not be an Entity as it is a value type, but for this example, we will consider *Address*as a separate Entity.
* For the **One to One** relationship, we need to simply annotate the data member of the corresponding class with **@OneToOne.**After running the application, if we look after the table created by the hibernate, we will find that the *user\_details*table has all the fields i.e **id**, **userName,** along with a foreign key of *address* table column (in this example). If we want to achieve two-way binding like *User* object should have *Address* and vice-versa. Then we have to make the User object inside Address class and annotate it with **@OneToOne.**Then, the address table will also contain the foreign key of the user column.

<https://dzone.com/articles/hibernate-mapping>