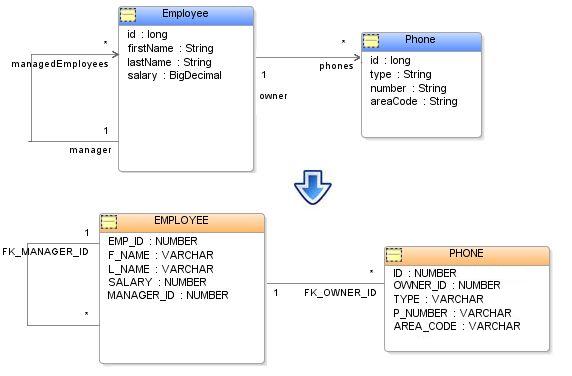
OneToMany[[edit](https://en.wikibooks.org/w/index.php?title=Java_Persistence/OneToMany&action=edit&section=1)]



A OneToMany relationship in Java is where the source object has an attribute that stores a collection of target objects and *if* those target objects had the inverse relationship back to the source object it would be a ManyToOne relationship. All relationships in Java and JPA are unidirectional, in that if a source object references a target object there is no guarantee that the target object also has a relationship to the source object. This is different than a relational database, in which relationships are defined through foreign keys and querying such that the inverse query always exists.

JPA also defines a [ManyToMany](https://en.wikibooks.org/wiki/Java_Persistence/ManyToMany) relationship, which is similar to a OneToMany relationship except that the inverse relationship (if it were defined) is a ManyToMany relationship. The main difference between a OneToMany and a ManyToMany relationship in JPA is that a ManyToMany always makes use of an intermediate relational join table to store the relationship, whereas a OneToMany can either use a join table, or a foreign key in target object's table referencing the source object table's primary key. If the OneToMany uses a foreign key in the target object's table JPA requires that the relationship be bi-directional (inverse ManyToOne relationship must be defined in the target object), and the source object must use the mappedBy attribute to define the mapping.

In JPA a OneToMany relationship is defined through the [@OneToMany](https://java.sun.com/javaee/5/docs/api/javax/persistence/OneToMany.html) annotation or the <one-to-many> element.

## Example of a OneToMany relationship database**[**[**edit**](https://en.wikibooks.org/w/index.php?title=Java_Persistence/OneToMany&action=edit&section=2)**]**

EMPLOYEE (table)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EMP\_ID | FIRSTNAME | LASTNAME | SALARY | MANAGER\_ID |
| 1 | Bob | Way | 50000 | 2 |
| 2 | Sarah | Smith | 75000 | null |

PHONE (table)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | TYPE | AREA\_CODE | P\_NUMBER | OWNER\_ID |
| 1 | home | 613 | 792-0000 | 1 |
| 2 | work | 613 | 896-1234 | 1 |
| 3 | work | 416 | 123-4444 | 2 |

## Example of a OneToMany relationship and inverse ManyToOne annotations

@Entity

**public** **class** **Employee** {

@Id

@Column(name="EMP\_ID")

**private** long id;

...

@OneToMany(mappedBy="owner")

**private** List<Phone> phones;

...

}

@Entity

**public** **class** **Phone** {

@Id

**private** long id;

...

@ManyToOne(fetch=FetchType.LAZY)

@JoinColumn(name="OWNER\_ID")

**private** Employee owner;

...

}

## Example of a OneToMany relationship and inverse ManyToOne XML**[**[**edit**](https://en.wikibooks.org/w/index.php?title=Java_Persistence/OneToMany&action=edit&section=4)**]**

**<entity** name="Employee" class="org.acme.Employee" access="FIELD"**>**

**<attributes>**

**<id** name="id"**/>**

**<one-to-many** name="phones" target-entity="org.acme.Phone" mapped-by="owner"**/>**

**</attributes>**

**</entity>**

**<entity** name="Phone" class="org.acme.Phone" access="FIELD"**>**

**<attributes>**

**<id** name="id"**/>**

**<many-to-one** name="owner" fetch="LAZY"**>**

**<join-column** name="OWNER\_ID"**/>**

**</many-to-one>**

**</attributes>**

**</entity>**

Note this @OneToMany mapping requires an inverse @ManyToOne mapping to be complete, see [ManyToOne](https://en.wikibooks.org/wiki/Java_Persistence/ManyToOne).

## Getters and Setters**[**[**edit**](https://en.wikibooks.org/w/index.php?title=Java_Persistence/OneToMany&action=edit&section=5)**]**

The relationship is bi-directional so, as the application updates one side of the relationship, the other side should also get updated, and be in sync. In JPA, as in Java in general it is the responsibility of the application, or the object model to maintain relationships. If your application adds to one side of a relationship, then it must add to the other side.

This can be resolved through add or set methods in the object model that handle both sides of the relationships, so the application code does not need to worry about it. There are two ways to go about this, you can either only add the relationship maintenance code to one side of the relationship, and only use the setter from one side (such as making the other side protected), or add it to both sides and ensure you avoid an infinite loop.

For example:

**public** **class** **Employee** {

**private** List phones;

...

**public** void addPhone(Phone phone) {

**this**.phones.add(phone);

**if** (phone.getOwner() != **this**) {

phone.setOwner(**this**);

}

}

...

}

**public** **class** **Phone** {

**private** Employee owner;

...

*/\*\**

*\* You have to ensure that the previous owner of this phone is no longer the owner of this*

*\* phone before you atribute it a new owner. Ensure this either by a*

*\* @requires !this.employee.getPhones().contains(this) or by adding to the beginning of*

*\* the method body:*

*\* if(this.employee != null)*

*\* this.employee.removePhone(this);*

*\*/*

**public** void setOwner(Employee employee) {

**this**.owner = employee;

**if** (!employee.getPhones().contains(**this**)) { *// warning this may cause performance issues if you have a large data set since this operation is O(n)*

employee.getPhones().add(**this**);

}

}

...

}

# JPA/Hibernate One To Many Relationship Mapping Example

<https://hellokoding.com/jpa-one-to-many-relationship-mapping-example-with-spring-boot-maven-and-mysql/>

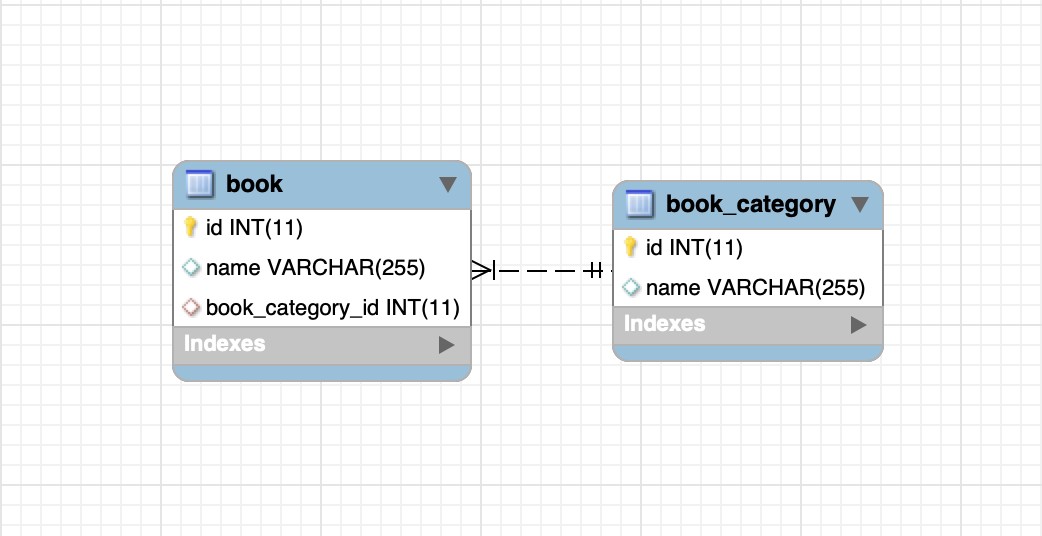
## Project structure



## One-To-Many Relationship

One-to-many relationship refers to the relationship between two entities/tables A and B in which one element/row of A may be linked with many elements of B, but a member of B is linked to only one element of A.

In this example, the book\_category and book tables have a one-to-many relationship. One category may be linked with many books but one book is linked to only one category.



book.book\_category\_id is a foreign key references to book\_category.id

## Define JPA Entities

JPA Entity is defined with @Entity annotation, represent a table in your database.

[Book.java](https://github.com/hellokoding/hellokoding-courses/blob/master/jpa-hibernate-examples/jpa-hibernate-one-to-many-mysql/src/main/java/com/hellokoding/jpa/book/Book.java)

package com.hellokoding.jpa.book;

import lombok.\*;

import javax.persistence.\*;

@Data

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

@ManyToOne

@JoinColumn

private BookCategory bookCategory;

public Book(String name) {

this.name = name;

}

}

[BookCategory.java](https://github.com/hellokoding/hellokoding-courses/blob/master/jpa-hibernate-examples/jpa-hibernate-one-to-many-mysql/src/main/java/com/hellokoding/jpa/book/BookCategory.java)

package com.hellokoding.jpa.book;

import lombok.Data;

import lombok.EqualsAndHashCode;

import javax.persistence.\*;

import java.util.Set;

import java.util.stream.Collectors;

import java.util.stream.Stream;

@Data

@EqualsAndHashCode(exclude = "books")

@Entity

public class BookCategory {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

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

private Set<Book> books;

public BookCategory(String name, Book... books) {

this.name = name;

this.books = Stream.of(books).collect(Collectors.toSet());

this.books.forEach(x -> x.setBookCategory(this));

}

}

@Id declares the entity identifier.

@Column maps the entity's field with the table's column. If @Column is omitted, the field name of the entity will be used as column name by default.

@OneToMany and @ManyToOne defines a one-to-many and many-to-one relationship between 2 entities. @JoinColumn indicates the entity is the owner of the relationship and the corresponding table has a column with a foreign key to the referenced table. mappedBy indicates the entity is the inverse of the relationship.

# [Storing a Map<String,String> using JPA](https://stackoverflow.com/questions/3393649/storing-a-mapstring-string-using-jpa)

<https://stackoverflow.com/questions/3393649/storing-a-mapstring-string-using-jpa>