1. Pesquisar e descrever resumidamente como implementar os demais tipos de relacionamento (One to Many, Many to One, e Many to Many).

- **One to Many:**
  - Esse relacionamento informa que um registro de uma entidade está relacionadao com vários outros registros de uma entidades.
- **Many to One:**
  - Esse relacionamento indica que existe muitos registros de uma entidade associados a outro registro de uma entidade.
- **Many to Many:**
  - Esse relacionamento indica que existe muitos registros de uma entidade associada a muitos registros de uma outra entidade.

---

2. Fazer a implementação de um relacionamento Onte to Many considerando que um Departamento pode conter vários Funcionários (criar os atributos que julgar necessário).

```
package br.com.luanhroliveira.atividade04.entity;

import lombok.Getter;
import lombok.Setter;
import org.springframework.data.jpa.domain.AbstractPersistable;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "department")
public class Department extends AbstractPersistable<Long> implements Serializable {

    private String name;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "employee_id", nullable = false)
    private List<Employee> employees = new ArrayList<>();
}
```

```
package br.com.luanhroliveira.atividade04.entity;

import lombok.Getter;
import lombok.Setter;
import org.springframework.data.jpa.domain.AbstractPersistable;

import javax.persistence.Entity;
import javax.persistence.Table;
import java.io.Serializable;

@Getter
@Setter
@Entity
@Table(name = "employee")
public class Employee extends AbstractPersistable<Long> implements Serializable {

    private String name;
}
```

---

3. Fazer a implementação de um relacionamento Many to One considerando que muitos Pedidos podem pertencer a um Cliente (criar os atributos que julgar necessário).

```
package br.com.luanhroliveira.atividade04.entity;

import lombok.AccessLevel;
import lombok.Getter;
import lombok.Setter;
import org.springframework.data.jpa.domain.AbstractPersistable;

import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import java.io.Serializable;

@Getter
@Setter
@Entity
@Table(name = "order")
public class Order extends AbstractPersistable<Long> implements Serializable {

    private String name;

    @Setter(AccessLevel.NONE)
    @ManyToOne
    @JoinColumn(name = "costumer_id", nullable = false)
    private Costumer costumer;
}
```

```java
package br.com.luanhroliveira.atividade04.entity;

import lombok.Getter;
import lombok.Setter;
import org.springframework.data.jpa.domain.AbstractPersistable;

import javax.persistence.Entity;
import javax.persistence.Table;
import java.io.Serializable;

@Getter
@Setter
@Entity
@Table(name = "costumer")
public class Costumer extends AbstractPersistable<Long> implements Serializable {
    private String name;
}
```

---

4. Fazer a implementação de um relacionamento Many to Many considerando que muitos Autores podem escrever muitos Livros (criar os atributos que julgar necessário).

```java
package br.com.luanhroliveira.atividade04.entity;

import lombok.Getter;
import lombok.Setter;
import org.springframework.data.jpa.domain.AbstractPersistable;

import javax.persistence.Entity;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "author")
public class Author extends AbstractPersistable<Long> {

    private String name;
    @ManyToMany(mappedBy = "authors")
    private List<Book> books = new ArrayList<>();
}
```

```
package br.com.luanhroliveira.atividade04.entity;

import lombok.Getter;
import lombok.Setter;
import org.springframework.data.jpa.domain.AbstractPersistable;

import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "book")
public class Book extends AbstractPersistable<Long> implements Serializable {

    private String name;

    @ManyToMany
    @JoinTable(name = "book_has_author", joinColumns =
        {@JoinColumn(name = "book_id")}, inverseJoinColumns =
        {@JoinColumn(name = "author_id")})
    private List<Author> authors = new ArrayList<>();
}
```

---

5. Pesquisar e descrever resumidamente como utilizar relacionamentos bidirecionais.

Utiliza-se o relacionamento bidirecional quando se deseja ter a informação de quem possui quem. Ou seja, quais objeto X determinada pessoa possui e quais pessoas possui determinados objetos X.

- Implementado acima, no exercício 4, a forma de se utilizar o relacionamento bidirecional. Um relacionamento bidirecional se dá quando ambos os lados do relacionamento possuem o mapeamento Many to Many.