**Mapping Concept in Spring Boot**

In Spring Boot, when we talk about mapping between entities, we usually refer to the process of mapping Java objects (entities) to database tables. This mapping is often facilitated by Object-Relational Mapping (ORM) frameworks like Hibernate, which is commonly used in Spring Boot applications.

Here's how the mapping between entities and database tables typically works in Spring Boot.  
  
**Entity Classes**: These are POJO (Plain Old Java Object) classes that represent objects in your application domain. In the context of Spring Boot and ORM frameworks like Hibernate, these classes usually represent database tables or views.  
  
**Database Tables**: These are the actual tables in your relational database where the data is stored.  
  
**Mapping Annotations**: In Spring Boot, you use annotations to specify the mapping between entity classes and database tables. For example, you might use **@Entity** to mark a class as an entity, and **@Table** to specify the name of the database table to which it maps.  
  
**Field Mapping**: You can use annotations like **@Column** to specify the mapping between entity class fields and database table columns. This includes specifying column names, types, constraints, and other attributes.  
  
**Relationship Mapping**: In addition to mapping fields to columns, you can also map relationships between entities. For example, you can use annotations like **@ManyToOne**, **@OneToMany**, **@OneToOne**, and **@ManyToMany** to define relationships like one-to-one, one-to-many, many-to-one, and many-to-many between entities.  
  
Here's a simplified example to illustrate mapping between entities in Spring Boot…..  
  
**@Entity**

**@Table(name = "employees")**

**public class Employee {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**

**private String firstName;**

**private String lastName**

**// Getters and setters**

**}**

**@Entity**

**@Table(name = "departments")**

**public class Department {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**

**private String name;**

**@OneToMany(mappedBy = "department")**

**private List<Employee> employees;**

**// Getters and setters**

**}**The **Employee** class is mapped to the **employees** table, and fields **firstName** and **lastName** are mapped to corresponding columns.

The **Department** class is mapped to the **departments** table, and field **name** is mapped to the **name** column.  
The **employees** field in the **Department** class establishes a one-to-many relationship with the **Employee** class, indicating that one department can have many employees.

By using these mapping annotations, Spring Boot handles the translation between Java objects and database tables, making it easier to work with relational databases in your application.  
  
   
  
  
 By  
 Sushmidhar Reddy Seelam