Essential Annotations for Defining Entities in Spring Boot

1 Introduction

In Spring Boot, several annotations are essential for defining entities that map to database tables. Below is an overview of the most necessary and useful annotations.

2 Essential Annotations

2.1 @Entity

Purpose: Marks the class as an entity, indicating that JPA should map it to a database table.
Usage: Essential for any class that needs to be persisted in the database.

```java
@Entity
public class Employee {
}
```

2.2 @Table

Purpose: Specifies the table name and other table-level options (optional).
Usage: Useful for customizing the table name or adding constraints.

```java
@Entity
@Table(name = "employees")
public class Employee {
}
```

2.3 @Id and @GeneratedValue

`@Id` identifies the primary key, while `@GeneratedValue` specifies the strategy for generating key values.

```java
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

2.4 `@Column`

Purpose: Customizes the mapping of a specific column.
Usage: Useful for setting attributes like `nullable`, `unique`, and `length`.

```
@Column(name = "employee_name", nullable = false, length = 100)
private String name;
```

2.5 Relationship Annotations

Purpose: Define relationships between entities. The following annotations are critical:

- `@ManyToOne`: Defines a many-to-one relationship.
- `@OneToMany`: Defines a one-to-many relationship.
- `@OneToOne`: Defines a one-to-one relationship.
- `@ManyToMany`: Defines a many-to-many relationship.

Example: Many-to-one relationship.

```
@ManyToOne
@JoinColumn(name = "department_id")
private Department department;
```

Example: One-to-many relationship.

```
@OneToMany(mappedBy = "department")
private List<Employee> employees;
```

2.6 `@JoinColumn`

Purpose: Specifies the foreign key column for relationships.
Usage: Commonly used in `@ManyToOne` and `@OneToOne` relationships.

```
@ManyToOne
@JoinColumn(name = "department_id")
private Department department;
```

2.7 Validation Annotations

Purpose: Ensure data integrity. Common validation annotations include:

- `@NotNull`: Ensures a field is not null.
- `@Size`: Limits the length of strings.
- `@Pattern`: Restricts values based on a regular expression.

Example:

```java
@NotNull
@Size(max = 100)
private String name;
```

2.8 @Lob

Purpose: Marks a field for large objects, such as large strings or binary data.
Usage: Use this for fields that may store large amounts of data.

```java
@Lob
private String description;
```

3 Practical Example: Employee Entity

Here is a complete example of an `Employee` entity using these annotations:

```java
import jakarta.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.util.List;

@Entity
@Table(name = "employees")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "employee_name", nullable = false, length = 100)
    @NotNull
    @Size(max = 100)
    private String name;

    @ManyToOne
    @JoinColumn(name = "department_id")
    private Department department;

    @Lob
    private String description;

    // Getters and Setters
}
```

4 Summary

The most practical annotations for defining entities in Spring Boot include:

- `@Entity`
- `@Table`
- `@Id` and `@GeneratedValue`
- `@Column`
- Relationship annotations (`@ManyToOne`, `@OneToMany`, etc.)
- `@JoinColumn`
- Validation annotations (`@NotNull`, `@Size`, etc.)
- `@Lob`

These annotations provide a comprehensive foundation for managing persistence, relationships, and data integrity in your Spring Boot applications.