

NEW!
v

Spring Boot para web

Frameworks Back-End



João Félix

Visão Geral

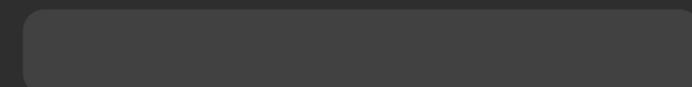
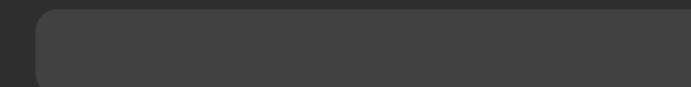
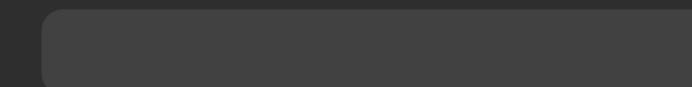
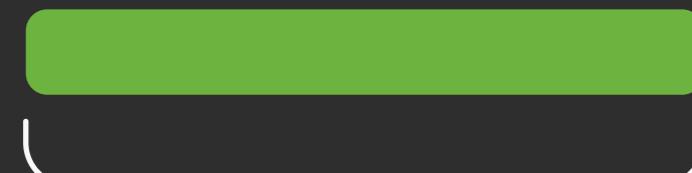
Tópicos

FERRAMENTAS DE
DESENVOLVIMENTO

RELACIONAMENTO DE
ENTIDADES

OBJETO DE
TRANSFERÊNCIA DE
DADOS

INTERACAO COM O
BANCO DE DADOS



Lombok

Spring Boot DevTools

FERRAMENTAS DE DESENVOLVIMENTO

Lombok



O Lombok é uma biblioteca Java que simplifica a escrita de código boilerplate, reduzindo a necessidade de escrever código repetitivo.

Como Instalar o Lombok



Maven

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

Gradle

```
dependencies {
    implementation("org.projectlombok:lombok:1.1
8.26")
}
```

Algumas Anotações do Lombok



@Getter: Gera métodos getter para todos os campos da classe. Isso significa que o Lombok criará automaticamente métodos public para acessar o valor de cada campo, como `getNome()`, `getIdade()`, etc.

@Setter: Gera métodos setter para todos os campos da classe. Similar ao **@Getter**, o Lombok cria automaticamente métodos public para definir o valor de cada campo, como `setNome(String nome)`, `setIdade(int idade)`, etc.

@ToString: O Lombok gera automaticamente um método `toString()` para sua classe. Este método personalizado irá incluir informações sobre os valores dos campos da classe, tornando a saída de impressão mais informativa para debugging e logs.

@AllArgsConstructor: Gera um construtor que recebe um argumento para cada campo da classe. Isso facilita a criação de novos objetos da classe, passando os valores de todos os campos de uma só vez.

@NoArgsConstructor: Gera um construtor sem argumentos. Isso garante que a classe sempre tenha um construtor disponível.

FERRAMENTAS DE DESENVOLVIMENTO

Exemplo



Antes

```
@Entity  
public class User {  
  
    @Id @GeneratedValue  
    private Long id;  
    private String firstName;  
    private String lastName;  
    private String email;  
    private Date age;  
    private String gender;  
  
    public User() {}  
  
    public User(String firstName, String lastName, String email, date age, String  
gender) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.email = email;  
        this.age = age;  
        this.gender = gender;  
    }  
    // getters and setters  
    ...  
}
```

Depois

```
@Entity  
@Getter @Setter @NoArgsConstructor  
public class User {  
  
    @Id @GeneratedValue  
    private Long id;  
    private String firstName;  
    private String lastName;  
    private String email;  
    private Date age;  
    private String gender;  
  
    public User(String firstName, String lastName, String email, Date age,  
String gender) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.email = email;  
        this.age = age;  
        this.gender = gender;  
    }  
}
```

FERRAMENTAS DE DESENVOLVIMENTO

Spring Boot DevTools



A dependência Spring Boot DevTools fornece ferramentas para melhorar a experiência de desenvolvimento durante a criação de aplicações Spring Boot

Como Instalar o Spring Boot DevTools



Maven

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
```

Gradle

```
dependencies {
  compileOnly("org.springframework.boot:spring-
  boot-devtools")
}
```

Spring Boot DevTools O Que ele Faz ?



Reinício automático: Quando você modifica arquivos específicos do classpath da sua aplicação (como classes Java, arquivos de configuração, templates), o Spring Boot DevTools reinicia a aplicação automaticamente. Isso elimina a necessidade de reiniciar a aplicação manualmente toda vez que você fizer uma alteração, poupando tempo.

Recarregamento automático de recursos: O DevTools também inclui um servidor LiveReload integrado. Com ele, sempre que você alterar determinados recursos (como arquivos HTML, CSS e JavaScript), o navegador é atualizado automaticamente, permitindo que você visualize as mudanças imediatamente, sem precisar reiniciar a aplicação.

Visão Geral

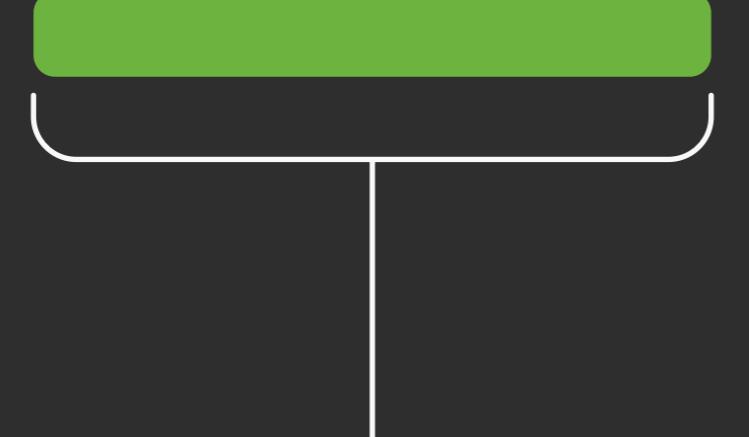
Tópicos

FERRAMENTAS DE
DESENVOLVIMENTO

RELACIONAMENTO DE
ENTIDADES

OBJETO DE
TRANSFERÊNCIA DE
DADOS

INTERACAO COM O
BANCO DE DADOS



- @OneToOne
- @ManyToOne
- @OneToMany
- @ManyToMany

Hibernate Mapping: Mapeando Relacionamentos entre Entidades



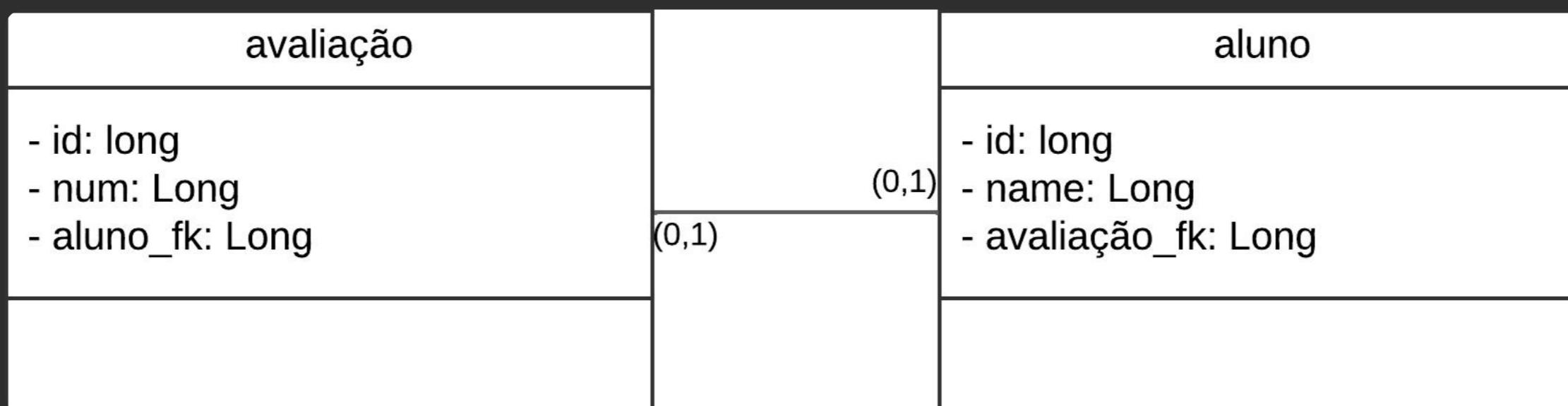
Hibernate facilita o armazenamento e a recuperação de objetos Java através do Mapeamento Objeto-Relacional (Object/Relational Mapping - ORM). O Hibernate oferece aos desenvolvedores a opção de criar mapeamentos entre modelos de base de dados e modelos de objetos através de duas formas: arquivos XML ou através de anotações no código fonte dos objetos.

O Hibernate usa e suporta anotações JPA 2 que por sua vez suportam o mapeamento entre entidades. JPA 2 suporta as associações One-to-one (Um para Um), One-to-Many (Um para Muitos), Many-to-one (Muitos para Um) e Many-to-Many (Muitos para Muitos).

Mapeamento @OneToOne



Se você usar o @OneToOne você modela o caso 1-para-1. Você pode fazer com que uma avaliação pertença a apenas um Aluno, mas nesse tipo de relacionamento, você também tem que um aluno só pode ter uma avaliação.



RELACIONAMENTO DE ENTIDADES

Exemplos

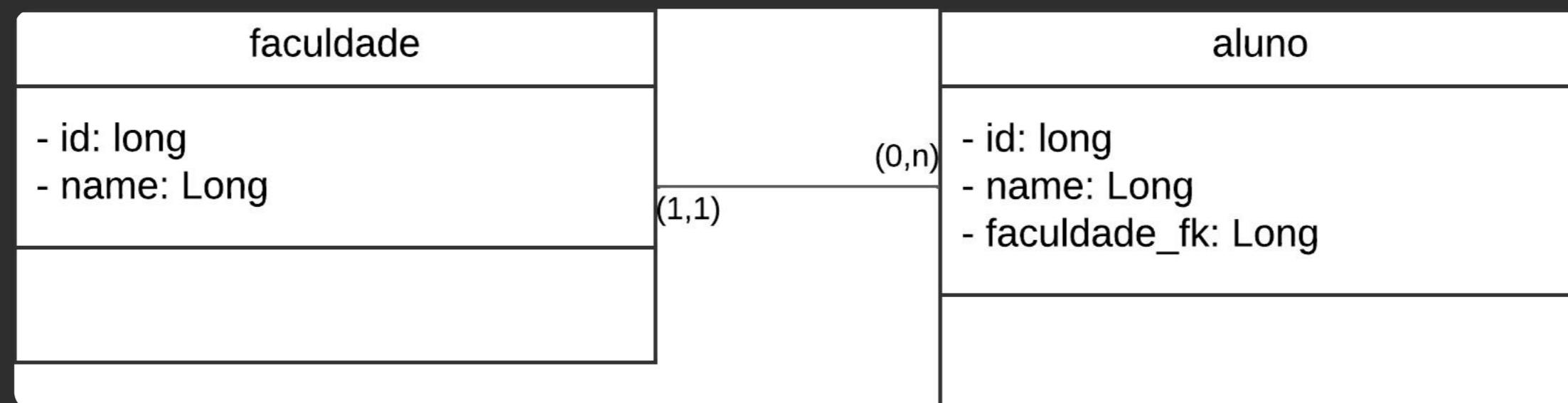
```
@Entity  
public class Avaliacao{  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(nullable = false, unique = true, updatable = false)  
    private Long id;  
  
    @OneToOne  
    @JoinColumn(name = "aluno_id")  
    private Aluno aluno;
```

```
@Entity  
public class Aluno {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(nullable = false, unique = true, updatable = false)  
    private Long id;  
    private String name;  
    private String email;  
  
    @OneToOne(mappedBy = "aluno")  
    private Avaliacao avaliacao;
```

Mapeamento @ManyToOne



O @ManyToOne significa muitos-para-1. Neste seu exemplo (vamos supor que o campo faculdade na tabela Aluno deveria se chamar `faculdade_id`), teríamos isso:



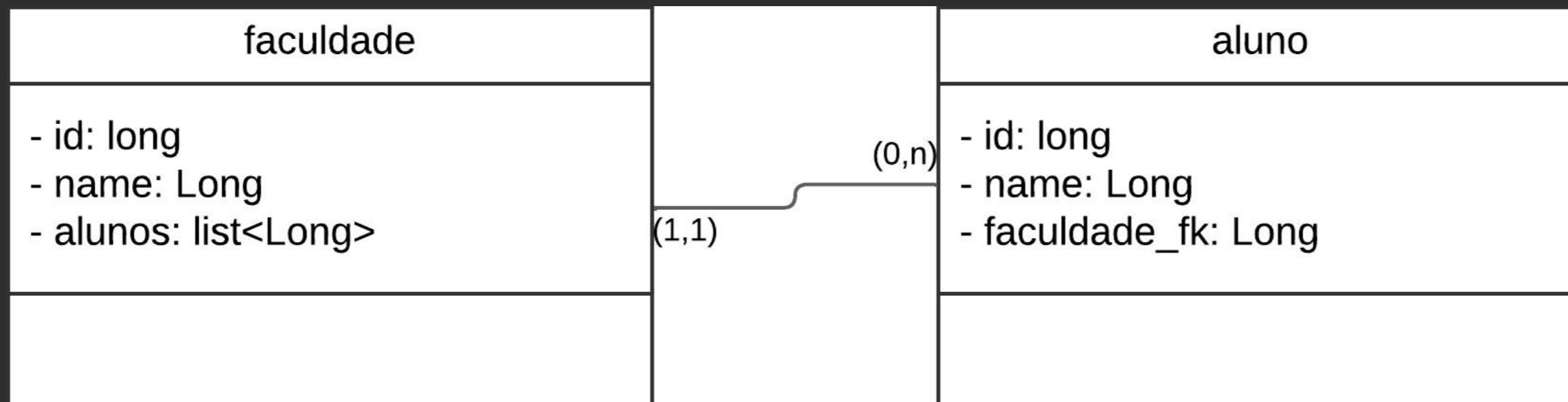
Exemplo

```
@Entity  
public class Aluno {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(nullable = false, unique = true, updatable = false)  
    private Long id;  
    private String name;  
    private String email;  
  
    @ManyToOne  
    @JoinColumn(name="faculdade_id", nullable=false)  
    private Faculdade faculdade;
```

Mapeamento @OneToMany



O @OneToMany é o oposto do que o @ManyToOne, ou seja é o 1-para-muitos. Por exemplo, poderíamos fazer isso:



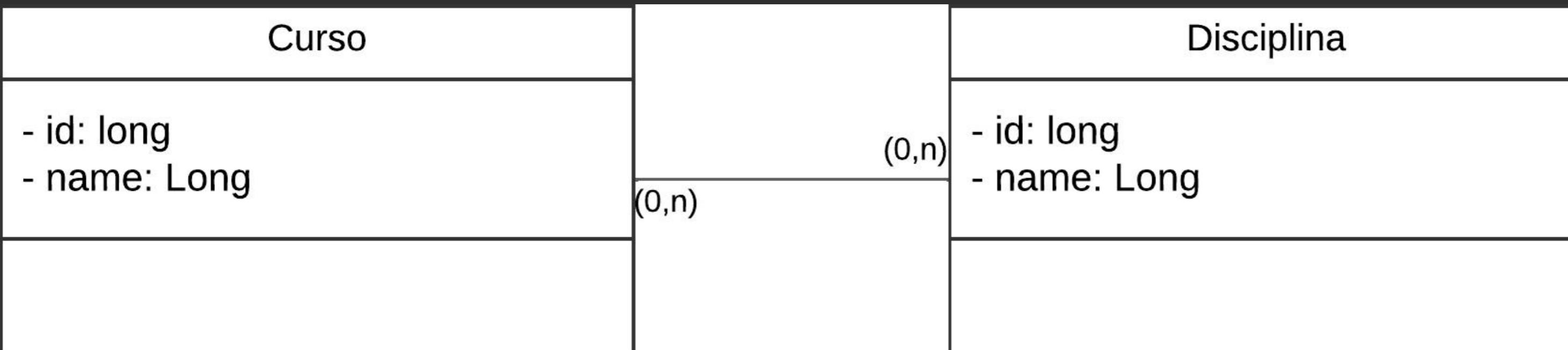
Exemplo

```
@Entity  
public class Faculdade {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(nullable = false, unique = true, updatable =  
false)  
    private Long id;  
    private String name;  
  
    @OneToMany  
    @JoinColumn(name = "faculdade_id")  
    private List<Aluno> alunos;
```

Mapeamento @ManyToMany



No relacionamento @ManyToMany, geralmente é criada uma tabela intermediária (também chamada de tabela de junção ou join table) para armazenar as associações entre as duas entidades. Cada linha dessa tabela intermediária contém chaves estrangeiras que referenciam as tabelas das entidades relacionadas.



RELACIONAMENTO DE ENTIDADES

Exemplos

```
@Entity  
public class Curso {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(nullable = false, unique = true, updatable = false)  
    private Long id;  
    private String name;  
  
    @ManyToMany  
    private List<Disciplina> disciplinas;
```

```
@Entity  
public class Disciplina{  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(nullable = false, unique = true, updatable = false)  
    private Long id;  
    private String name;  
  
    @ManyToMany(mappedBy = "disciplinas")  
    private List<Curso> cursos;
```

Visão Geral

Tópicos

FERRAMENTAS DE
DESENVOLVIMENTO

RELACIONAMENTO DE
ENTIDADES

OBJETO DE
TRANSFERÊNCIA DE
DADOS

INTERACAO COM O
BANCO DE DADOS

O que é
Prática

OBJETO DE TRANSFERÊNCIA DE DADOS

O que é



Data Transfer Object (DTO) ou simplesmente Transfer Object é um padrão de projetos bastante usado em Java para o transporte de dados entre diferentes componentes de um sistema, diferentes instâncias ou processos de um sistema distribuído ou diferentes sistemas via serialização.

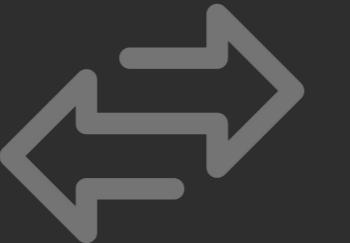
A ideia consiste basicamente em agrupar um conjunto de atributos numa classe simples de forma a otimizar a comunicação.

Numa chamada remota, seria ineficiente passar cada atributo individualmente. Da mesma forma seria ineficiente ou até causaria erros passar uma entidade mais complexa.

Além disso, muitas vezes os dados usados na comunicação não refletem exatamente os atributos do seu modelo. Então, um DTO seria uma classe que provê exatamente aquilo que é necessário para um determinado processo.

OBJETO DE TRANSFERÊNCIA DE DADOS

Prática



- 1 - Crie um projeto que contenha apenas um objeto chamado Usuario.
- 2 - Esse objeto deve ter os seguintes atributos: id, nome, email, senha e admin. Registre um usuário com os atributos nome, email e senha.
- 3 - Em seguida, registre outro usuário com os atributos nome, email, senha e admin definido como true. Observe o que acontece ao fazer isso.