

# PROGRAMAÇÃO PARA WEB I

## JPA – RELACIONAMENTOS

**Profa. Silvia Bertagnolli**

# RELACIONAMENTOS

# HERANÇA

@Inheritance

Estratégias:

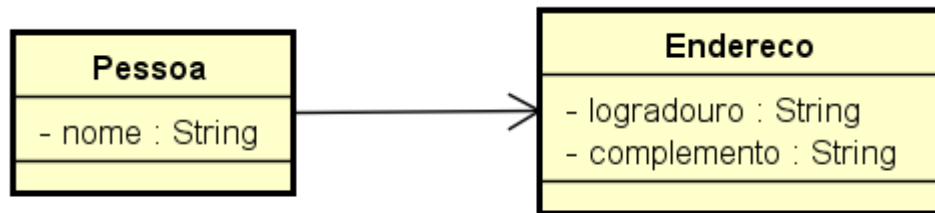
- Uma única tabela - `InheritanceType.SINGLE_TABLE` e `@DiscriminatorColumn`
- Uma tabela para cada classe (JOINED) - `InheritanceType.JOINED`
- Uma tabela por classe concreta - `InheritanceType.TABLE_PER_CLASS`

ONE TO ONE

# RELACIONAMENTOS: @ONEONE

**@OneToOne:** Usada para mapear um relacionamento “um para um”. Ou seja, cada instância de uma entidade A se relaciona com no máximo uma instância de uma entidade B, e vice-versa

Exemplo: uma Pessoa tem um Endereco



# CLASSE ENDERECO

**@Entity**

```
public class Endereco implements Serializable {  
    private static final long serialVersionUID = 1L;
```

**@Id**

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

```
    private Long idEndereco;  
    private String logradouro;  
    private String complemento;
```

```
    ...  
}
```

Classe **Endereco** é criada  
como uma entidade sem  
referência à classe Pessoa

# CLASSE PESSOA

```
@Entity
public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idPessoa;
    private String nome;
    @OneToOne
    private Endereco endereco;
```

...

Classe **Endereco** permanece inalterada, já em **Pessoa** usamos `@OneToOne` para informar que existe um relacionamento entre as duas entidades

# RELACIONAMENTOS: @ONE2ONE

Como nenhuma configuração foi realizada para indicar qual o nome da chave estrangeira na tabela Pessoa o JPA vai procurar na tabela uma coluna chamada `endereco_id`

O ideal é fazer a definição da coluna que vai determinar onde a chave estrangeira ficará e qual será o seu nome



# ERRO!!!

```
Endereco endereco = new Endereco("logradouro1", "complemento1");  
new EnderecoDAO().salvar(endereco);
```

```
Pessoa pessoa = new Pessoa("Fulano", endereco);  
PessoaDAO objPessoaDAO = new PessoaDAO();  
if (objPessoaDAO.salvar(pessoa))  
System.out.print("Pessoa foi salva!!!");
```

```
System.out.println("\nLISTAR TODOS");  
for (Pessoa p : objPessoaDAO.buscarTodos())  
System.out.printf(p.toString());
```

Unknown column 'ENDERECO\_IDENDERECO' in  
'field list'  
Error Code: 1054

# PRINCÍPIO Nº1

# PRINCÍPIO NO 1: QUEM É DOMINANTE?

Para mapear corretamente os relacionamentos devemos definir qual é o lado dominante

DOMINANTE = tabela do BD que terá a chave estrangeira

No exemplo, Pessoa conhece o seu endereço, logo ela é dominante e ela terá a FK

# CLASSE PESSOA

```
@Entity
public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    @OneToOne (cascade = CascadeType.PERSIST)
    @JoinColumn (name="idEndereco")
    private Endereco endereco;
```

...

Usamos JoinColumn para definir o nome da chave estrangeira

# CLASSE PESSOA

```
@Entity
public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    @OneToOne (cascade = CascadeType.PERSIST)
    @JoinColumn (name="idEndereco")
    private Endereco endereco;
```

...

Classe **Pessoa** é criada como uma entidade que possui a coluna de junção chamada `idEndereco` – isso faz com que o JPA procure em `Endereco` a coluna `idEndereco`

# CASCADE

É comum ter duas ou mais entidades envolvidas em uma transação

Para fazer a configuração correta do cascade você deve:

- 1) Relembre os estados de uma entidade: managed, removed, detached, ...
- 2) Conhecer os possíveis tipos de cascade

# CASCADE=CASCADETYPE

- **MERGE** = disparado toda vez que uma alteração é executada em uma entidade. Faz update nos filhos quando faz update no pai  
– você só pode dar update no objeto se ele estiver salvo
- **PERSIST** = disparado toda vez que uma nova entidade for inserida. Salva o filho quando salva o pai, sendo assim, você pode dar um save ou persist no objeto

# CASCADE=CASCADETYPE

- **REFRESH** = disparada quando uma entidade for atualizada com informações. Salva o pai e mantém o filho sem alterar
- **REMOVE** = disparada quando uma entidade é removida do BD, e os relacionamentos marcados também são eliminados. Remove o filho quando remove o pai
- **ALL** = Esse é o cascade do fim do mundo, você vai salvar, fazer update, remover, o que quiser com seu objeto, é altamente não recomendável utilizá-lo.



# COMBINAÇÕES DE CASCADE

**@Cascade(cascade={CascadeType.PERSIST,CascadeType.MERGE})**

salva em cascata, altera pai e filho em cascata

**@Cascade(cascade={CascadeType.PERSIST,CascadeType.MERGE,  
CascadeType.REMOVE})**

salva em cascata, altera pai e filho em cascata, exclui  
em cascata

# COMBINAÇÕES DE CASCADE

```
@Cascade(cascade={CascadeType.PERSIST,CascadeType.REFRESH})
```

salva em cascata, altera apenas o pai e mantém o filho

# CLASSE PESSOA

```
@Entity
public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Column(name="id_pessoa")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    @OneToOne (cascade = CascadeType.PERSIST)
    @JoinColumn (name="idEndereco")
    private Endereco endereco;
    ...
}
```

Toda vez que Pessoa for persistida o mesmo comando será repassado ao endereço

## OBSERVAÇÃO

Como o relacionamento entre Pessoa e Endereço está marcado com Cascade, o JPA ao salvar a pessoa vai gravar automaticamente o endereço




# @ONETOONE (TESTE)

```
public static void main(String[] args) {  
    Endereco endereco = new Endereco("logradouro1", "complemento1");  
    Pessoa pessoa = new Pessoa("Fulano", endereco);  
    PessoaDAO objPessoaDAO = new PessoaDAO();  
    if (objPessoaDAO.salvar(pessoa))  
        System.out.print("Pessoa foi salva!!!");  
  
    System.out.println("\nLISTAR TODOS");  
    for (Pessoa p : objPessoaDAO.buscarTodos())  
        System.out.printf(p.toString());  
}
```



Como tem Cascade.PERSIST na classe dominante, ao salvar o objeto salva o endereço também

# NO BD (@ONETOONE)...

+ Opções

←T→	▼	IDENDereco	COMPLEMENTO	LOGRADOURO
<input type="checkbox"/>	 Edita	 Copiar	 Apagar	1 complemento1 logradouro1

+ Opções

←T→	▼	idPessoa	NOME	idEndereco
<input type="checkbox"/>	 Edita	 Copiar	 Apagar	1 Fulano 1

Na tabela pessoa é feito o mapeamento automático da chave estrangeira que identifica o endereço

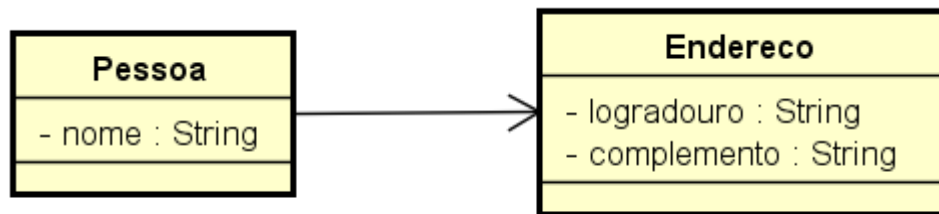
# PRINCÍPIO N°2

RELACIONAMENTOS PODEM SER UNI OU  
BIDIRECIONAIS

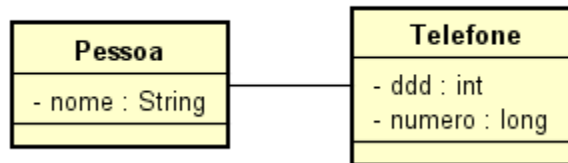


# RELACIONAMENTO UNIDIRECIONAL

- Relacionamento em que apenas uma entidade conhece a outra



# RELACIONAMENTO BIDIRECIONAL



**O que muda usando essa representação?** Telefone tem uma referência à Pessoa

# CLASSE TELEFONE

## @Entity

```
public class Telefone implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idTelefone;  
    private int ddd;  
    private long numero;
```

## @OneToOne

```
private Pessoa pessoa;
```

```
...
```

```
}
```

# CLASSE PESSOA

## @Entity

```
public class Pessoa implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idPessoa;  
    private String nome;  
    @OneToOne (cascade = CascadeType.PERSIST)  
    @JoinColumn (name="idTelefone")  
    private Telefone telefone;  
    ...  
}
```

# ISSO ESTÁ CORRETO?

**Não!!!**

Como não existe lado dominante no relacionamento o JPA analisará cada relacionamento como único – entende como dois relacionamentos diferentes e não como UM SÓ!!!

Devemos indicar qual é o relacionamento não dominante usando `mappedBy`

# CLASSE PESSOA

## @Entity

```
public class Pessoa implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idPessoa;  
    private String nome;  
    @OneToOne (optional=false, cascade = CascadeType.PERSIST)  
    @JoinColumn (name="idTelefone")  
    private Telefone telefone;  
    ...  
}
```

optional - indica que ao salvar um objeto do tipo pessoa o telefone também tem que ser informado

# CLASSE TELEFONE

## @Entity

```
public class Telefone implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idTelefone;  
    private int ddd;  
    private long numero;
```

```
    @OneToOne(mappedBy = "telefone")  
    private Pessoa pessoa;
```

O valor de mappedBy deve ser igual ao nome da propriedade na classe Pessoa que associa a classe Telefone com a classe Pessoa

...

}

# CLASSE TELEFONE

## @Entity

```
public class Telefone implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idTelefone;  
    private int ddd;  
    private long numero;
```

```
    @OneToOne(mappedBy = "telefone")  
    private Pessoa pessoa;
```

```
    ...
```

```
}
```

**RELACIONAMENTO NÃO DOMINANTE**  
Nome do atributo na classe  
Pessoa



# ATENÇÃO:

JPA NÃO FAZ O RELACIONAMENTO AUTOMATICAMENTE

# RELACIONANDO OBJETOS

```
Endereco endereco = new Endereco("logradouro1", "complemento1");
```

```
Telefone telefone = new Telefone();  
telefone.setDdd(51);  
telefone.setNumero(33445566);
```

```
Pessoa pessoa = new Pessoa();  
pessoa.setNome("Fulano");  
pessoa.setEndereco(endereco);  
pessoa.setTelefone(telefone);
```

Como a referência não foi passada para o outro objeto – o relacionamento não existirá no outro lado

Bidirecional – precisamos vincular os objetos nos 2 sentidos

# RELACIONANDO OBJETOS

```
Endereco endereco = new Endereco("logradouro1", "complemento1");
```

```
Telefone telefone = new Telefone();  
telefone.setDdd(51);  
telefone.setNumero(33445566);
```

```
Pessoa pessoa = new Pessoa();  
pessoa.setNome("Fulano");  
pessoa.setEndereco(endereco);  
pessoa.setTelefone(telefone);  
telefone.setPessoa(pessoa);
```

```
PessoaDAO objPessoaDAO = new PessoaDAO();  
if (objPessoaDAO.salvar(pessoa))  
System.out.print("Pessoa foi salva!!!");
```




Relaciona os dois lados  
**Lembre-se:** relacionamento  
**bidirecional**

Pessoa -> telefone  
Telefone -> pessoa

## NO BD (RELACIONAMENTO BIDIRECIONAL)...

+ Opções					idPessoa	NOME	idEndereco	idTelefone
	 Editar	 Copiar	 Apagar		1	Fulano	1	1

+ Opções				IDTELEFONE	DDD	NUMERO
<input type="checkbox"/>	 Editar	 Copiar	 Apagar	1	51	33445566

+ Opções		IDENDEREÇO	COMPLEMENTO	LOGRADOURO
<input type="checkbox"/>	 Editar  Copiar  Apagar	1	complemento1	logradouro1

@MANYTOMANY

# RELACIONAMENTOS: @MANYTOMANY

**@ManyToMany:** Este é o chamado relacionamento “muitos para muitos”, ou seja, cada instância de A pode se associar a várias instâncias B e vice-versa

# EXEMPLO



Precisamos definir a entidade dominante – nesse exemplo Autor  
Sistema de uma editora – o mais importante é o autor  
Sistema de uma biblioteca – o mais importante é o livro  
Como decidir? Pensar no foco do sistema

# CLASSE LIVRO

```
@Entity
public class Livro implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idLivro;
    private String titulo;
    private int anoPublicacao;
    private String editora;
    @ManyToMany(mappedBy="livros")
    private List<Autor> autores;
```

Para fazer o mapeamento bidirecional, a entidade “fraca” deve apenas fazer referência ao nome da propriedade que mapeou a coleção na entidade dominante, usando o atributo mappedBy



# CLASSE AUTOR

@Entity

```
public class Autor implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idAutor;  
    private String nome;  
    @ManyToMany (cascade= CascadeType.PERSIST)  
    @JoinTable( name="autor_rel_livro",  
                joinColumns={ @JoinColumn(name="idAutor")},  
                inverseJoinColumns={@JoinColumn(name="idLivro")})  
    private List<Livro> livros;
```

# CLASSE AUTOR

@Entity

```
public class Autor implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idAutor;  
    private String nome;  
    @ManyToMany (cascade= CascadeType.PERSIST)  
    @JoinTable( name="autor_rel_livro",  
                joinColumns={ @JoinColumn(name="idAutor")},  
                inverseJoinColumns={@JoinColumn(name="idLivro")})  
    private List<Livro> livros;
```

A entidade com a anotação  
@JoinTable é a dominante

# CLASSE AUTOR

@Entity

```
public class Autor implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idAutor;  
    private String nome;  
    @ManyToMany (cascade= CascadeType.PERSIST)  
    @JoinTable( name="autor_rel_livro",  
                joinColumns={ @JoinColumn(name="idAutor")},  
                inverseJoinColumns={@JoinColumn(name="idLivro")})  
    private List<Livro> livros;
```

Cria uma tabela de relacionamento (autor\_rel\_livro) que tem como colunas idAutor e idLivro

# CLASSE AUTOR

```
@Entity
public class Autor implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idAutor;
    private String nome;
    @ManyToMany (cascade= CascadeType.PERSIST)
    @JoinTable( name="autor_rel_livro",
                joinColumns={ @JoinColumn(name="idAutor")},
                inverseJoinColumns={@JoinColumn(name="idLivro")})
    private List<Livro> livros;
```

## Coluna 1

idAutor tem como origem a entidade atual

# COMO TESTAR

```
public class TesteManyToMany {  
    public static void main(String[] args) {  
        Livro livro1 = new Livro("Java 11", 2019, "Editora X" );  
        Livro livro2 = new Livro("JPA", 2029, "Editora Y" );  
        List<Livro> livros = new LinkedList<>();  
        livros.add(livro1);  
        livros.add(livro2);  
  
        Autor autor = new Autor();  
        autor.setNome("Fulano");  
        autor.setLivros(livros);  
        if (new AutorDAO().salvar(autor))  
            System.out.print("Autor foi salvo!!!");  
    }  
}
```

Primeiro criar os objetos mapeados com mapped, depois os que possuem o JoinTable – Salvando apenas o objeto da dominante

# NO BD (@MANYTOMANY)








## Tabela Autor

		IDAUTOR	NOME
<input type="checkbox"/>	 <a href="#">Edita</a>  <a href="#">Copiar</a>  <a href="#">Apagar</a>	1	Fulano

## Tabela Livro

		IDLIVRO	ANOPUBLICACAO	EDITORIA	TITULO
<input type="checkbox"/>	 <a href="#">Edita</a>  <a href="#">Copiar</a>  <a href="#">Apagar</a>	1	2029	Editora Y	JPA
<input type="checkbox"/>	 <a href="#">Edita</a>  <a href="#">Copiar</a>  <a href="#">Apagar</a>	2	2019	Editora X	Java 11

## Tabela autor\_rel\_livro

		idLivro	idAutor
<input type="checkbox"/>	 <a href="#">Edita</a>  <a href="#">Copiar</a>  <a href="#">Apagar</a>	1	1
<input type="checkbox"/>	 <a href="#">Edita</a>  <a href="#">Copiar</a>  <a href="#">Apagar</a>	2	1