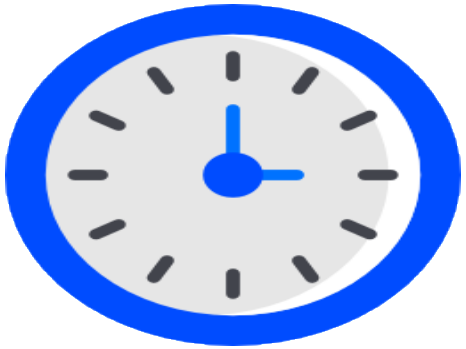
FIAP GRADUAÇÃO

JAVA ADVANCED

TRAJETÓRIA

- JPA Introdução
- JPA API
- Design Patterns
- Relacionamentos

#05 - AGENDA



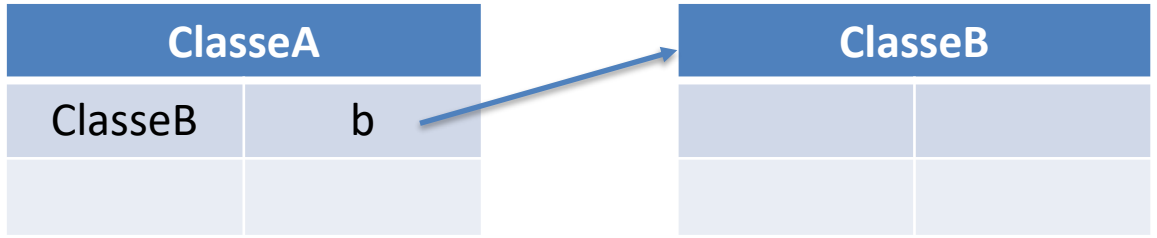
- Relacionamentos entre Entidades
  - Unidirecional X Bidirecional
- Relacionamento Um-para-Um
- Parâmetos: Cascade e Fetch
- Relacionamentos Muitos-para-Um e Um-para-Muitos
- Relacionamento Muitos-para-Muitos

RELACIONAMENTO ENTRE ENTIDADES

- Existem **quatro** tipos de associações possíveis:
  - **Um para um**
  - **Muitos para um**
  - **Um para muitos**
  - **Muitos para muitos**
- Os relacionamentos podem ser **unidirecionais** ou **bidirecionais**;
- Nas **unidirecionais** a associação ocorre em somente um sentido, isto é, somente uma das entidades associadas tem conhecimento da outra;
- Nas **bidirecionais** a associação ocorre nos dois sentidos, isto é, as entidades "enxergam-se" mutuamente;
- Para a entidade que mapeia o campo de chave estrangeira dizemos que ela é a **dona do relacionamento** (*relationship owner*);

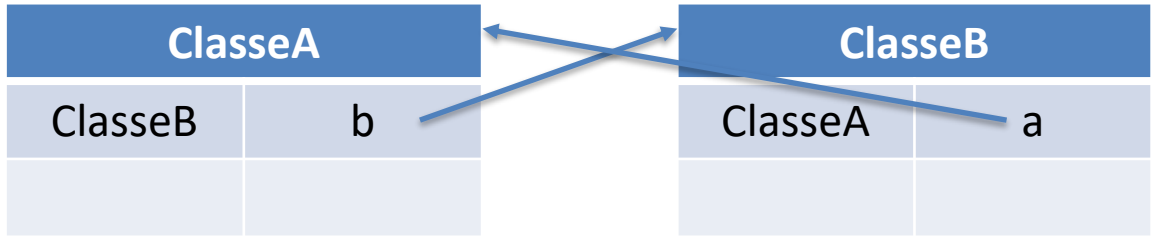
RELACIONAMENTO ENTRE ENTIDADES

Unidirecional




```
ClasseA a = new ClasseA();
ClasseB b = new ClasseB();
a.getB();   ✔
b.getA();   ✖
```

Bidirecional

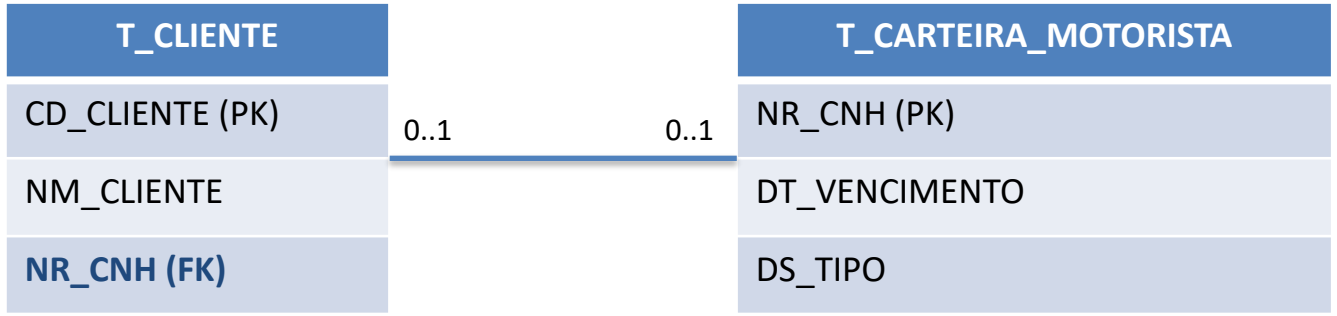



```
ClasseA a = new ClasseA();
ClasseB b = new ClasseB();
a.getB();   ✔
b.getA();   ✔
```

UM-PARA-UM

UM-PARA-UM UNIDIRECIONAL

- Uma **única** instância A pode referenciar no máximo **uma** e somente uma instância B;




```
public class Cliente {
    //Cliente acessa a Carteira de Motorista
    private CarteiraMotorista cnh;
}
public class CarteiraMotorista {
    //Carteira não acessa o Cliente
}
```

UM-PARA-UM UNIDIRECIONAL

- **Sempre** será necessário **configurar** o **tipo de relação** através de uma anotação, a anotação **@OneToOne** mapeia a associação de **um-para-um;**
- A anotação **@JoinColumn** (opcional) define o nome do campo de **chave estrangeira** na tabela associada e deve ser declarado no lado dono do relacionamento;

```
public class Cliente {
        @OneToOne
        @JoinColumn(name="NR_CNH")
        private CarteiraMotorista cnh;
}
```

UM-PARA-UM – PESQUISA E CADASTRO

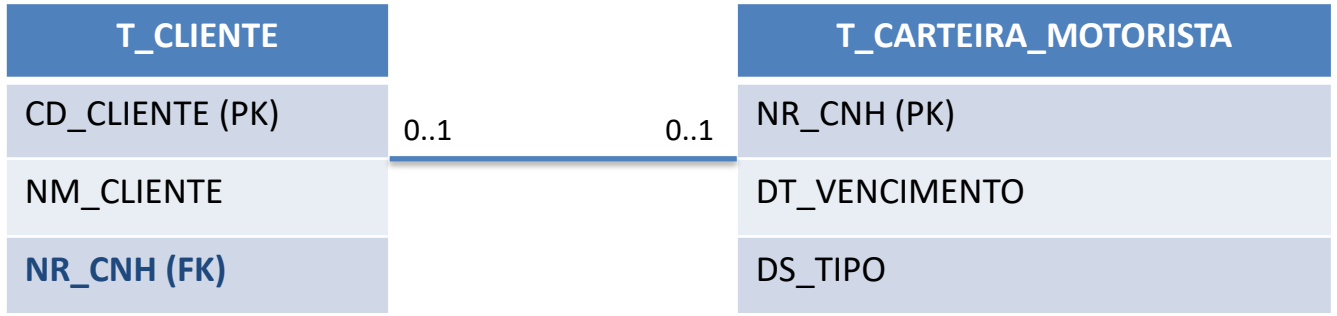
- Para criarmos um **cliente** com a **carteira de motorista**:
  1. Instanciar e persistir a **carteira de motorista**;
  2. Instanciar e definir a carteira de motorista do **Cliente**;
  3. Persistir o **Cliente**.

- Para obter a **data de vencimento da carteira** do cliente id = 10:

```
Cliente c = em.find(Cliente.class, 10);
Calendar dtVencimento = c.getCnh().getDataVencimento();
```

UM-PARA-UM BIDIRECIONAL

- Uma única instância A pode referenciar no máximo uma e somente uma instância B e vice-versa (**bidirecional**);

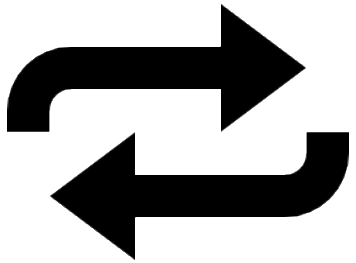



```
public class Cliente {
   //Cliente acessa a Carteira de Motorista
   private CarteiraMotorista cnh;
}
public class CarteiraMotorista {
   //Carteira acessa o Cliente
   private Cliente cliente;
}
```

UM-PARA-UM BIDIRECIONAL

- Agora a classe **CarteiraMotorista** acessa o **Cliente**;

```
public class CarteiraMotorista{
    @OneToOne(mappedBy="cnh")
    private Cliente cliente;
}
```



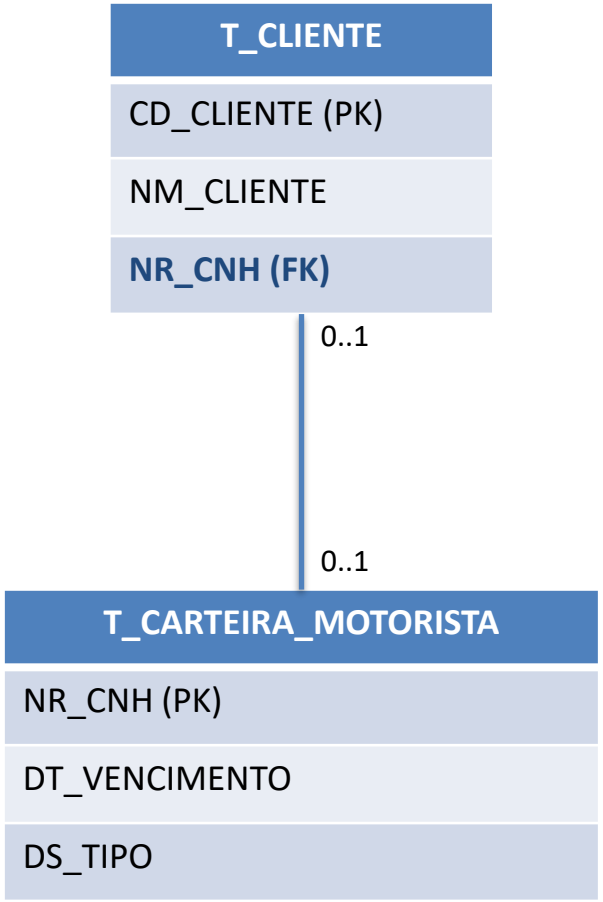
- O parâmetro **mappedBy** indica o **nome do atributo** que mapeia a associação o lado **dono da chave** estrangeira, no caso, o atributo **cnh** na entidade **Cliente**;

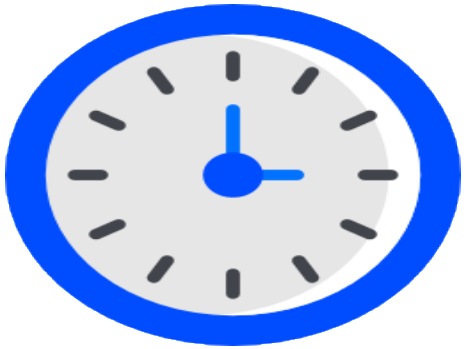
O **mappedBy** é utilizado para indicar associações **bidirecionais** e deve ser declarado no lado não dono do relacionamento.

UM-PARA-UM BIDIRECIONAL - VISÃO GERAL

```
@Table(name="T_CLIENTE")
public class Cliente {
   @OneToOne
   @JoinColumn(name="nr_cnh")
   private CarteiraMotorista cnh;
   ...
}

@Table(name="T_CARTEIRA_MOTORISTA)
public class CarteiraMotorista {
   @OneToOne(mappedBy="cnh")
   private Cliente cliente;
   ...
}
```





CASCADE

PARÂMETRO CASCADE

- Disponível para **todas** as **anotações** que mapeiam **associações**;
- Indica quando uma **alteração na entidade pai** será propagada para as **entidades filhas**;
- O parâmetro **cascade** pode assumir os valores abaixo:
  - **CascadeType.ALL** - **todas** as operações na entidade pai serão refletidas na(s) filho(s);
  - **CascadeType.MERGE** - somente operação de **merge** será refletida;
  - **CascadeType.PERSIST** - somente operação de **persist** será refletida;
  - **CascadeType.REFRESH** - somente operação **refresh** será refletida;
  - **CascadeType.REMOVE** - somente operação **remove** será refletida.
- Pode-se combinar vários tipos:

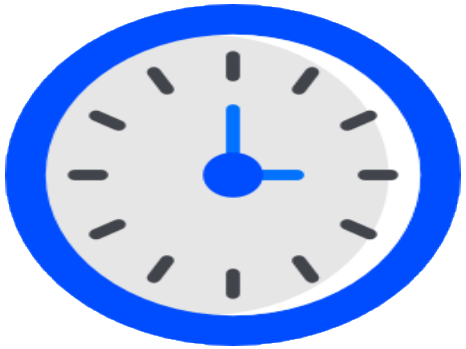
  @OneToOne(cascade={CascadeType.MERGE, CascadeType.REMOVE})

PARÂMETRO CASCADE

```
public class Cliente {
        @OneToOne(cascade=CascadeType.ALL)
        @JoinColumn(name=”nr_cnh")
        private CarteiraMotorista cnh;
}
```

- Com essa configuração ao **persistir um cliente**, também persistirá os dados da **carteira de motorista** associados;
- Para criarmos um cliente com carteira de motorista com *cascade*:
  1. Instanciar uma **carteira de motorista**;
  2. Instanciar o **cliente** e definir a sua carteira de motorista;
  3. Persistir o **cliente**.

Ao remover um cliente seus dados da carteira também serão removidos!!!



FETCH

PARÂMETRO FETCH

- Pode-se adiar o **carregamento** em memória das entidades filhas em um associação;
- Para tanto, nas associações, existe o parâmetro **fetch** que pode ser:
  - **FetchType.LAZY** - adia o carregamento das entidades filhas nas associações;
  - **FetchType.EAGER** - ao carregar o pai também carrega os filhos;

```
public class Cliente {
        @OneToOne(cascade=CascadeType.ALL, fetch=FetchType.LAZY)
        @JoinColumn(name="COD_DADO_PAGTO")
        private DadoPagamento dadoPagamento;
}
```

PARÂMETRO FETCH - LAZY

```
Cliente c  = dao.buscar(8);

System.out.println(c.getNome());
```

1

c:Cliente	
codigo	8
nome	João
cnh	

2

c:CarteiraMotorista	
numero	1234522312
tipo	B
dataVencimento	10/10/2025

2

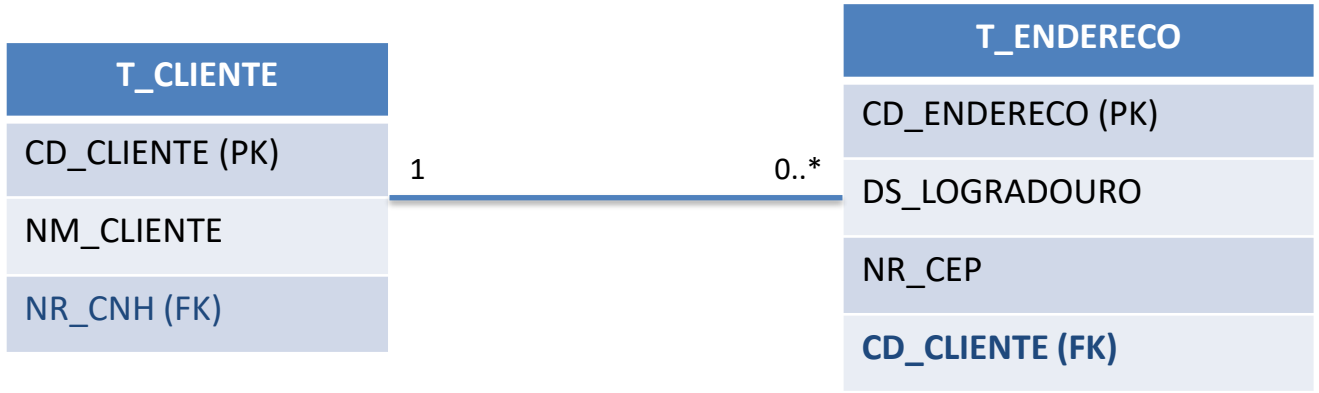
1. **Primeiro** carrega o **cliente** na memória;
2. **Quando** necessitar da **data de vencimento** da carteira, então **carrega** os dados da **carteira de motorista**;

```
System.out.println(c.getCnh().getDataVencimento());
```

MUITOS-PARA-UM E UM-PARA-MUITOS

MUITOS-PARA-UM

- **Muitas entidades filhas** associadas a **uma única entidade pai**;
- O lado **dono** do relacionamento (muitos) fará referência a **uma única** instância da entidade pai;
- Utilizar a anotação **@ManyToOne** no lado **dono** do relacionamento;
- Lembre-se de que a anotação **@JoinColumn** **pode** ser utilizada para indicar o nome da coluna que representa a chave estrangeira;



Aqui temos muitos endereços para um único cliente;

MUITOS-PARA-UM

- **Exemplo** do mapeamento:

```
public class Endereco{
        @ManyToOne
        @JoinColumn(name="CD_CLIENTE")
        private Cliente cliente;
        ...
}
```

- Dado um endereço podemos saber o nome do cliente associado a ele conforme abaixo:

```
Endereco e = em.find(Endereco.class, 10);
String nome = e.getCliente().getNome();
```

UM-PARA-MUITOS

- Para transformar uma associação ***muitos-para-um*** em **bidirecional** devemos definir o lado **não dono** da associação como ***um-para-muitos*;**
- Uma entidade representa suas muitas entidades associadas por meio de uma **Collection**;
- Utilizar a anotação **@OneToMany** no atributo que representa a associação;
- Lembre-se que o atributo **mappedBy** deve ser utilizado em conjunto com o **@OneToMany** assim como visto no **@OneToOne**;
- Além disso, os atributos **fetch** e **cascade** também continuam válidos;

UM-PARA-MUITOS

- **Exemplo** do mapeamento:

```
public class Cliente{
        @OneToMany(mappedBy="cliente", cascade=CascadeType.ALL,
        fetch=FetchType.LAZY)
        private List<Endereco> enderecos;
        ...
}
```

MÉTODO ADICIONAR

- Para adicionar novos endereços na lista do cliente é conveniente criar um método **add** conforme abaixo (na classe **Cliente**):

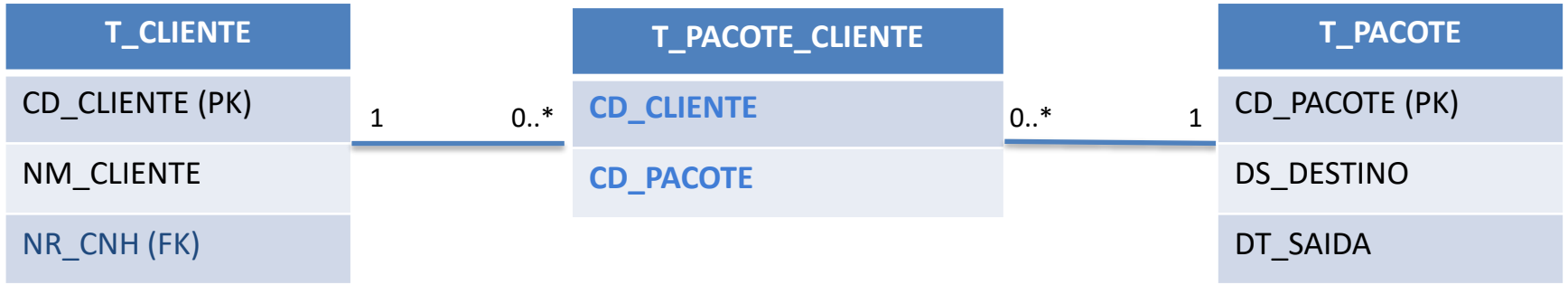
```
public void addEndereco(Endereco enderecoNovo) {
        //lembre-se que a associação é bidirecional
        enderecoNovo.setCliente(this);
        this.enderecos.add(enderecoNovo);
}
```

MUITOS-PARA-MUITOS

MUITOS-PARA-MUITOS

- **Muitas entidades A** podem ser associadas a outras **muitas entidades B** e vice-versa;
- Utiliza a anotação **@ManyToMany**;
- Representada através de uma **Collection** nas duas extremidades (caso **bidirecional**);
- Utilizar a anotação **@JoinTable** (opcional) associada para referenciar a tabela associativa e os campos de chave estrangeira:
  - **name:** nome da tabela associativa;
  - **joinColumns:** colunas de chave estrangeira que referenciam a entidade diretamente;
  - **inverseJoinColumns:** colunas de chave estrangeira que referenciam a entidade no outro lado da relação;

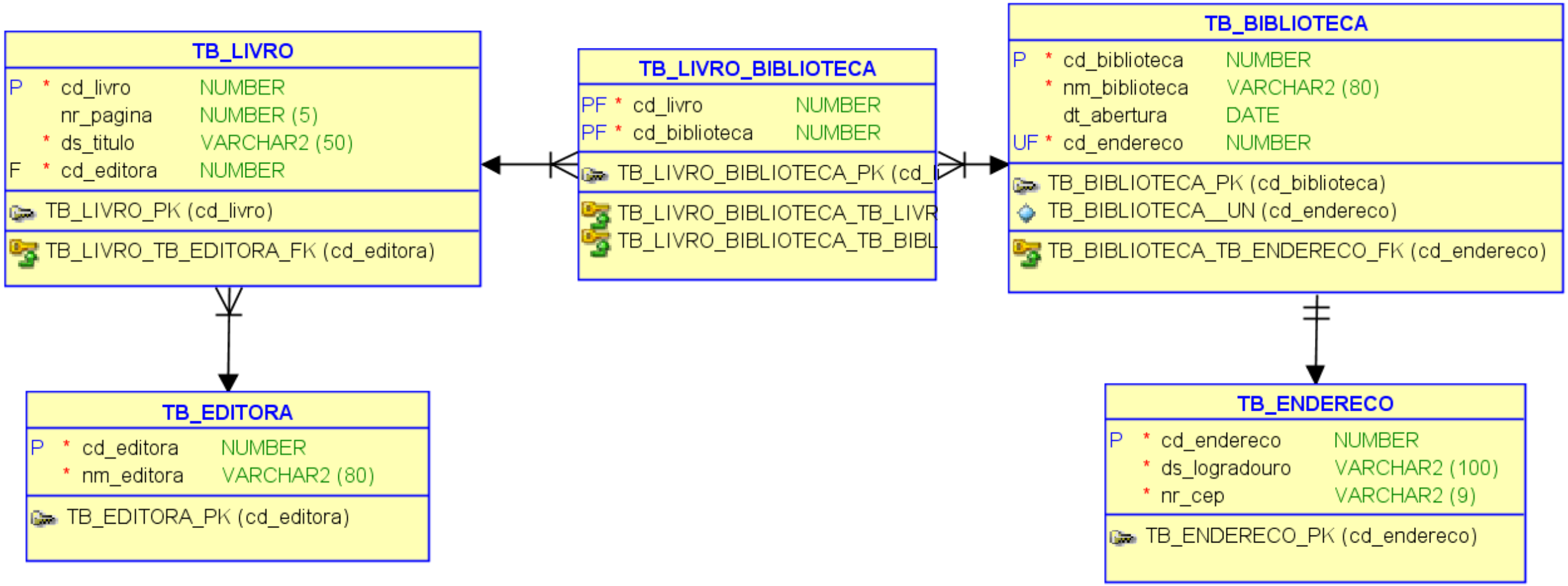
MUITOS-PARA-MUITOS




```
public class Pacote {
        @ManyToMany
        @JoinTable(name="T_PACOTE_CLIENTE",
        joinColumns=@JoinColumn(name="CD_PACOTE"),
        inverseJoinColumns=@JoinColumn(name="CD_CLIENTE"))
        private List<Cliente> clientes;
}
```

CODAR!

Implementar o mapeamento O/R para o modelo abaixo considerando quando necessário, a **navegação bidirecional** entre as entidades. Trata-se de um sistema para cadastro de livros publicados por Editoras em Bibliotecas e, ainda, realização de consulta de livros em uma determinada Biblioteca.



VOCE APRENDEU..

- Trabalhar com as relações entre as entidades de forma **unidirecional** e **bidirecional**;
- Mapear um relacionamento **um-para-um** de forma bidirecional, utilizando o parâmetro **mappedBy**;
- Utilizar os paramentos **Cascade** e **Fetch**;
- Mapear um relacionamento **muitos-para-um** e **um-para-muitos;**
- Mapear a tabela associativa para um relacionamento **muitos-para-muitos;**

Agradecimentos: Prof. Me Gustavo Torres Custódio | Prof. Me. Thiago T. I. Yamamoto

“Se a vida não ficar mais fácil, trate de ficar mais forte.”