FIAP GRADUAÇÃO

FIAP

ENTERPRISE APPLICATION DEVELOPMENT

Prof. THIAGO T. I. YAMAMOTO

#05 – JPA RELACIONAMENTOS

thiagoyama thiagoyama@gmail.com

AGENDA

- Relacionamentos Entre Entidades
- Um para Um
- Muitos para Um
- Um para Muitos
- Muitos para Muitos

RELACIONAMENTO ENTRE ENTIDADES

Em uma aplicação real as entidades encontram-se associadas entre si;

Existem quatro tipos de associações possíveis:

- Um para um
- Muitos para um
- Um para muitos
- Muitos para muitos

Além disso 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;

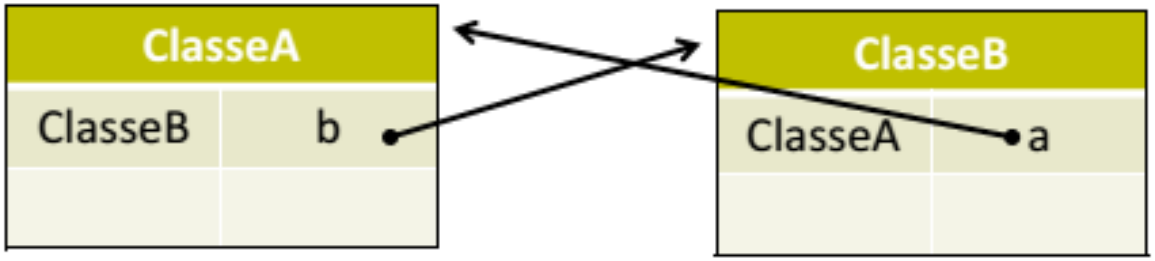
Para a entidade que mapeia o campo de chave estrangeira dizemos que ela é a dona do relacionamento (*relationship owner*);

RELACIONAMENTO ENTRE ENTIDADES

Unidirecional


```
ClasseA a = new ClasseA();
ClasseB b = new ClasseB();
a.getB() → OK!
b.getA() → ???
```

Bidirecional



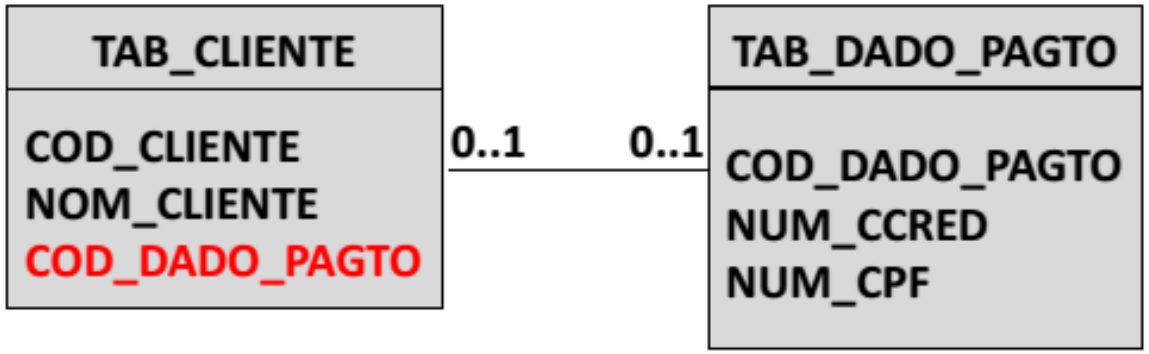

```
ClasseA a = new ClasseA();
ClasseB b = new ClasseB();
a.getB() → OK!
b.getA() → OK!
```

RELACIONAMENTO: UM PARA UM

UM PARA UM UNIDIRECIONAL

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

Como exemplo, supor a associação unidirecional entre cliente e seus dados de pagamento:




```
public class Cliente {
        // Cliente acessa dados de pagamento
        private DadoPagamento dadoPagamento;
}
public class DadoPagamento {
        // DadoPagamento NÃO acessa cliente
}
```

Duas tabelas
Duas classes
Sem herança!

UM PARA UM UNIDIRECIONAL

A anotação **@OneToOne** mapeia a associação de um para um;

```
public class Cliente {
    @OneToOne
    @JoinColumn(name="COD_DADO_PAGTO")
    private DadoPagamento dadoPagamento;
}
```

Observe que a classe **Cliente** é a dona do relacionamento com **DadoPagamento**

A anotação **@JoinColumn** define o nome do campo de chave estrangeira na tabela associada e deve ser declarado no lado dono do relacionamento;

Para criarmos um cliente com dado de pagamento:

1. Instanciar e persistir o dado de pagamento;
2. Instanciar e definir o dado de pagamento do Cliente;
3. Persistir o cliente.

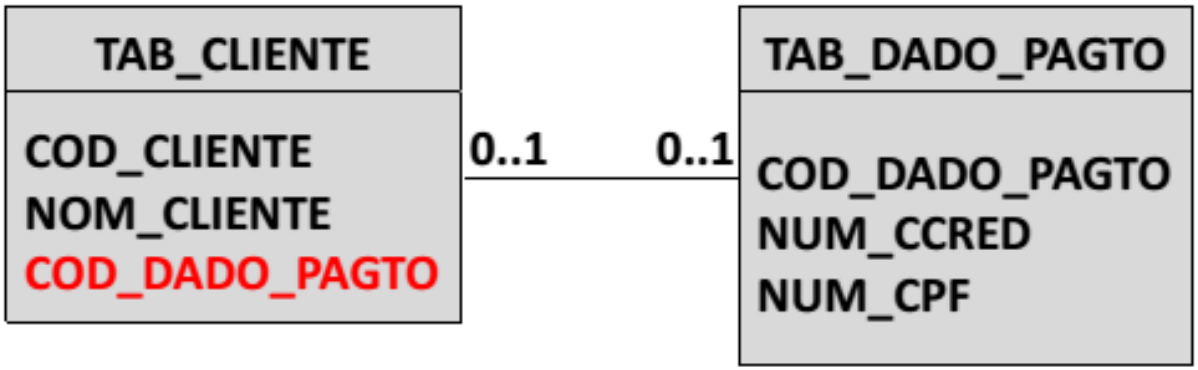
Para obter o CPF do cliente id = 10:

```
ClienteEntity c = em.find(ClienteEntity.class, 10);
String cpf = c.getDadoPagamento().getCPF();
```

UM PARA UM BIDIRECIONAL

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

Para a mesma associação entre cliente e dado de pagamento:




```
public class Cliente {
        // Cliente acessa dados de pagamento
        private DadoPagamento dadoPagamento;
}
public class DadoPagamento {
        // DadoPagamento acessa dados de Cliente
        private Cliente cliente;
}
```

UM PARA UM BIDIRECIONAL

FIAP

Agora **DadoPagamento** acessa também **Cliente**;

```
public class DadoPagamento{
    @OneToOne(mappedBy="dadoPagamento")
    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 **dadoPagamento** na entidade **Cliente**;

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

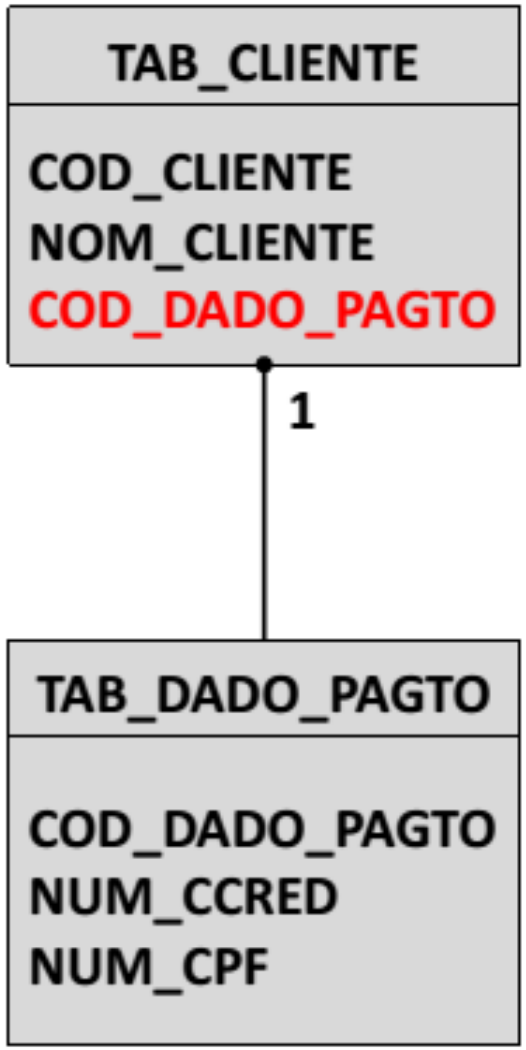
UM PARA UM BIDIRECIONAL

FIAP

Resumindo...

```
@Table(name="TAB_CLIENTE")
public class Cliente{
    @OneToOne
    @JoinColumn(name="COD_DADO_PAGTO")
    private DadoPagamento dadoPagamento;
    ...
}

@Table(name="TAB_DADO_PAGTO")
public class DadoPagamento{
    @OneToOne(mappedBy="dadoPagamento")
    private Cliente cliente;
    ...
}
```



CASCADE

PARÂMETRO CASCADE

Disponível para todas as anotações que mapeiam associações;
Indica quando uma alteração na entidade pai será propagara 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

Exemplo:

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

Com isso pode-se ao persistir um cliente, também persistir os dados de pagamento associados;

Para criarmos um cliente com dado de pagamento com *cascade*:

1. Instanciar o dado de pagamento;
2. Instanciar o cliente e definir o seu dado de pagamento;
3. Persistir o cliente.

Ao remover um cliente seus dados de pagamento também serão removidos!!!

FETCH

PARÂMETRO FETCH

Pode-se adiar o carregamento em memória das entidades filho em um associação;

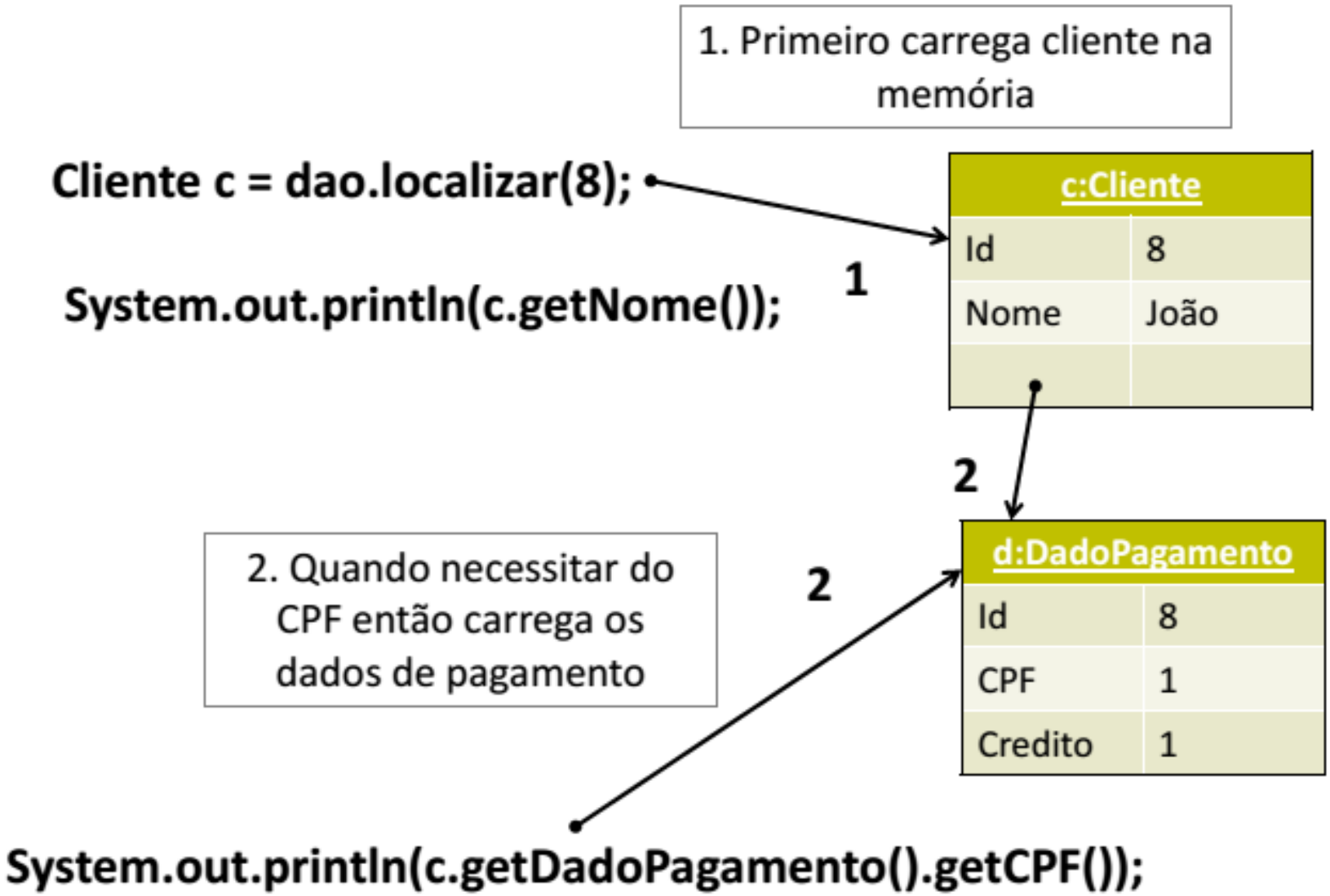
Para tanto, nas associações, existe o parâmetro **fetch** que pode ser:

- **FetchType.LAZY** - adia o carregamento das entidades filho nas associações;
- **FetchType.EAGER -** ao carregar o pai também carrega os filhos;

Exemplo:

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

PARÂMETRO FETCH - LAZY



RELACIONAMENTO: MUITOS PARA UM e UM PARA MUITOS

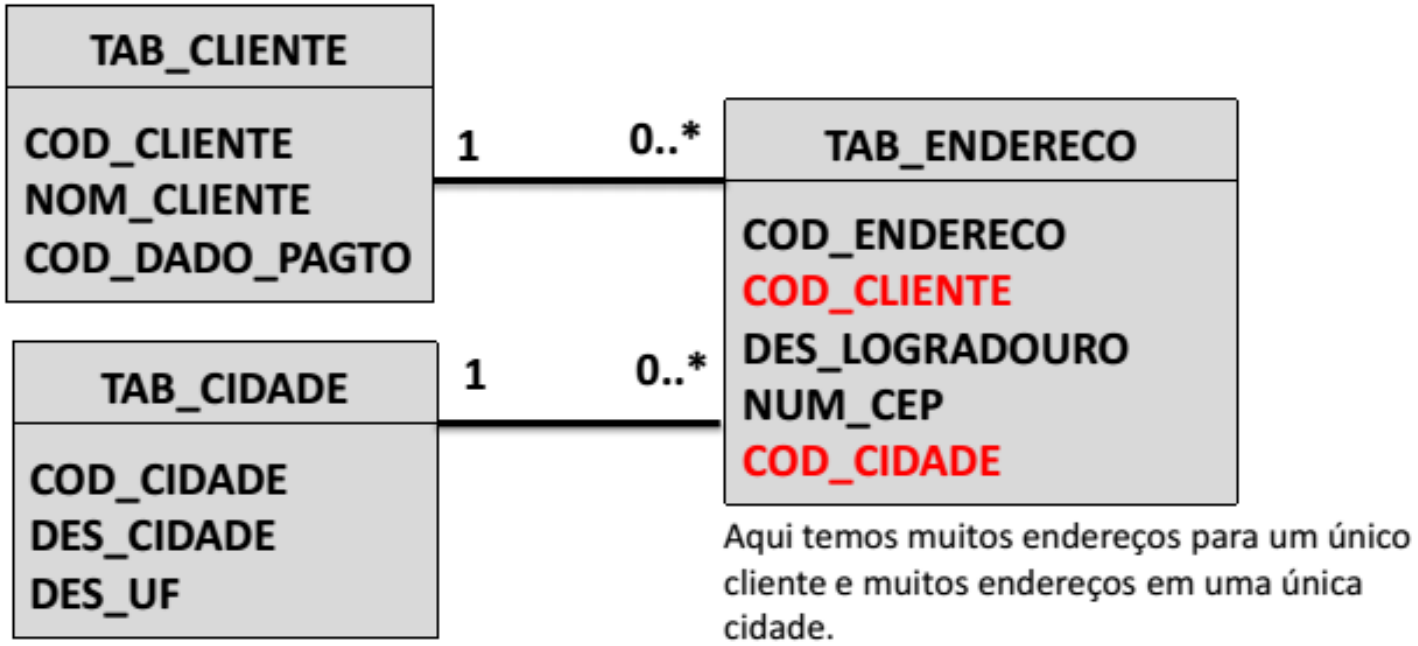
MUITOS PARA UM

Muitas entidades filho 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;



MUITOS PARA UM

Exemplo:

```
public class Endereco{
    @ManyToOne
    @JoinColumn(name="COD_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 cliente= e.getCliente().getNome();
```

Faça agora o mapeamento entre **Endereco** e **Cidade**
Como você poderia, dado um endereço, exibir o nome do cliente e o estado associado a ele?

UM PARA MUITOS

Para transformarmos uma associação *muitos para um* em bidirecional devemos definir o lado não dono da associação como sendo *um para muitos*;

Uma entidade representada 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; Exemplo:

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

DICAS...

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);
}
```

Para remover um endereço de um cliente basta incluir na anotação **@OneToMany** com endereço o atributo **orphanRemoval=true** e depois:

```
//localiza o cliente com o find tornando-o gerenciado
Cliente c = em.find(Cliente.class, 10);
//remove o primeiro endereço da lista (índice 0) de endereços do
cliente, veja que o remove aqui é um método de List e não da JPA
c.getEnderecos().remove(0);
```

RELACIONAMENTO: MUITOS PARA MUITOS

MUITOS PARA MUITOS

Muitas entidades A podem ser associadas a outras muitas entidades B e vice-versa;

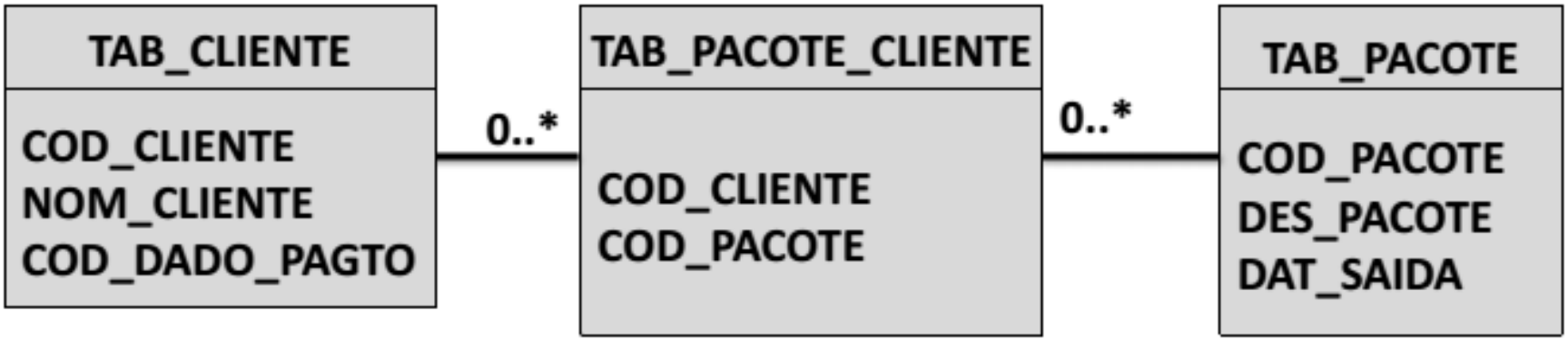
Utilizar a anotação **@ManyToMany**;

Representada através de uma **Collection** nas duas extremidades (caso bidirecional);

Utilizar a anotação **@JoinTable** 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



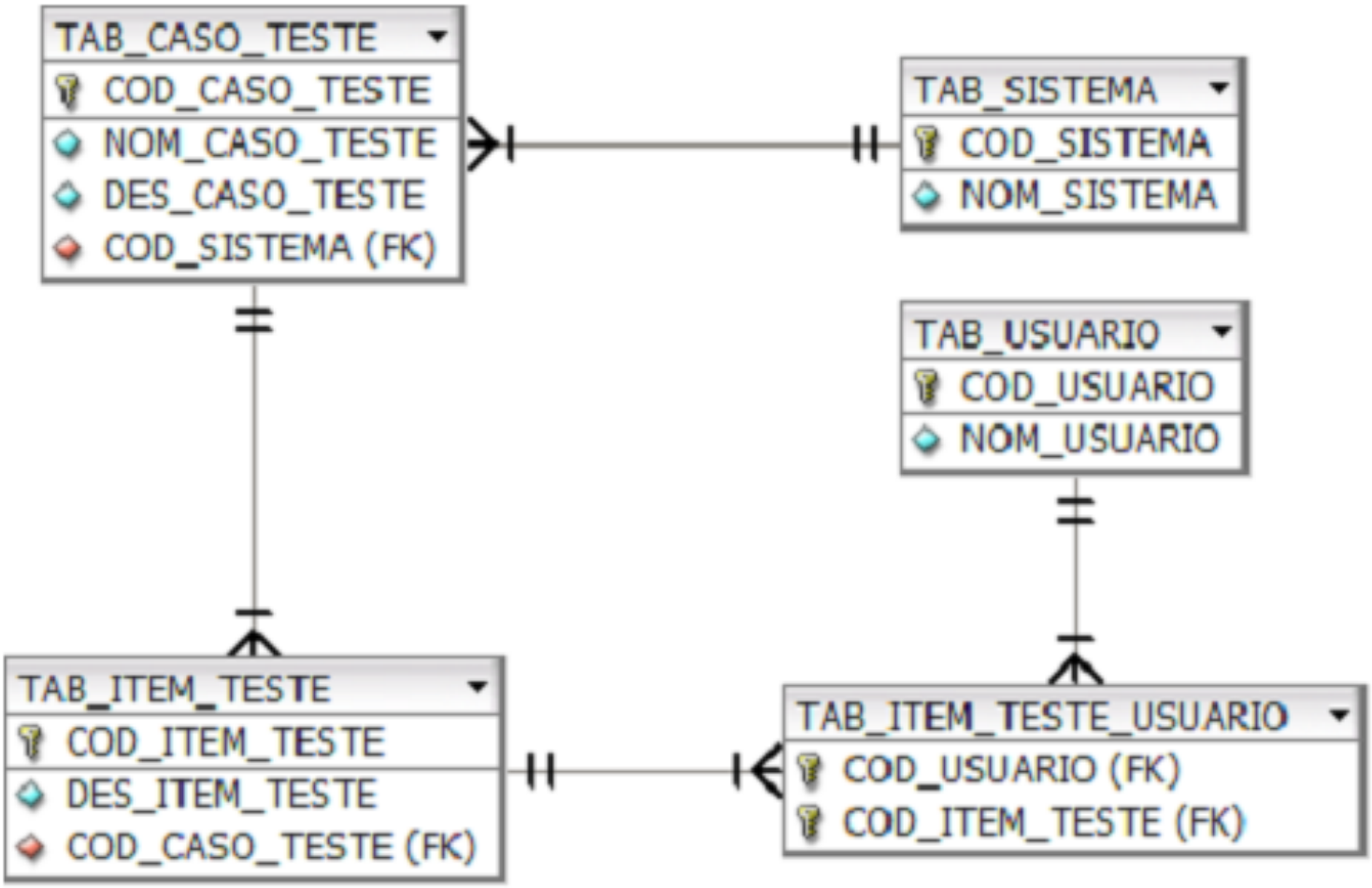

```
public class Pacote {
        @ManyToMany
        @JoinTable(name="TAB_PACOTE_CLIENTE",
        joinColumns={@JoinColumn(name="COD_PACOTE")},
        inverseJoinColumns={@JoinColumn(name="COD_CLIENTE")})
        private List<Cliente> clientes;
}
```

EXERCÍCIO

EXERCÍCIO

Implementar o mapeamento O/R para o modelo abaixo considerando sempre a navegação bidirecional entre as entidades.

Trata-se de um sistema para controlar testes de sistema efetuados pelos usuários.



FIAP

"Descobri que quanto mais eu estudo, mais sorte eu pareço ter nas provas"