INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA
Campus João Pessoa

Persistência de Objetos

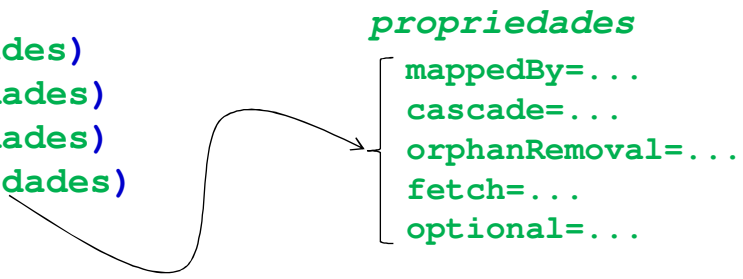
Fausto Maranhão Ayres

8 Persistência com JPA (Relacionamentos)

Tipos de Relacionamentos

- As classes devem ser anotadas de acordo com a cardinalidade

```
@OneToOne(propriedades)
@ManyToOne(propriedades)
@OneToMany(propriedades)
@ManyToMany(propriedades)
```



- As propriedades são opcionais e podem ocorrer em qualquer ordem. Ex:

```
@OneToOne(mappedBy=___,fetch =___,orphanremoval=___,
        cascade =___, optional=___)
```

Tipos de Relacionamentos

- Unidirecional
  Um lado será anotado

- Bidirecional
  Os dois lados serão anotados

Roteiro

- Persistência de relacionamento 1:* bidirecional
- Persistência de relacionamento 1:* unidirecional
- Persistência de relacionamento 1:* de tipos básicos
- Persistência de relacionamento *:*

RELACIONAMENTO 1:* BIDIRECIONAL

Propriedade do relacionamento: **mappedBy**

- É usada no lado 1 e indica a referência inversa existente no lado N

```
@Entity
public class Pessoa {...
 @Id
 private int id; ...
 @OneToMany(mappedBy="pessoa")
 private List<Telefone> telefones=..;
}
```

```
@Entity
public class Telefone {
 @Id
 private int id;
 private String numero;
 @ManyToOne
 private Pessoa pessoa;
}
```

Ref. inversa

Cuidado! Usar sempre a interface

Pessoa	
id	nome

Telefone		
id	numero	pessoa_id

O nome da chave estrangeira default é definido como: fk_pk

Propriedade do relacionamento: **cascade**

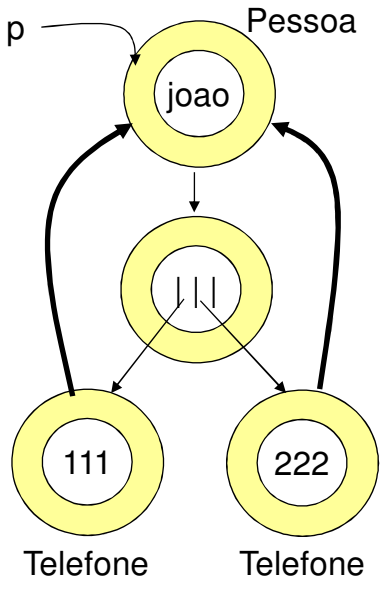
- configura a propagação das operações em cascata:

```
public class Pessoa {
  @Id
  private int id;

  @OneToMany(cascade={CascadeType.PERSIST,
       CascadeType.REMOVE, CascadeType.MERGE})
  //@OneToMany(cascade=CascadeType.ALL)

  private List<Telefone> telefones = new ...;
  ...
}
```


persist() com cascade

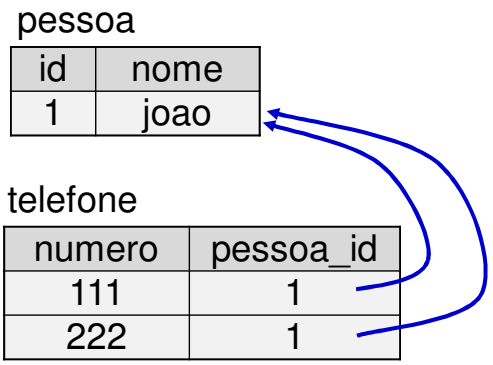
- Exemplo

```
manager.getTransaction().begin();
Pessoa p;
Telefone t1,t2;

p = new Pessoa("joao" ...);
t1 = new Telefone("111");
t2 = new Telefone("222");
p.addTelefone(t1);
p.addTelefone(t2);
t1.setPessoa(p);
t2.setPessoa(p);

manager.persist(p);
manager.getTransaction().commit();
```



SQL gerados no commit()

INSERT INTO PESSOA (NOME) VALUES ('joao')
INSERT INTO TELEFONE (NUMERO, PESSOA_ID) VALUES (111, 1)
INSERT INTO TELEFONE (NUMERO, PESSOA_ID) VALUES (222, 1)



Cuidado !!

- O que ocorre se você esquecer de relacionar bidirecionalmente os objetos na memória ?

```
...
p.addTelefone(t1);
p.addTelefone(t2);
//t1.setPessoa(p);
//t2.setPessoa(p);

manager.persist(p);
...
```

Pessoa

id	nome
1	joao

Telefone

numero	pessoa_id
111	null
222	null

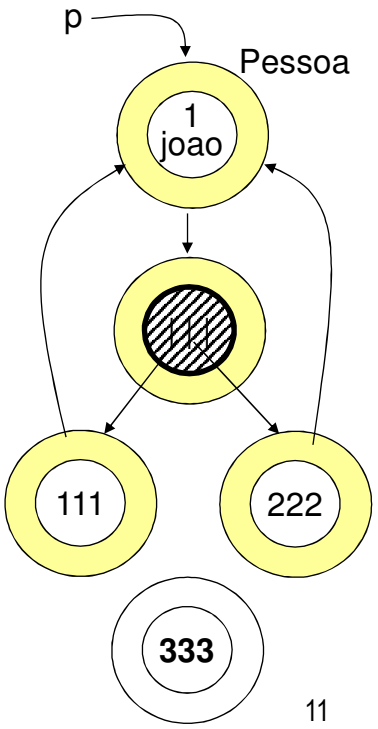
não cria o relacionamento

INSERT INTO PESSOA (NOME) VALUES ('joao')
INSERT INTO TELEFONE (NUMERO, PESSOA_ID) VALUES (111, null)
INSERT INTO TELEFONE (NUMERO, PESSOA_ID) VALUES (222, null)

Órfãos

- O filho se torna órfão no banco quando é desligado do pai

```
manager.getTransaction().begin();
Pessoa p = manager.find(Pessoa.class, 1);
Telefone t = p.localizarTelefone("333");

p.removeTelefone(t);    Telefone 333 desligado
t.setPessoa(null);

manager.merge(p);
manager.getTransaction().commit();
```



Update TELEFONE set PESSOA_ID = NULL
Where NUMERO = 333

telefone

numero	pessoa_id
111	1
222	1
333	null

pessoa

id	nome
1	joao

órfão

Remoção de órfão automática

```
public class Pessoa {
  ...
  @OneToMany(...,orphanRemoval=true ...)
  List<Telefone> telefones = ...;
}
```

Quando o telefone for removido da pessoa ele será automaticamente apagado do banco

remove() com cascade

- Exemplo: remoção de uma pessoa e seus telefones em cascata

```
manager.getTransaction().begin();

p = manager.find(Pessoa.class, 1);        //joao

if (p!=null) {
   manager.remove(p);
   manager.getTransaction().commit();
}
else
   manager.getTransaction().rollback();
```


De que lado colocar a propriedade **cascade**?

- Pode ser nos dois lados do relacionamento

```
public class Fabricante{...
  ...
  @OneToMany(
    mappedBy="fab",
    cascade=...
  )
  List<Produto> produtos;
}
```

```
public class Produto{
  ...
  @ManyToOne(cascade=...)
  Fabricante fab;
}
```

```
produto1.getFabricante().setNome("...");
manager.merge(produto1);   //atualiza o fabricante
...
```

Propriedade do relacionamento: **fetch**

- Indica a forma de **leitura** dos objetos relacionados

```
public class Pessoa {
  ...
  @OneToMany(fetch=FetchType.EAGER)
  List<Telefone> telefones = new ArrayLit<>();
}
```

FetchType.**EAGER** (leitura imediata dos telefones)
FetchType.**LAZY** (leitura postergada dos telefones)

Leitura imediata (FetchType.EAGER)

- Recomendado para coleções pequenas
- **É default** para relacionamentos 1:1

Ex:

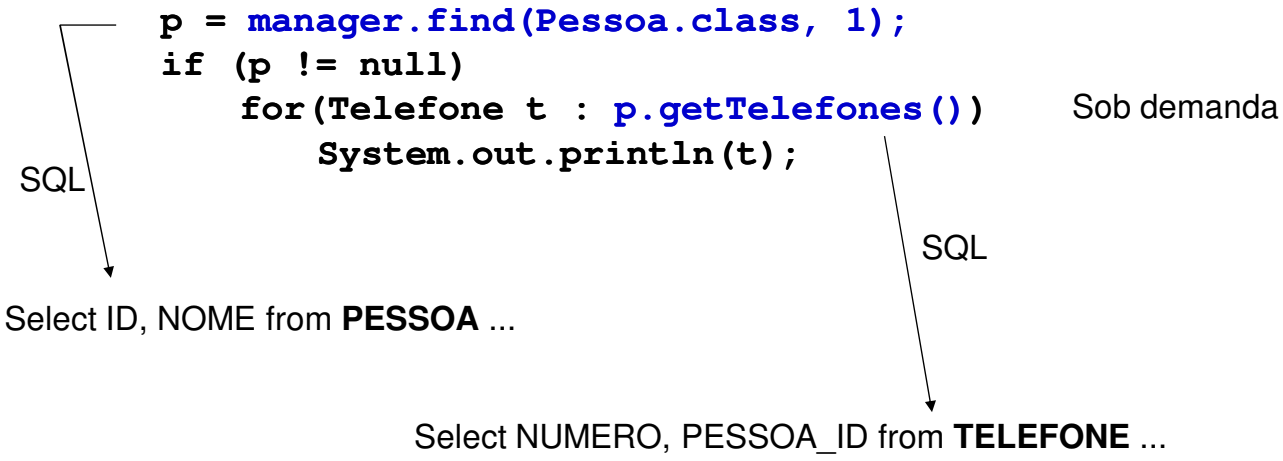
```
p = manager.find(Pessoa.class, 1);

if (p != null)
    System.out.println(p.getTelefones());
else
    System.out.println("inexistente");
```

SQL

Select ID, NOME from **PESSOA** ...
Select NUMERO, PESSOA_ID from **TELEFONE** ...

Leitura postergada (FetchType.LAZY)

- Recomendado para coleções grandes
- **É default** para relacionamentos 1:* e *:*



Obs. sobre fetch

- No Eclipselink
  - O conteúdo da coleção não será exibido automaticamente no modo LAZY. Ocorre o erro:

    "IndirectList: not instantiate"

- No Hibernate
  - O conteúdo da coleção será exibido automaticamente nos dois modos
  - Mas não aceita mais de uma coleção no modo EAGER. Ocorre o erro:

    "cannot simultaneously fetch multiple bags"

RELACIONAMENTO 1:* UNIDIRECIONAL

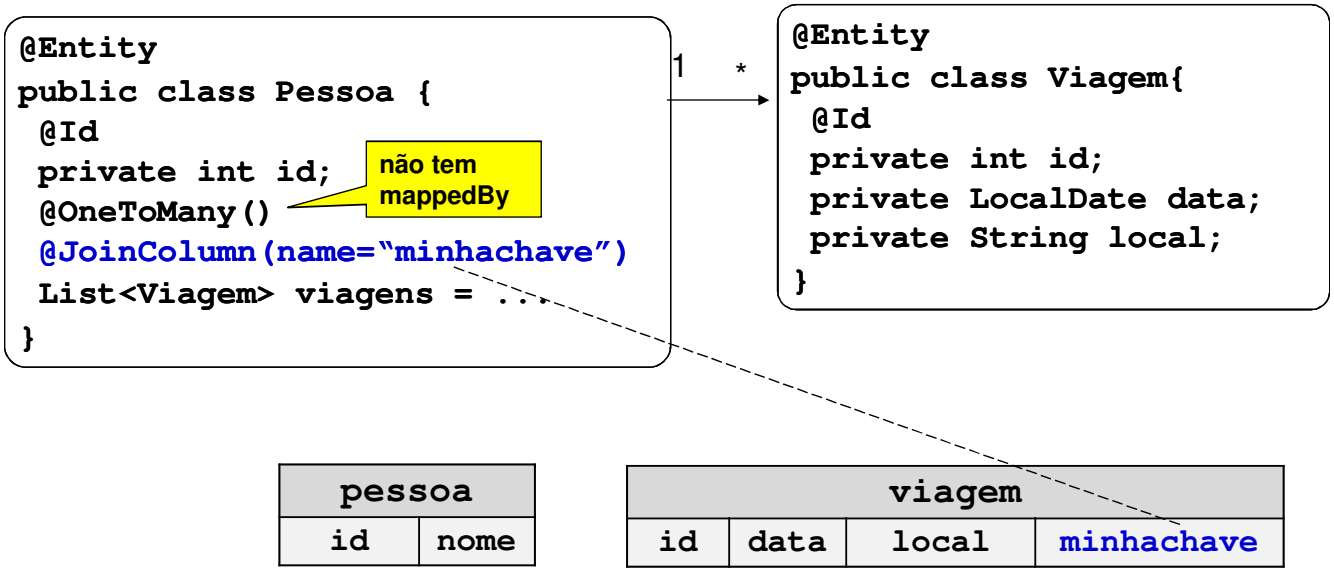
Relacionamentos 1:* unidirecional

- Como vimos anteriormente, no relacionamento **bidirecional** é criada **automaticamente** uma chave estrangeira na tabela filha (Telefone)

- No relacionamento **unidirecional** teremos que criar **manualmente** esta chave estrangeira.

Criando a chave estrangeira

@JoinColumn cria a chave estrangeira na tabela "filha"



RELACIONAMENTO 1:*
DE TIPOS BÁSICOS

Coleções de tipos básicos Java

- São coleções que não são entidades (não possuem chave primária):

  List<String>, List<Integer>, List<LocalDate>, etc.

```
@Entity
public class Pessoa {
    @Id
    private int id;
    private String nome;
    private List<String> apelidos = new...
```

Problema:

@OneToMany não pode ser usado porque só mapeia coleção de entidades (ex: List<Telefone>, List<Reuniao>)

Solução

- **@ElementCollection** mapeia a coleção para uma tabela auxiliar

```
@Entity
public class Pessoa {
    @Id
    private int id;
    private String nome;

    @ElementCollection
    private List<String> apelidos = ...;
}
```

Tabela auxiliar

pessoa	
id	nome

pessoa	apelidos
pessoa_id	apelidos

Obs: uma não-entidade não possui chave primária

RELACIONAMENTO *:*

Relacionamento muitos-para-muitos

A propriedade **mappedBy** é usada em qualquer lado do relacionamento

```
public class Pessoa{...
 @Id int id;
 String nome;
 @ManyToMany(mappedBy="pessoas")
 List<Reuniao> reuniões = ...;
}
```

```
public class Reuniao{...
 @Id LocalDate data;
 String assunto;
 @ManyToMany()
 List<Pessoa> pessoas...
}
```

tabela associativa