Desenvolvimento para Servidores-II

Relacionamentos em JPA

Neste tópico abortaremos o uso de JPA para mapear relacionamentos entre entidades

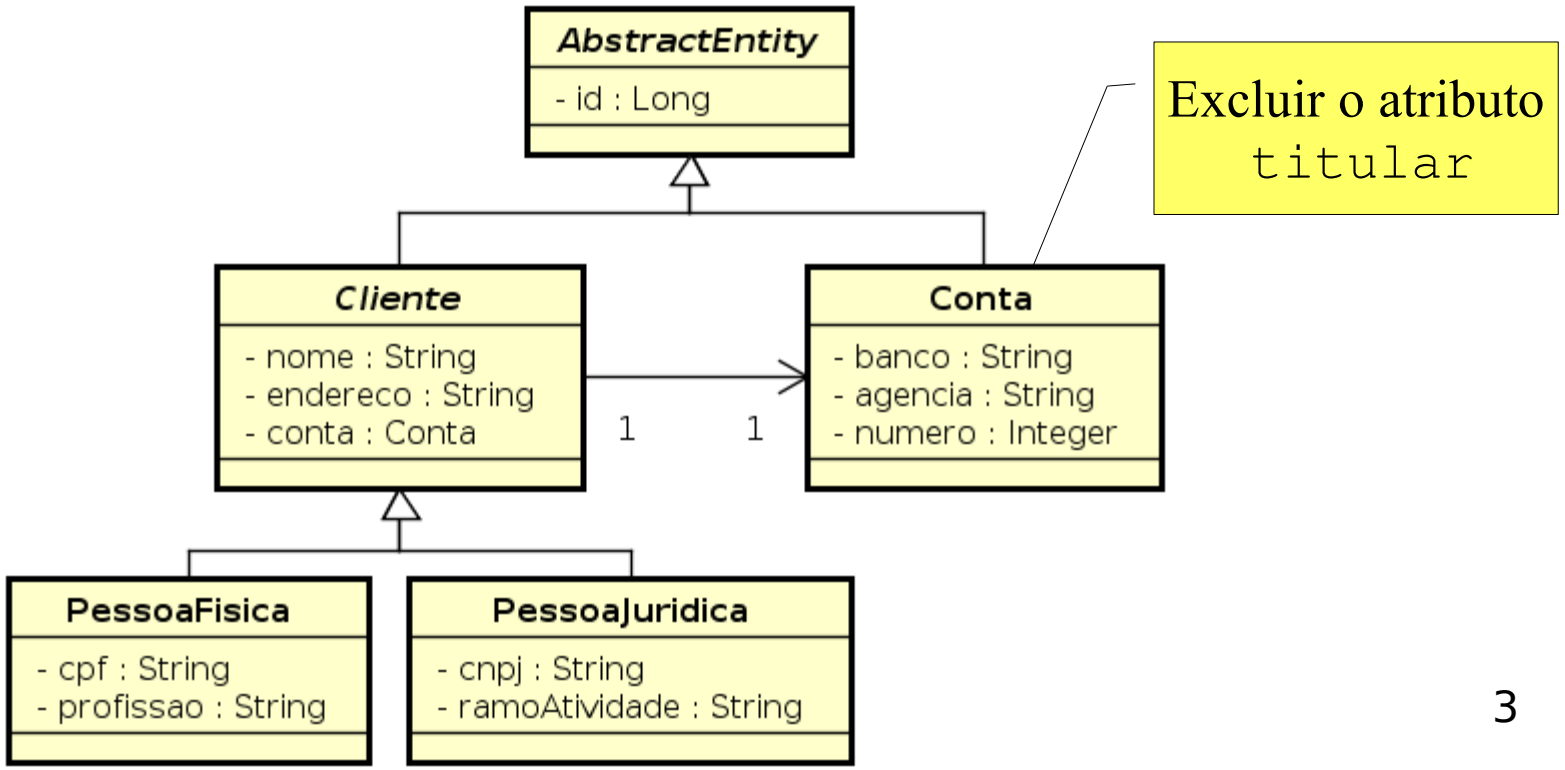
Prof. Ciro Cirne Trindade

Relacionamentos

- Os relacionamentos em JPA devem ser marcados com as seguintes anotações:
  - `@OneToOne`: um para um
  - `@OneToMany`: um para muitos
  - `@ManyToOne`: muitos para um
  - `@ManyToMany`: muitos para muitos
- Os atributos que representam relacionamentos `@OneToMany` e `@ManyToMany` devem ser `Collections`

@OneToOne

- Vamos ilustrar o relacionamento `@OneToOne` pelas classes `Conta` e `Cliente`, onde um `Cliente` possui uma única `Conta`



@OneToOne

- Para indicar um relacionamento **um para um** entre `Cliente` e `Conta` usamos a anotação `@OneToOne`:

```
@Entity
@Inheritance(strategy =
            InheritanceType.JOINED)
public abstract class Cliente
          extends AbstractEntity {
    @Column(name="nm_nome",length=60)
    private String nome;
    @Column(name="nm_endereco",
            length=120)
    private String endereco;

    @OneToOne
    private Conta conta;

    // getters e setters
}
```

```
@Entity
public class Conta
       extends AbstractEntity {
    @Column(name = "nr_agencia")
    private Integer agencia;
    @Column(name = "nm_numero",
            length = 10)
    private String numero;
    @Column(name = "vl_saldo")
    private Float saldo;

    // getters e setters
}
```

Exemplo de JSON para a classe PessoaFisica

```
{
    "nome": "Ciro Cirne Trindade",
    "endereco": "Santos(SP)",
    "cpf": "79906885874",
    "profissao": "professor",
    "conta": {
        "agencia": 9832,
        "numero": "03434-5",
        "saldo": 250.5
    }
}
```

Cascade

- Ao tentar gravar um objeto derivado da classe `Cliente` relacionado a uma `Conta` que não está no estado *Managed* (gerenciado), você receberá a seguinte mensagem de erro:
  - `org.hibernate.TransientPropertyValueException : object references an unsaved transient instance`
- Isto acontece porque o JPA não sabe o que fazer com o objeto da classe `Conta` que está relacionado a `Cliente`

Cascade

- Para ajudar ao desenvolvedor com entidades que estão associadas a outras, o JPA implantou o conceito e *Cascade*, semelhante ao que alguns bancos de dados suportam
- Permite dizer que a ação (`persist`, `merge` ou `remove`) que for disparada em uma entidade deve ser refletida para as demais entidades relacionadas

Cascade

- As definições do *Cascade* são passadas dentro das anotações `@OneToOne`, `@OneToMany` e `@ManyToMany`
- As possibilidades de valores para o *Cascade* podem ser encontradas dentro do `enum javax.persistence.CascadeType`

Principais Tipos de Cascade (1/2)

- `CascadeType.PERSIST`: disparado toda vez que uma nova entidade for inserida no banco de dados pelo comando `entityManager.persist()`
- `CascadeType.MERGE`: disparado toda vez que uma alteração é executada em uma entidade
  - Pode acontecer ao final de uma transação na qual uma *managed Entity* foi alterada, ou pelo comando `entityManager.merge()`

Principais Tipos de Cascade (2/2)

- `CascadeType.REMOVE`: disparado quando uma entidade é removida do banco de dados através do comando `entityManager.remove()`,
  - Os relacionamentos marcados também serão eliminados
- `CascadeType.ALL`: todos os eventos anteriores serão sempre refletidos nas entidades relacionadas

Cascade

- Para configurar o *Cascade*, basta indicar no relacionamento:

```
@OneToOne(cascade = {CascadeType.PERSIST,
                     CascadeType.REMOVE})
private Conta conta;
```

@OneToOne

- As tabelas geradas seriam:
  - `tb_pessoa_fisica` e `tb_pessoa_juridica`
  - `tb_conta: id, nr_agencia, nm_numero, vl_saldo`
  - `tb_cliente: id, nm_nome, ds_endereco,` **`conta_id`**
    - Como nenhuma configuração foi definida para indicar qual o nome da chave estrangeira, a JPA gerará na tabela uma coluna chamada `conta_id`
    - Por padrão o nome da coluna será o nome da entidade + o nome de seu atributo anotado com `@Id`

@OneToOne

- É possível definir o nome da chave estrangeira através da anotação `@JoinColumn` como no código:

```
@JoinColumn(name = "fk_conta_id")
private Conta conta;
```

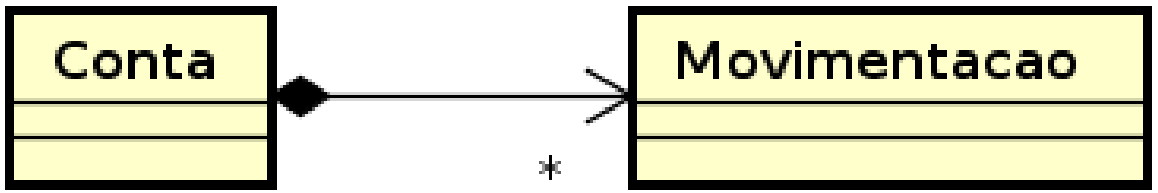
@OneToOne

- Embora o relacionamento entre `Cliente` e `Conta` seja um para um, é possível cadastrar mais de um objeto derivado de `Cliente` associado a mesma `Conta`
- Para evitar isso, defina a conta como sendo única

```
@JoinColumn(unique = true)
private Conta conta;
```

@OneToMany

- Vamos supor que uma conta está associada a várias movimentações, conforme ilustra o diagrama de classes abaixo:



Exemplo: uma conta associada a várias movimentações (1/4)

```
@Table(name = "tb_conta")
@Entity
public class Conta extends AbstractEntity {
    private static final long serialVersionUID = 1L;

    @Column(name = "nr_agencia")
    private Integer agencia;
    @Column(name = "nm_numero", length = 10)
    private String numero;
    @Column(name = "vl_saldo")
    private Float saldo;

    @OneToMany(cascade=CascadeType.ALL, orphanRemoval=true)
    private List<Movimentacao> movimentacoes;

    public Conta() { }

    // getters e setters
```

Marca entidades "filhas" para serem excluídas quando não têm qualquer outro vinculo com uma entidade pai

Exemplo: uma conta associada a várias movimentações (2/4)

Indica que o atributo ou método anotado deve ser ignorado na serialização e desserialização

Usada para mapear nomes de atributos em chaves JSON durante a serialização e desserialização

```
    @JsonIgnore
    public List<Movimentacao> getMovimentacoes() {
        return movimentacoes;
    }


    @JsonProperty
    public void setMovimentacoes(List<Movimentacao>
                                                movimentacoes) {
        this.movimentacoes = movimentacoes;
    }

}
```

Exemplo: uma conta associada a várias movimentações (3/4)

```
@Table(name="tb_movimentacao")
@Entity
public class Movimentacao extends AbstractEntity {
    private static final long serialVersionUID = 1L;

    @Column(name = "vl_valor")
    private Float valor;
    @Enumerated(EnumType.STRING)
    @Column(name = "nm_tipo_movimentacao")
    private TipoMovimentacao tipo;
    @Column(name = "ds_descricao", length = 100)
    private String descricao;
```

```
public enum TipoMovimentacao{
     ENTRADA, SAIDA;
}
```

Exemplo: uma conta associada a várias movimentações (4/4)

```
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "dt_data")
    @JsonFormat(shape = JsonFormat.Shape.STRING,
                pattern = "yyyy-MM-dd HH:mm")
    private Calendar data;

    public Movimentacao() { }

    // getters e setters
}
```

`JsonFormat` pode ser utilizado para definir o formato de serialização e desserialização de um objeto JSON

@Enumerated

- Uma `enum` de uma entidade deve ser anotada com `@Enumerated` que indica à JPA como armazenar o valor do `enum` no banco de dados
- 2 abordagens possíveis:
  - `ORDINAL` (*default*): o valor salvo no banco de dados será um valor numérico
  - `STRING`: o valor salvo no banco de dados será textual

@Temporal

- Atributos do tipo data de uma entity devem ser anotadas com `@Temporal` para indicar como a data será armazenada no banco de dados
- 3 alternativas
  - `TemporalType.DATE`: apenas a data
  - `TemporalType.TIME`: apenas a hora
  - `TemporalType.TIMESTAMP`: data e hora

@OneToMany

- Tabelas geradas
  - `tb_cliente, tb_pessoa_fisica, tb_pessoa_juridica`
  - `tb_conta: id, nr_agencia, nm_numero, vl_saldo`
  - `tb_conta_tb_movimentacao: conta_id, movimentacoes_id`
  - `tb_movimentacao: id, dt_data, ds_descricao, nm_tipo_movimentacao, vl_valor`

Tabela extra para representar o relacionamento

@OneToMany

- É possível ter o relacionamento unidirecional, no qual apenas `Movimentacao` tem referência para `Conta`, sem a tabela adicional
- Para não precisar de uma tabela adicional vamos utilizar a anotação `@JoinColumn`

Evitando a tabela extra com @JoinColumn

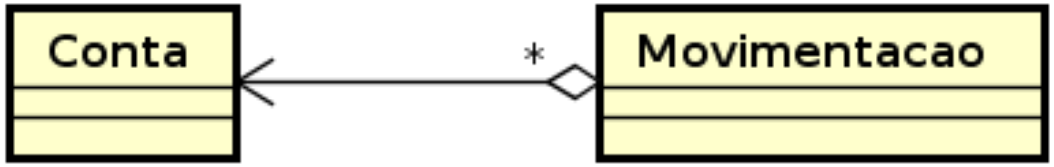
```
...
@Entity
public class Conta extends AbstractEntity {

    ...

    @OneToMany(cascade=CascadeType.ALL,
               orphanRemoval=true)
    @JoinColumn(name = "conta_id")
    private List<Movimentacao> movimentacoes;

    public Conta() { }

    // getters e setters
}
```

@OneToMany

- Uma vez que utilizamos a anotação `@JoinColumn`, o JPA procurará por uma coluna chamada `conta_id` dentro da tabela `tb_movimentacao`
- Veja como ficará a tabela `tb_movimentacao`:
  - `id, dt_data, ds_descricao, nm_tipo_movimentacao, vl_valor,` **`conta_id`**

@ManyToOne

- Um efeito semelhante seria obtido se a classe `Movimentação` fosse responsável por mapear o relacionamento com `Conta`
- Neste cado teríamos um relacionamento **muitos para um** entre `Movimentação` e `Conta`



@ManyToOne (1/2)

```
@Entity
public class Movimentacao extends AbstractEntity {
    private static final long serialVersionUID = 1L;

    @Column(name = "vl_valor")
    private Float valor;
    @Enumerated(EnumType.STRING)
    @Column(name = "nm_tipo_movimentacao")
    private TipoMovimentacao tipo;
    @Column(name = "ds_descricao", length = 100)
    private String descricao;
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "dt_data")
    @JsonFormat(shape = JsonFormat.Shape.STRING,
                pattern = "yyyy-MM-dd HH:mm")
    private Date data;

    @ManyToOne(fetch = FetchType.LAZY)
    private Conta conta;

}
```

Performance melhor que EAGER

@ManyToOne (2/2)

```
    // getters e setters

    @JsonIgnore
    public Conta getConta() {
        return conta;
    }

    @JsonProperty
    public void setConta(Conta conta) {
        this.conta = conta;
    }

}
```

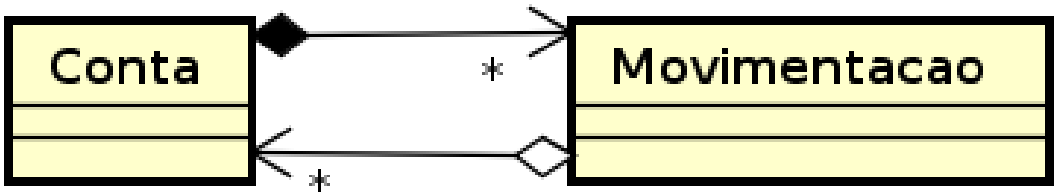
Necessário se o fetch é `LAZY`

Fetch

- FetchTypes
  - EAGER: as entidades filhas são carregadas quando a entidade pai é selecionada
  - LAZY: as entidades filhas são carregadas apenas quando são referenciadas (*on demand*)
  - Valores default
    - @OneToOne e @ManyToOne: FetchType.EAGER
    - @OneToMany e @ManyToMany: FetchType.LAZY

Relacionamentos bidirecionais

- Em relacionamentos bidirecionais é necessário informar o “dono” da relação
- Isso pode ser feito pela anotação `@JoinColumn` ou através do atributo `mappedBy` das anotações `@*ToMany`



Relacionamentos bidirecionais

```
@Entity
...
public class Movimentacao extends AbstractEntity {
    …

    @ManyToOne(fetch = FetchType.LAZY)
    private Conta conta;

    public Movimentacao() { }

    // getters e setters
}
```

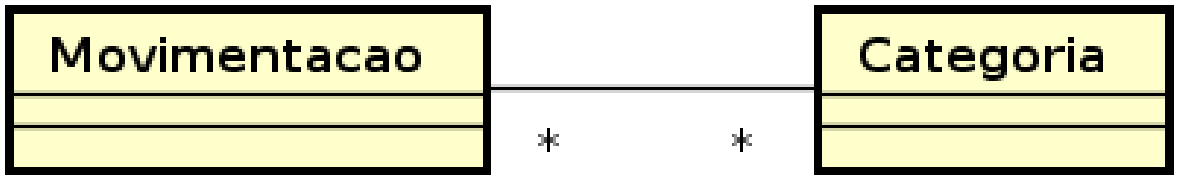
Relacionamentos bidirecionais

```
@Entity
...
public class Conta extends AbstractEntity {

    ...

    @OneToMany(cascade=CascadeType.ALL,
               orphanRemoval=true
               mappedBy="conta")
    private List<Movimentacao> movimentacoes;

    public Conta() { }

    // getters e setters

}
```

O atributo **mappedBy** foi utilizado para definir que a entidade `Movimentacao` é a dona do relacionamento

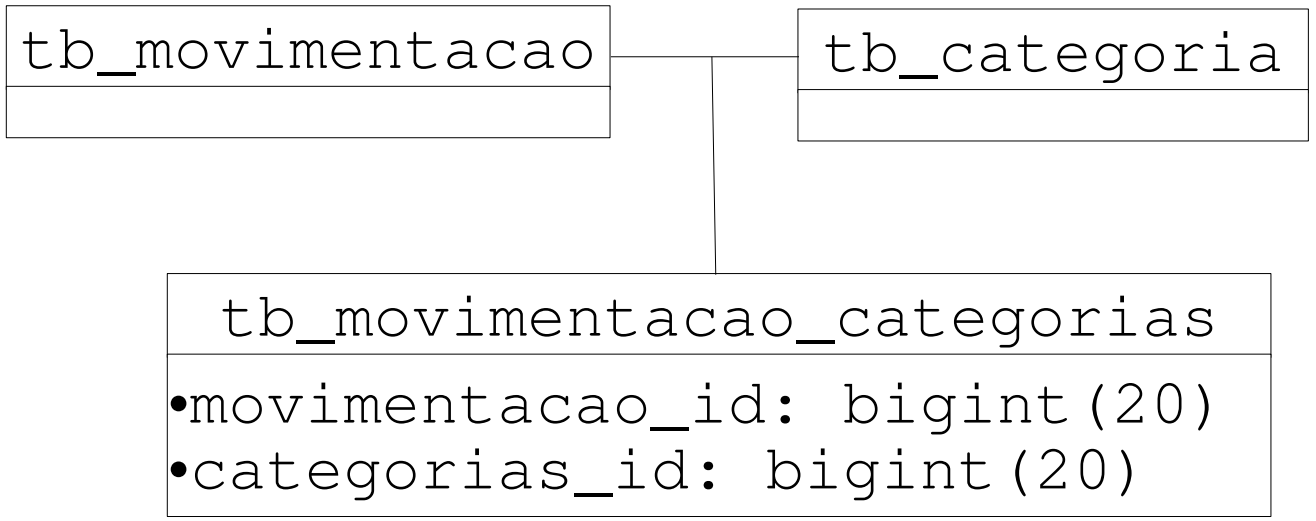
@ManyToMany

- Vamos supor que uma movimentação possa ter várias categorias (lazer, negócio, família, etc.) associadas a ela e que uma categoria possa estar associada a várias movimentações



@ManyToMany

- No caso de um relacionamento `@ManyToMany` é necessário uma tabela extra que faça a união das tabelas



@ManyToMany

```
@Table(name = "tb_categoria")
@Entity
public class Categoria extends AbstractEntity {
   private static final long serialVersionUID = 1L;
   @Column(name = "nm_nome", length = 50)
   private String nome;

   public Categoria() { }
   public String getNome() {
      return nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }
}
```

@ManyToMany

```
@Entity
...
public class Movimentacao extends AbstractEntity {
    …

    @ManyToOne(fetch = FetchType.LAZY)
    private Conta conta;


    @ManyToMany
    private List<Categoria> categorias;

    public Movimentacao() { }

    // getters e setters
}
```

@ManyToMany

- É possível usar a anotação `@JoinTable` para definir o nome da tabela de relacionamento, bem como os nomes das chaves estrangeiras

```
@ManyToMany
@JoinTable(name = "tb_movimentacao_categoria",
     joinColumns=@JoinColumn(
                         name="fk_movimentacao_id"),
     inverseJoinColumns=@JoinColumn(
                         name="fk_categoria_id"))
private List<Categoria> categorias;
```

Referências

- ORACLE Corporation. *The Java EE 7 Tutorial*. Disponível em: https://docs.oracle.com/javaee/7/JEETT.pdf, 2014.