

*Desenvolvimento Rápido de Aplicações*

# *Mapeamento Objeto Relacional*

Profa. Joyce Miranda

## Mapeamento Objeto Relacional

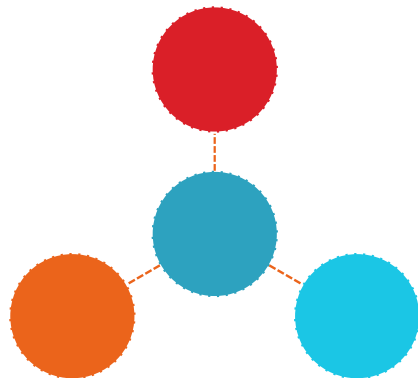
---

### ► Persistência de Dados

- Armazenamento **não-volátil** dos dados em um sistema de armazenamento.

### ► Persistência de Objetos

- Capacidade de um objeto “sobreviver” fora dos limites da aplicação.



**Objetos**  
**(Mundo OO)**

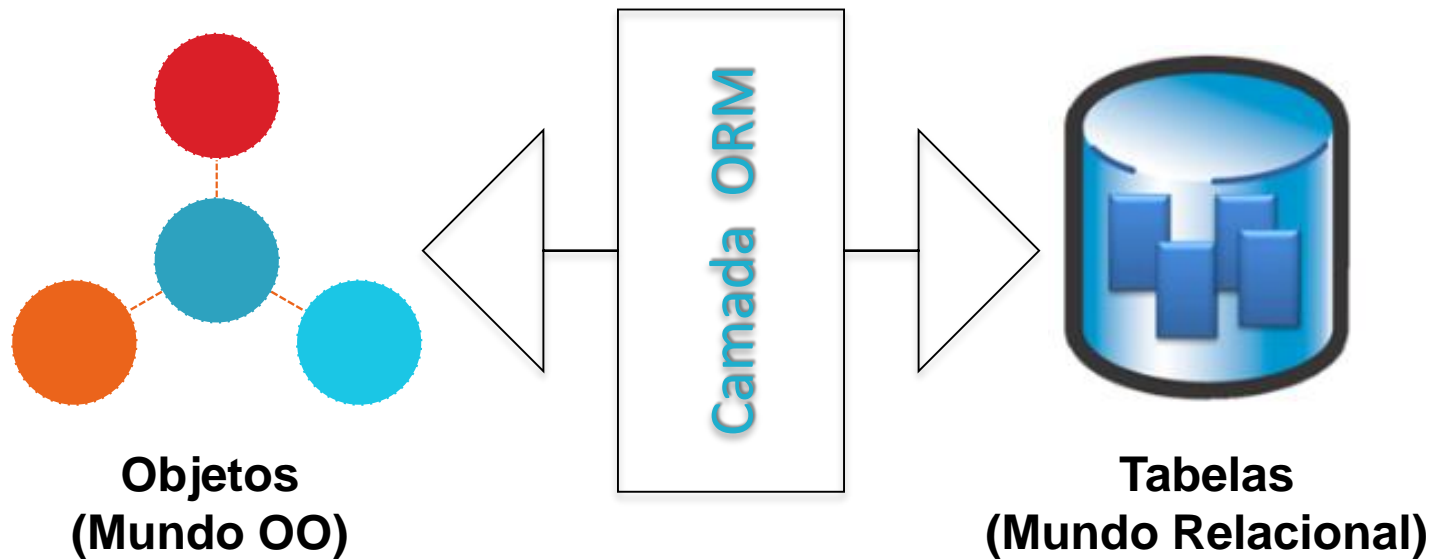


**Tabelas**  
**(Mundo Relacional)**

## Mapeamento Objeto Relacional

---

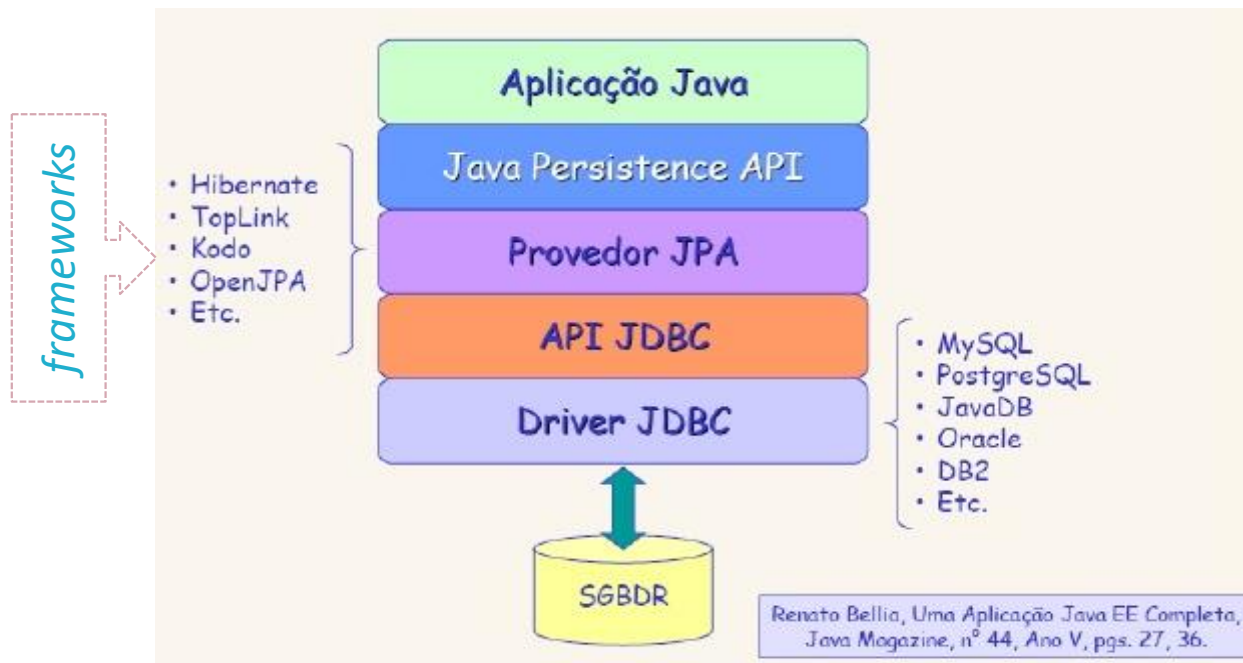
- ▶ Ferramentas ORM (*Object Relational Mapping*)
  - ▶ Representam objetos de maneira relacional na gravação do banco de dados, e conseguem fazer o caminho inverso sem perder informação.



# Mapeamento Objeto Relacional

## ► JPA – Java Persistence API

- Especificação JAVA para persistência de dados.
  - API para abstração da camada de persistência das aplicações OO.
- Deve ser implementado por **frameworks** que queiram seguir esse padrão.



## Mapeamento Objeto Relacional

---

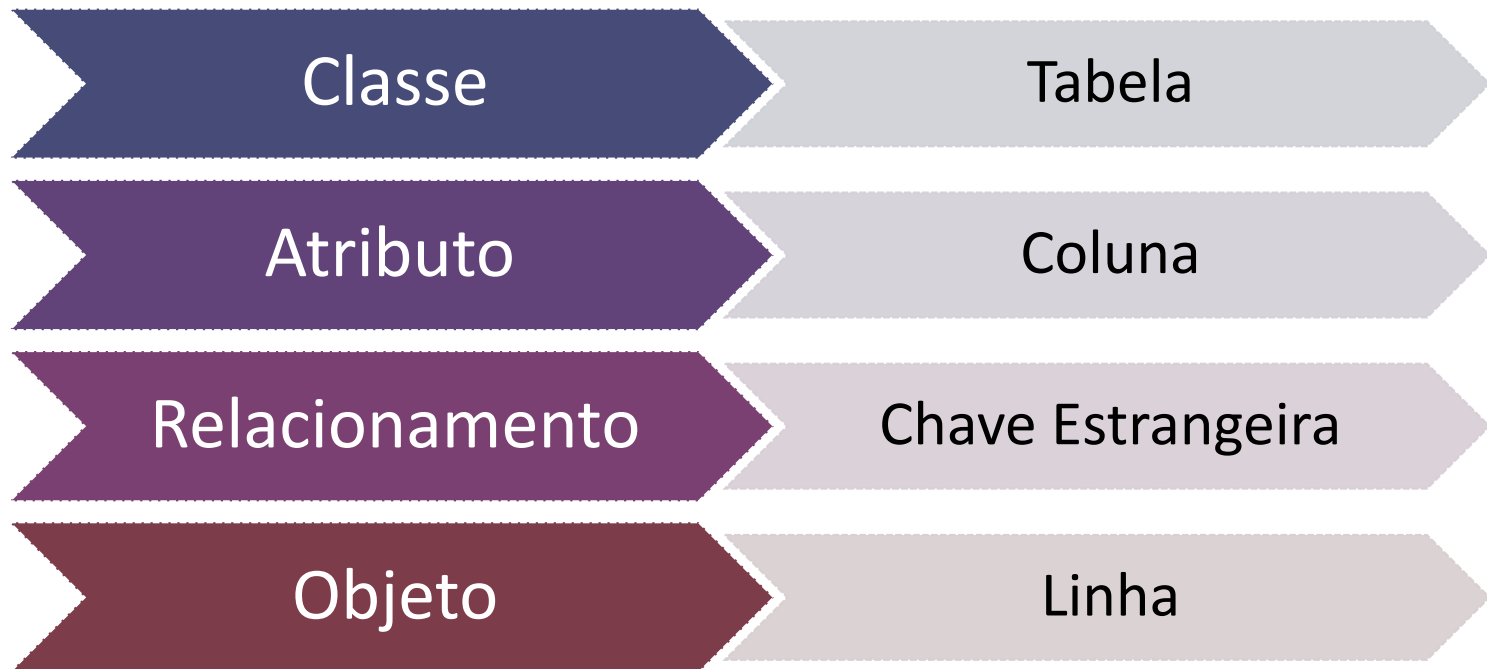
- ▶ JPA – *Java Persistence API*
  - ▶ Vantagens *Frameworks JPA*
    - ▶ Independência de SGBD
    - ▶ Portabilidade de código
    - ▶ Abstração de código SQL
  - ▶ Exemplos de *Frameworks JPA*
    - ▶ Hibernate
    - ▶ Toplink
    - ▶ Kodo
    - ▶ OpenJPA

## Mapeamento Objeto Relacional

---

### ► JPA – *Java Persistence API*

#### ► Mapeamento



#### ► Salvar, consultar, atualizar e excluir objetos do banco de dados

# Mapeamento Objeto Relacional

## ► Como usar JPA - Passo a Passo

### Configurar bibliotecas do projeto

- Provedor JPA + Driver de Conexão JDBC

### Configurar unidade de persistência

- persistence.xml

### Fazer o Mapeamento Objeto Relacional

### Criar classes de gerenciamento de objetos

- EntityManager

# Mapeamento Objeto Relacional

---

## ▶ Como usar JPA - Passo a Passo

### Configurar bibliotecas do projeto

#### ▶ Provedores JPA

##### ▶ Hibernate

□ <http://jpa.hibernate.org>

##### ▶ Driver JDBC MySQL

□ <https://dev.mysql.com/downloads/connector/j/>

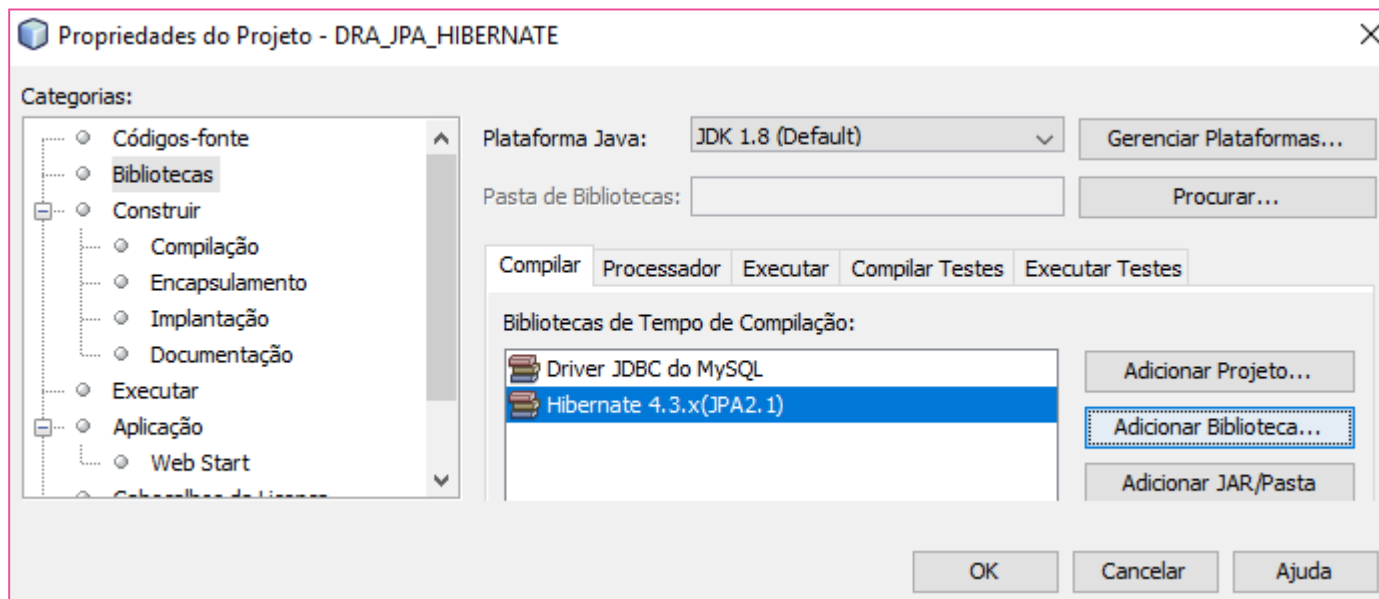


## Mapeamento Objeto Relacional

### ► Como usar JPA - Passo a Passo

#### Configurar bibliotecas do projeto

- Versões mais recentes do Netbeans já vêm com Hibernate e com o driver de conexão JDBC MySQL.



# Mapeamento Objeto Relacional

---

## ▶ Como usar JPA - Passo a Passo

### Configurar unidade de persistência

#### ▶ Unidade de Persistência

##### ▶ Define informações sobre:

- ☐ Provedor do JPA
- ☐ Banco de dados
- ☐ Classes que serão mapeadas como entidades no banco de dados

##### ▶ É representada pelo arquivo “persistence.xml”

- ☐ \*\*Deve ser salvo no pacote META-INF

## Configurar unidade de persistência

```
<persistence-unit name="SysControleAcademicoJPA" transaction-type="RESOURCE_LOCAL">
```

```
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
```

*Define Unidade de Persistência*

```
<properties>
```

```
<property name="hibernate.connection.username" value="root"/>
```

```
<property name="hibernate.connection.password" value="root"/>
```

```
<property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
```

```
<property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>
```

```
<property name="hibernate.hbm2ddl.auto" value="update"/>
```

```
<property name="hibernate.show_sql" value="true"/>
```

```
<property name="hibernate.format_sql" value="true"/>
```

```
</properties>
```

```
</persistence-unit>
```

```
</persistence>
```

## Configurar unidade de persistência

```
<persistence-unit name="SysControleAcademicoJPA" transaction-type="RESOURCE_LOCAL">
```

```
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
```

*Define o provedor  
JPA*

```
<properties>
```

```
<property name="hibernate.connection.username" value="root"/>
```

```
<property name="hibernate.connection.password" value="root"/>
```

```
<property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
```

```
<property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>
```

```
<property name="hibernate.hbm2ddl.auto" value="update"/>
```

```
<property name="hibernate.show_sql" value="true"/>
```

```
<property name="hibernate.format_sql" value="true"/>
```

```
</properties>
```

```
</persistence-unit>
```

```
</persistence>
```

## Configurar unidade de persistência

```
<persistence-unit name="SysControleAcademicoJPA" transaction-type="RESOURCE_LOCAL">
```

```
  <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
```

```
  <properties>
```

```
    <property name="hibernate.connection.username" value="root"/>
```

```
    <property name="hibernate.connection.password" value="root"/>
```

```
    <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
```

```
    <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>
```

```
    <property name="hibernate.hbm2ddl.auto" value="update"/>
```

```
    <property name="hibernate.show_sql" value="true"/>
```

```
    <property name="hibernate.format_sql" value="true"/>
```

```
  </properties>
```

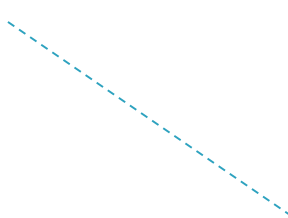
```
</persistence-unit>
```

```
</persistence>
```

*Propriedades do  
BD*

## Configurar unidade de persistência

```
<persistence-unit name="SysControleAcademicoJPA" transaction-type="RESOURCE_LOCAL">  
  
  <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>  
  
  <properties>  
    <property name="hibernate.connection.username" value="root"/>  
    <property name="hibernate.connection.password" value="root"/>  
    <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>  
    <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>  
  
    <property name="hibernate.hbm2ddl.auto" value="update"/>  
  
    <property name="hibernate.show_sql" value="true"/>  
    <property name="hibernate.format_sql" value="true"/>  
  </properties>  
  
</persistence-unit>  
  
</persistence>
```



- `validate` : validar o schema, não faz mudanças no banco de dados.
- `update` : faz update o schema.
- `create` : cria o schema, destruindo dados anteriores.
- `create-drop` : drop o schema quando ao terminar a sessão.

## Configurar unidade de persistência

```
<persistence-unit name="SysControleAcademicoJPA" transaction-type="RESOURCE_LOCAL">

  <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

  <properties>
    <property name="hibernate.connection.username" value="root"/>
    <property name="hibernate.connection.password" value="root"/>
    <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
    <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>

    <property name="hibernate.hbm2ddl.auto" value="update"/>

    <property name="hibernate.show_sql" value="true"/>
    <property name="hibernate.format_sql" value="true"/>
  </properties>

</persistence-unit>

</persistence>
```

*Apresenta no console o SQL gerado pelo Hibernate*

## Configurar unidade de persistência

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

  <persistence-unit name="SysControleAcademicoJPA" transaction-type="RESOURCE_LOCAL">

    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <properties>
      <property name="hibernate.connection.username" value="root"/>
      <property name="hibernate.connection.password" value="root"/>
      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>

      <property name="hibernate.hbm2ddl.auto" value="update"/>

      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```



## Mapeamento Objeto Relacional

---

### ► Como usar JPA - Passo a Passo

Fazer o Mapeamento Objeto Relacional

#### ► Anotações JAVA

@Entity

@Id

@GeneratedValue

## Mapeamento Objeto Relacional

---

### ► Como usar JPA - Passo a Passo

#### Fazer o Mapeamento Objeto Relacional

#### ► Anotações JAVA

@Entity

- Deve aparecer antes do nome da classe que terá os objetos persistidos no banco de dados.
- Classes são mapeadas para tabelas [@Table: opcional]
- Atributos são mapeados para colunas [@Column: opcional]

## Mapeamento Objeto Relacional

---

### ▶ Como usar JPA - Passo a Passo

#### Fazer o Mapeamento Objeto Relacional

#### ▶ Anotações JAVA

@Id

- ▶ Indica qual atributo será mapeado como chave primária.
- ▶ Geralmente atributos mapeados com @Id são do tipo *Long*.

## Mapeamento Objeto Relacional

---

### ► Como usar JPA - Passo a Passo

#### Fazer o Mapeamento Objeto Relacional

#### ► Anotações JAVA

**@GeneratedValue**

- Indica que o valor do atributo que compõe a chave primária deve ser gerado automaticamente pelo banco de dados.
- Geralmente vem acompanhado pela anotação @Id

# Mapeamento Objeto Relacional

## ► Como usar JPA - Passo a Passo

### Fazer o Mapeamento Objeto Relacional

```
package jpa.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Curso {

    @Id
    @GeneratedValue
    private Long idCurso;
    private String sigla;
    private String descricao;

}
```

#### Curso

- idCurso : long
- sigla : String
- descricao : String

## Mapeamento Objeto Relacional

---

### ► Como usar JPA - Passo a Passo

#### Criar classes de gerenciamento de objetos

- Geração automática de tabelas no banco de dados.
  - As tabelas são geradas através de um método estático da classe **Persistence**.
  - Método
    - **createEntityManagerFactory(String persistenceUnit)**
      - ❖ persistenceUnit: unidade de persistência definida no arquivo [persistence.xml](#).

```
EntityManagerFactory factory =  
Persistence.createEntityManagerFactory("SysControleAcademicoJPA");
```

# Mapeamento Objeto Relacional

## ► Como usar JPA - Passo a Passo

### Criar classes de gerenciamento de objetos

#### ► Geração automática de tabelas no banco de dados.

```
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class GeraTabelas {

    public static void main(String[] args) {
        EntityManagerFactory factory =
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");
        factory.close();
    }
}
```

- Em caso de erro de execução (jandex): Importar biblioteca [jandex.jar](#)
- <https://www.youtube.com/watch?v=GRYkZR2SxfU>

## Mapeamento Objeto Relacional

---

### ▶ Como usar JPA - Passo a Passo

#### Criar classes de gerenciamento de objetos

- ▶ Para manipular as entidades da nossa aplicação, devemos utilizar um objeto EntityManager.
- ▶ Responsabilidade do EntityManager
  - ▶ Gerenciar o estado dos objetos
  - ▶ Sincronizar os dados da aplicação com o banco de dados.



## Mapeamento Objeto Relacional

---

### ► Como usar JPA - Passo a Passo

#### Criar classes de gerenciamento de objetos

- Um objeto EntityManager que é obtido através da classe EntityManagerFactory.

```
EntityManagerFactory factory =  
Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
EntityManager manager = factory.createEntityManager();
```

## Mapeamento Objeto Relacional

---

### ► Como usar JPA - Passo a Passo

**Criar classes de gerenciamento de objetos**

- `javax.persistence.EntityManager`
  - Implementa métodos para manipular entidades na aplicação.

*persist*

*find/query*

*merge*

*remove*

# Mapeamento Objeto Relacional

## ► Manipulando Entidades

### ► Inserindo (*persist*)

```
public class InserirCursoJPA {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        manager.persist(new Curso("TECINFO", "Técnico em Informática"));  
  
        manager.getTransaction().begin();  
        manager.getTransaction().commit();  
        manager.close();  
  
        factory.close();  
    }  
}
```

Define Unidade de Persistência

# Mapeamento Objeto Relacional

## ► Manipulando Entidades

### ► Inserindo (*persist*)

```
public class InserirCursoJPA {  
    public static void main(String args[]) {  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        manager.persist(new Curso("TECINFO", "Técnico em Informática"));  
  
        manager.getTransaction().begin();  
        manager.getTransaction().commit();  
        manager.close();  
  
        factory.close();  
    }  
}
```

Cria o EntityManager para manipular entidades

# Mapeamento Objeto Relacional

## ► Manipulando Entidades

### ► Inserindo (*persist*)

```
public class InserirCursoJPA {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        manager.persist(new Curso("TECINFO", "Técnico em Informática"));  
  
        manager.getTransaction().begin();  
        manager.getTransaction().commit();  
        manager.close();  
  
        factory.close();  
    }  
}
```

Hibernate:  
insert  
into  
Curso  
(descricao, sigla)  
values  
(?, ?)

# Mapeamento Objeto Relacional

## ► Manipulando Entidades

### ► Inserindo (*persist*)

```
public class InserindoCursoJPA {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        manager.persist(new Curso("TECINFO", "Técnico em Informática"));  
  
        manager.getTransaction().begin();  
        manager.getTransaction().commit();  
        manager.close();  
  
        factory.close();  
    }  
}
```

*Inicia Transação e Sincroniza com BD*

## Mapeamento Objeto Relacional

---

### ► Manipulando Entidades

#### ► Transações

- As modificações (insert/update/delete) realizadas nos objetos administrados pelo EntityManager são mantidas em memória.
- Para validar essas modificações é necessário iniciar uma transação e sincronizar as modificações com o banco de dados.
  - *getTransaction.begin()*
    - Inicia uma transação.
  - *getTransaction.commit()*
    - Sincroniza as informações com o banco.

## Mapeamento Objeto Relacional

---

### ► Manipulando Entidades

#### ► Atualizando (*merge*)

```
public class AtualizandoCursoJPA {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        Curso curso = new Curso(1, "TECINFOR - Alterado", "Técnico em Informática");  
  
        manager.merge(curso);  
  
        manager.getTransaction().begin();  
        manager.getTransaction().commit();  
        manager.close();  
  
        factory.close();  
    }  
}
```



## Mapeamento Objeto Relacional

---

### ► Manipulando Entidades

#### ► Atualizando (alternativa com *find*)

```
public class AtualizandoComFindCursoJPA {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        Curso curso = manager.find(Curso.class, 1L);  
        curso.setSigla("TECINFOR - Alterado");  
  
        manager.getTransaction().begin();  
        manager.getTransaction().commit();  
        manager.close();  
  
        factory.close();  
    }  
}
```

# Mapeamento Objeto Relacional

---

## ► Manipulando Entidades

### ► Excluindo(remove)

```
public class ExcluindoCurso {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        Curso curso = manager.find(Curso.class, 1L);  
  
        manager.remove(curso);  
  
        manager.getTransaction().begin();  
        manager.getTransaction().commit();  
        manager.close();  
  
        factory.close();  
    }  
}
```

# Mapeamento Objeto Relacional

---

## ► Manipulando Entidades

### ► Buscando por ID (*find*)

```
public class BuscandoPorIDCurso {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        Curso curso = manager.find(Curso.class, 1L);  
  
        System.out.println("Curso: " + curso.getSigla()  
                           + " - " + curso.getDescricao() );  
  
        manager.close();  
  
        factory.close();  
    }  
}
```

# Mapeamento Objeto Relacional

---

## ► Manipulando Entidades

### ► Buscando com JPQL - Java Persistence Query Language

```
public class BuscandoComQueryCurso {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        Query query = manager.createQuery("select c from Curso as c "  
                                         + "where c.sigla LIKE :param ");  
        query.setParameter("param", "%TEC%");  
        List<Curso> listaCursos = query.getResultList();  
  
        for (Curso listaCurso : listaCursos) {  
            System.out.println("-" + listaCurso.getDescricao());  
        }  
  
        manager.close();  
  
        factory.close();  
    }  
}
```

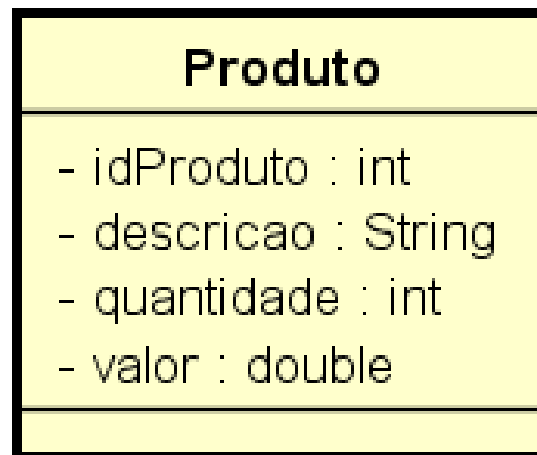
## Mapeamento Objeto Relacional

---

### ► Manipulando Entidades

#### ► Pratique!

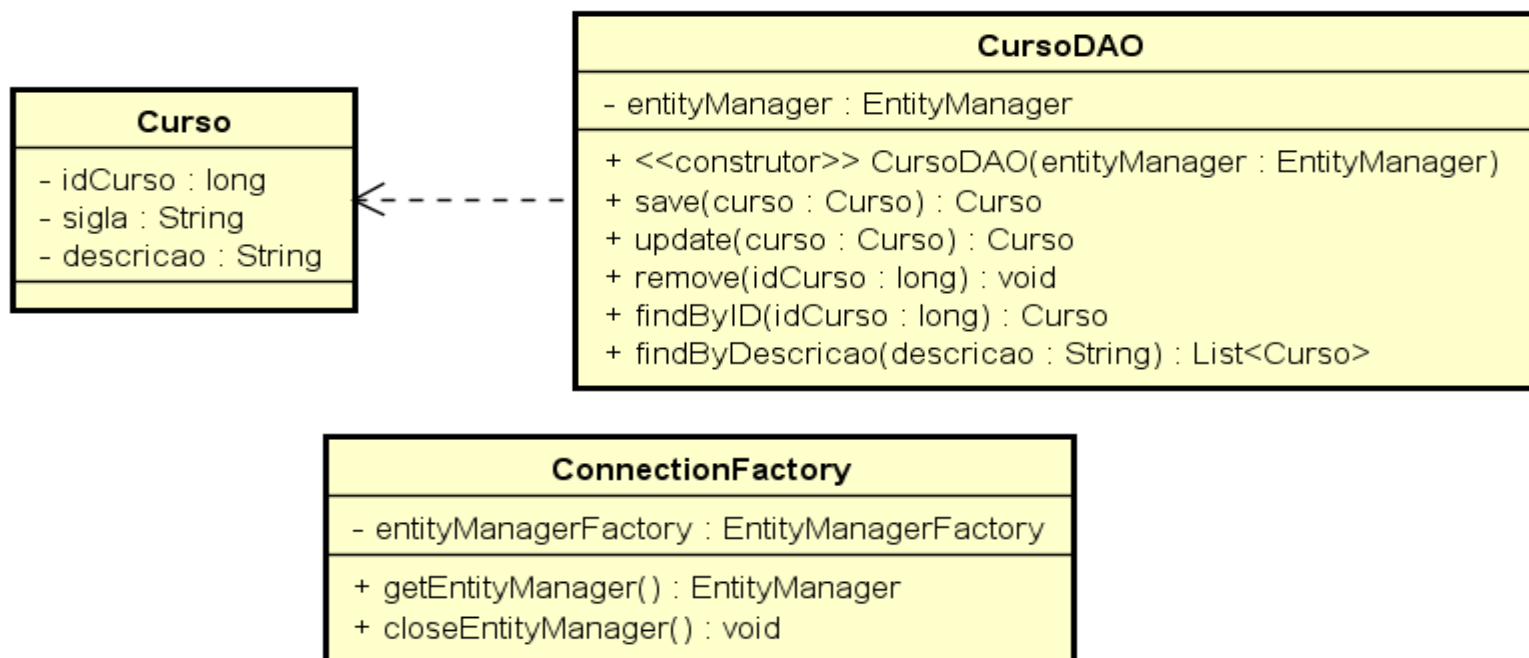
- Utilizando o Hibernate, crie as classes necessárias para garantir o mapeamento objeto-relacional associados à classe **Produto**.



## Mapeamento Objeto Relacional

### ► Manipulando Entidades

#### ► Discussão sobre classe DAO



Código disponível em:

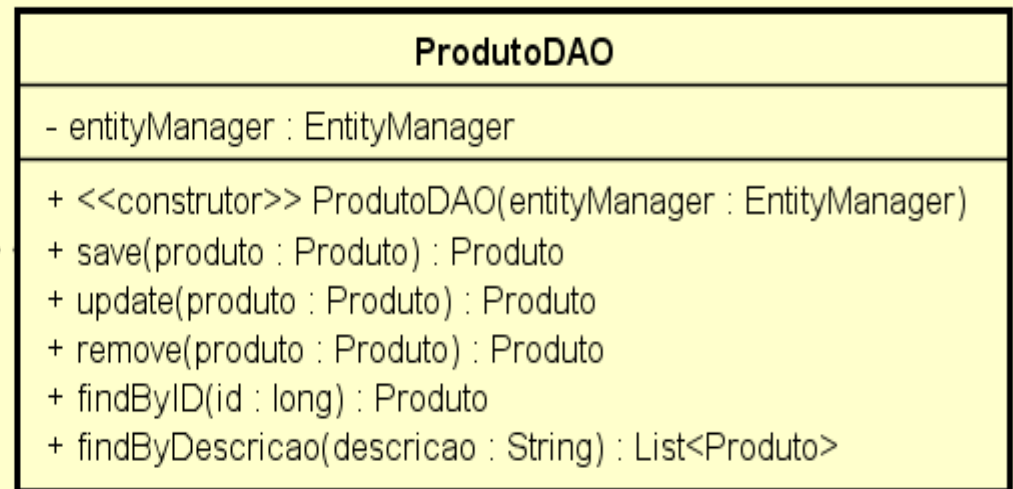
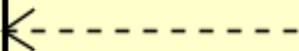
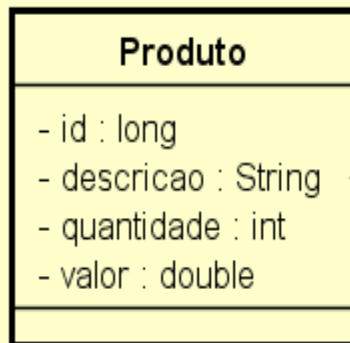
[https://github.com/joyceMiranda/classCodes/tree/master/DRA\\_JPA\\_HIBERNATE](https://github.com/joyceMiranda/classCodes/tree/master/DRA_JPA_HIBERNATE)

## Mapeamento Objeto Relacional

PROGRAMAÊ!

### ► Tarefa de Implementação

- De acordo com o modelo abaixo, implemente as classes aplicando JPA + *Hibernate*.
- Crie uma aplicação com interação de entrada de dados pelo teclado para testar os métodos implementados.



## Mapeamento Objeto Relacional

---

- ▶ Manipulando Entidades
  - ▶ Apresentação do DAO Genérico
    - ▶ Estudo dirigido

<https://www.profissionaisti.com.br/2016/12/design-pattern-criando-uma-classe-dao-generica/>

**Exemplo: Código disponível em:**

[https://github.com/joyceMiranda/classCodes/tree/master/DRA\\_JPA\\_HIBERNATE](https://github.com/joyceMiranda/classCodes/tree/master/DRA_JPA_HIBERNATE)



## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

- Responsabilidades
  - Gerenciar o estado dos objetos
  - Sincronizar os dados da aplicação e do banco de dados
- Estados

Novo  
*(New)*

Administrado  
*(Managed)*

Desvinculado  
*(Detached)*

Removido  
*(Removed)*

# Mapeamento Objeto Relacional

---

## ► *Entity Manager*

### ► Estados

#### ► Novo (*New*)

- ❑ Objeto recém-criado com o comando *new*.
- ❑ Não possui identidade (chave) e não está associado a um *EntityManager*.

#### ► Administrado (*Managed*)

- ❑ Possui identidade e está associado a um *Entity Manager*.
- ❑ A cada sincronização, os dados de um objeto *managed* são atualizados no banco de dados.

## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

#### ► Estados

##### ► Desvinculado (*Detached*)

- Possui uma identidade, mas não está associado a um *EntityManager*.
- O conteúdo desse objeto não é sincronizado com o banco de dados.

##### ► Removido(*Removed*)

- Possui uma identidade e está associado a um *EntityManager*.
- O conteúdo desse objeto será removido do banco de dados quando houver sincronização.

## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

- Sincronização com o banco de dados
  - Propagar no banco de dados as modificações, remoções e inserções de entidades realizadas em memória através de um *EntityManager*.
  - Acontece com objetos nos estados *managed* e *removed*.
  - Só pode ocorrer se uma transação estiver ativa.



## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

- Sincronização com o banco de dados
  - Cada *EntityManager* possui uma única transação associada.
    - *getTransaction()* : recupera a transação associada a um *EntityManager*.
    - *begin()*: ativa uma transação.

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

## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

#### ► Sincronização com o banco de dados

##### □ *commit()*

- Confirma uma transação.
- Sincroniza com o banco de dados.
- Finaliza a transação.

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

## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

#### ► Sincronização com o banco de dados

##### □ *flush()*

- Dispara uma sincronização.
- As alterações no banco de dados não são visíveis a outras transações.
- As alterações só estarão visíveis dentro da própria transação.
- Somente após o *commit()* as alterações ficarão visíveis para as outras transações.

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

## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

#### ► Sincronização com o banco de dados

##### □ *flush()*

- Todas as transações (*update*, *remove* e *persist*) poderão ser desfeitas pelo método *rollback()*.
- O método *rollback()* também finaliza a transação.

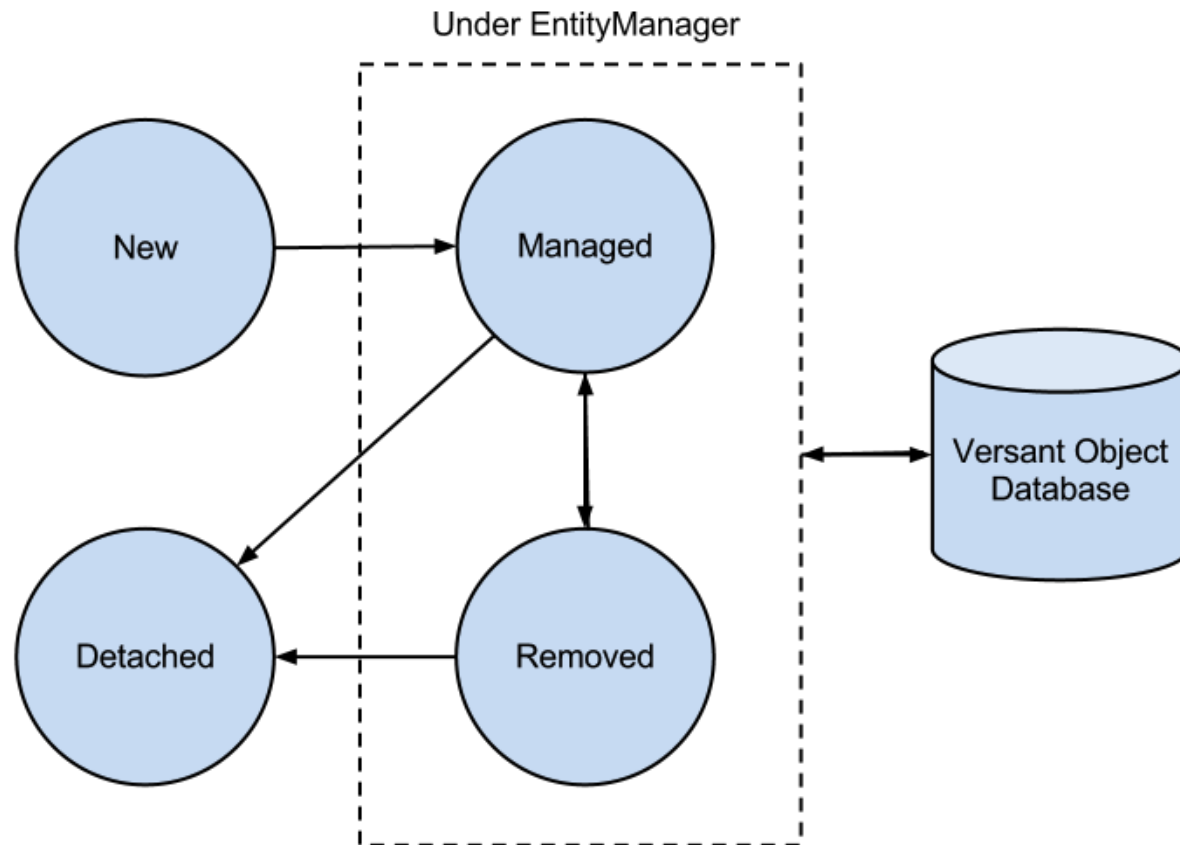
```
manager.getTransaction().begin();  
...  
manager.flush();  
...  
manager.getTransaction().rollback();
```



# Mapeamento Objeto Relacional

## ► *Entity Manager*

### ► Transição entre Estados



## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

#### ► Transição entre Estados

##### ► *New -> Managed*

- Utilização do método *persist()*.

```
manager.getTransaction().begin();  
  
Pessoa p = new Pessoa();  
p.setNome("Rafael Cosentino");  
manager.persist();  
  
manager.getTransaction().commit();
```

## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

#### ► Transição entre Estados

##### ► BD -> *Managed*

- ❑ Objetos recuperados do banco de dados encontram-se no estado *managed*.

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

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

```
1 Query query = manager.createQuery("select p from Pessoa p");  
2 List<Pessoa> lista = query.getResultList();
```

## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

#### ► Transição entre Estados

##### ► *Managed -> Detached*

- *detach()*: desvincula um único objeto.

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

- *clear()*: desvincula todos os objetos administrados por um *Entity Manager*.

```
manager.clear();
```

- *close()*: fecha o *Entity Manager* e desvincula todos os objetos administrados por ele.

```
manager.close();
```

## Mapeamento Objeto Relacional

---

### ► *Entity Manager*

#### ► Transição entre Estados

##### ► *Managed -> Managed*

- O conteúdo de um objeto no estado *managed* pode ficar desatualizado em relação ao banco de dados se alguém ou alguma aplicação alterar os dados na base de dados.
- *refresh()*: atualiza um objeto *managed* com os dados do banco de dados.

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

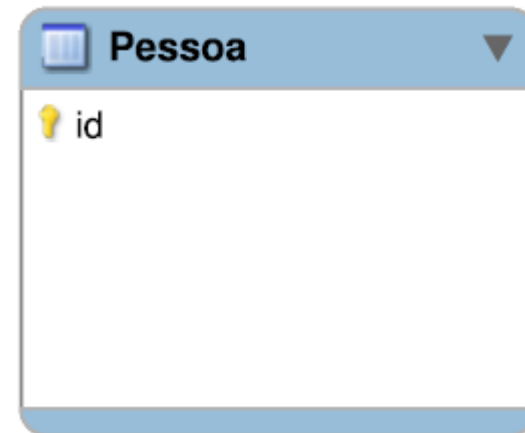
# Mapeamento Objeto Relacional

---

## ► Mapeamento

- Uma revisão
  - **@Entity**: tabela
  - **@Id**: chave primária
  - **@GeneratedValue**: valor AUTO-INCREMENT

```
@Entity
class Pessoa {
    @Id
    @GeneratedValue
    private Long id;
}
```



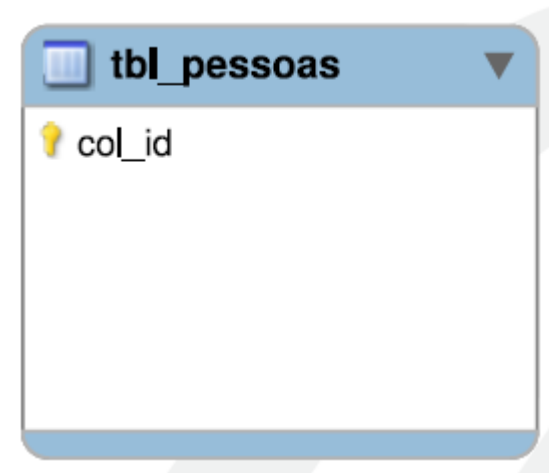
## Mapeamento Objeto Relacional

---

### ► Mapeamento

#### ► Uma revisão

```
@Entity
@Table(name = "tbl_pessoas")
class Pessoa {
    @Id
    @Column(name = "col_id")
    private Long id;
}
```



- As anotações @Table e @Column podem ser usadas para personalizar os nomes das tabelas e das colunas.

# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Definindo restrições: **@Column**

length	Limita a quantidade de caracteres de uma string
nullable	Determina se o campo pode possuir valores null ou não
unique	Determina se uma coluna pode ter valores repetidos ou não
precision	Determina a quantidade de dígitos de um número decimal a serem armazenadas
scale	Determina a quantidade de casas decimais de um número decimal

```
@Entity
class Pessoa {
    @Id
    private Long id;

    @Column(length=30, nullable=false, unique=true)
    private String nome;

    @Column(precision=3, scale=2)
    private BigDecimal altura;
}
```



# Mapeamento Objeto Relacional

---

## ► Mapeamento

- Acontece de forma automática para tipos básicos.
  - Tipos primitivos
    - byte, short, char, int, long, float, double e boolean
  - Classes Wrappers
    - Byte, Short, Character, Integer, Long, Float, Double e Boolean
  - String
  - BigInteger e BigDecimal
  - java.util.Date e java.util.Calendar
  - java.sql.Date, java.sql.Time e java.sql.Timestamp
  - Array de byte ou char
  - Enums
  - Serializables

# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Data e Hora

#### ► @Temporal

- ❑ **TemporalType.DATE**: Armazena apenas a data (dia, mês e ano).
- ❑ **TemporalType.TIME**: Armazena apenas o horário (hora, minuto e segundo).
- ❑ **TemporalType.TIMESTAMP** (Padrão): Armazena a data e o horário.

```
@Entity
class Pessoa {
    @Id
    @GeneratedValue
    private Long id;

    private Calendar nascimento;
}
```

```
@Entity
class Pessoa {
    @Id
    @GeneratedValue
    private Long id;

    @Temporal(TemporalType.DATE)
    private Calendar nascimento;
}
```

# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Objetos grandes (**Large Objects**)

#### ► **@LOB**

- ❑ Imagem, música, texto
- ❑ Aplicado em atributos dos tipos: String, byte[], Byte[], char[] ou Character[]

```
@Entity
class Pessoa {
    @Id
    @GeneratedValue
    private Long id;

    @Lob
    private byte[] avatar;
}
```

# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Dados Transientes

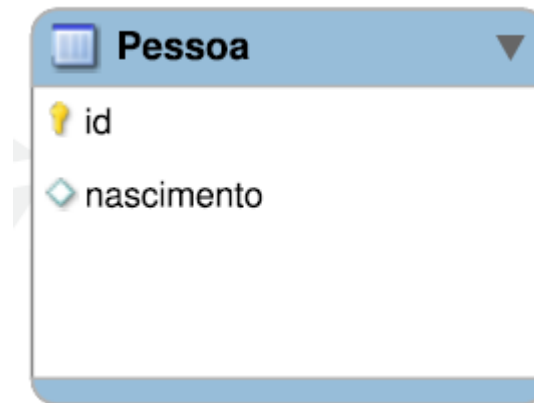
#### ► **@Transient**

- Aplicados em atributos que não serão persistidos no banco de dados

```
@Entity
class Pessoa {
    @Id
    @GeneratedValue
    private Long id;

    @Temporal(TemporalType.DATE)
    private Calendar nascimento;

    @Transient
    private int idade;
}
```



# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Tipos Enumerados

- Tipos enumerados em Java são mapeados para colunas numéricas inteiras no banco de dados.
- Cada elemento de um Enum é associado a um número inteiro.
  - Essa associação é baseada na ordem em que os elementos do Enum são declarados. Primeiro -> 0; Segundo -> 1...

```
@Entity
public class Turma {
    @Id
    @GeneratedValue
    private Long id;

    private Periodo periodo;
}
```

```
public enum Periodo {
    MATUTINO,
    NOTURNO
}
```

# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Tipos Enumerados

#### ► Problema

- A inclusão de um novo período poderia gerar inconsistência em dados já existentes no banco de dados.

```
public enum Período {  
    MATUTINO,  
    NOTURNO  
}
```



```
public enum Período {  
    MATUTINO,  
    VESPERTINO,  
    NOTURNO  
}
```

# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Tipos Enumerados

#### ► Solução

- ❑ **@Enumerated**: faz com que elementos do tipo Enum sejam associados a uma String ao invés de um numero inteiro.

```
@Entity
public class Turma {
    @Id
    @GeneratedValue
    private Long id;

    @Enumerated(EnumType.STRING)
    private Periodo periodo;
}
```

```
public enum Periodo {
    MATUTINO,
    VESPERTINO,
    NOTURNO
}
```

# Mapeamento Objeto Relacional

## ► Mapeamento

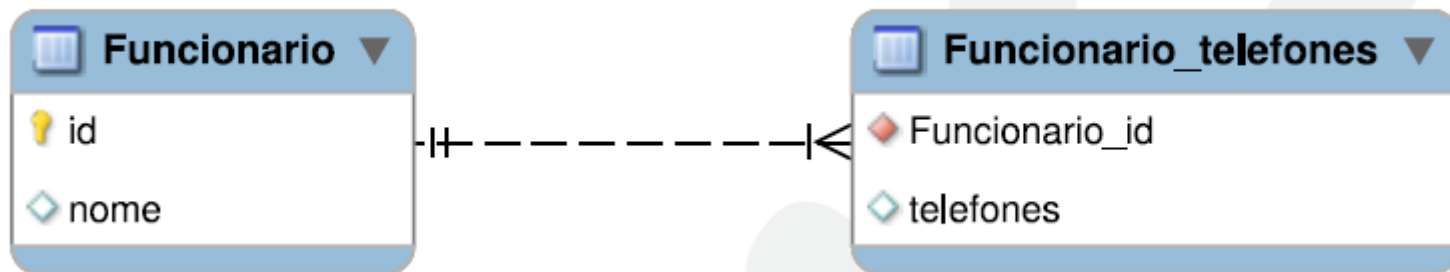
### ► Coleções

#### ► @ElementCollection

```
@Entity
public class Funcionario implements Serializable {

    @Id @GeneratedValue
    private Long id;
    private String nome ;
    @ElementCollection
    private Collection<String> telefones ;

}
```





## Mapeamento Objeto Relacional

---

### ► Mapeamento

#### ► Coleções

- **@CollectionTable**: renomeia tabela resultante do relacionamento.
- **@JoinColumn**: renomeia coluna chave estrangeira.
- **@Column**: renomeia coluna que representa um item da coleção.

```
@Entity
public class Funcionario {

    @Id @GeneratedValue
    private Long id;

    private String nome;

    @ElementCollection
    @CollectionTable(
        name="Telefones_dos_Funcionarios",
        joinColumns=@JoinColumn(name="func_id"))
    @Column(name="telefone")
    private Collection<String> telefones;
}
```

# Mapeamento Objeto Relacional

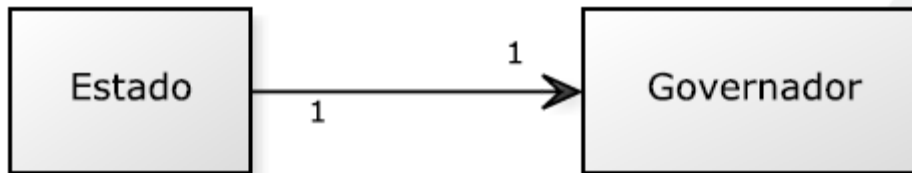
---

## ► Mapeamento

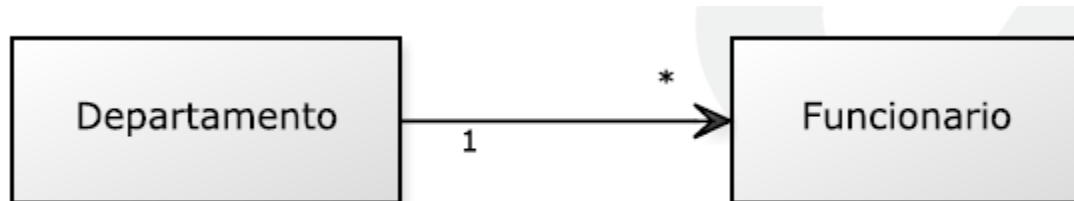
### ► Relacionamentos

#### ► Tipos

##### □ *One To One* (Um pra Um)



##### □ *One To Many* (Um pra Muitos)



# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Relacionamentos

#### ► Tipos

□ *Many To One* (Muitos pra Um)



□ *Many To Many* (Muitos pra Muitos)

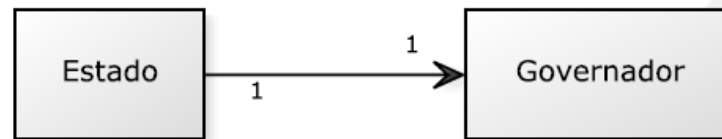


# Mapeamento Objeto Relacional

## ► Mapeamento

### ► Relacionamentos

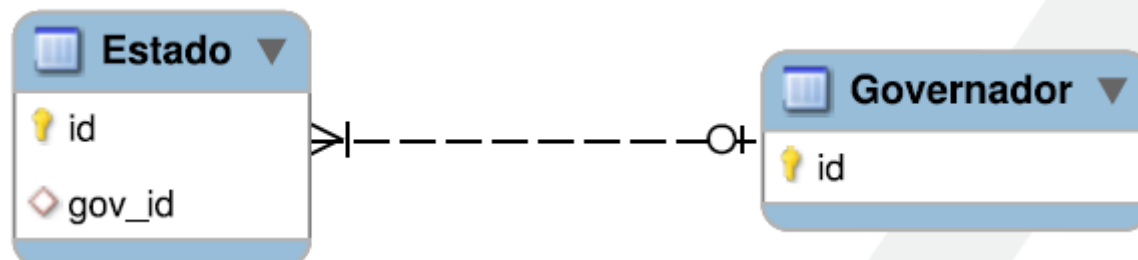
#### ► **One To One (Um pra Um)**



```
@Entity
class Estado {
    @Id
    @GeneratedValue
    private Long id;

    @OneToOne
    @JoinColumn(name="gov_id")
    private Governador governador;
```

```
@Entity
class Governador {
    @Id
    @GeneratedValue
    private Long id;
}
```



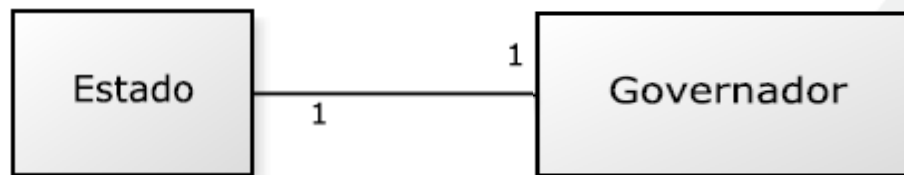
# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Relacionamentos

#### ► Bidirecional: **Sentido 1**



```
@Entity
class Estado {
    @Id
    @GeneratedValue
    private Long id;

    @OneToOne
    private Governador governador;

    // GETTERS E SETTERS
}
```

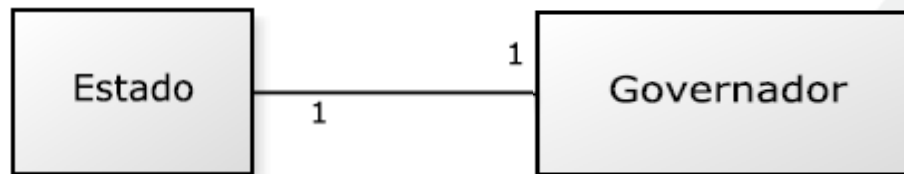
```
Estado e = manager.find(Estado.class, 1L);
Governador g = e.getGovernador();
```

# Mapeamento Objeto Relacional

## ► Mapeamento

### ► Relacionamentos

#### ► Bidirecional: **Sentido 2**



```
@Entity
class Governador {
    @Id
    @GeneratedValue
    private Long id;

    @OneToOne
    private Estado estado;

    // GETTERS E SETTERS
}
```

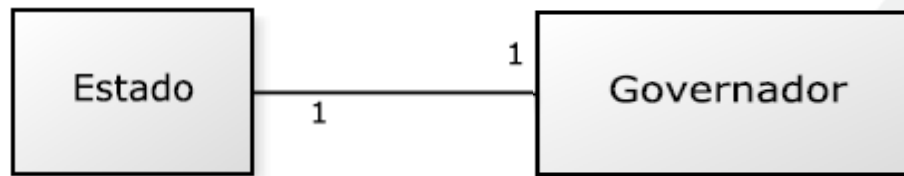
```
Governador g = manager.find(Governador.class, 1L);
Estado e = g.getEstado();
```

## Mapeamento Objeto Relacional

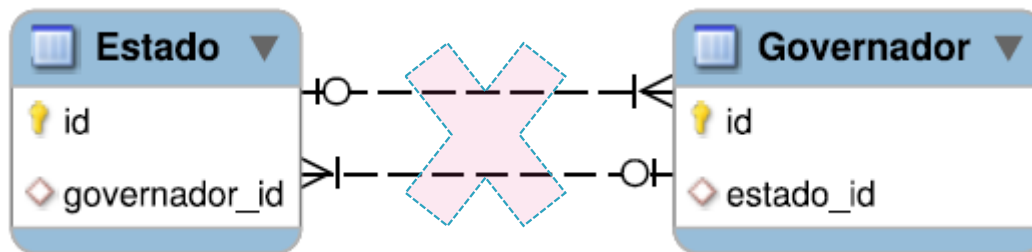
### ► Mapeamento

#### ► Relacionamentos

##### ► Bidirecional



- **Problema:** São criadas duas colunas de relacionamento, quando deveria existir apenas uma.



# Mapeamento Objeto Relacional

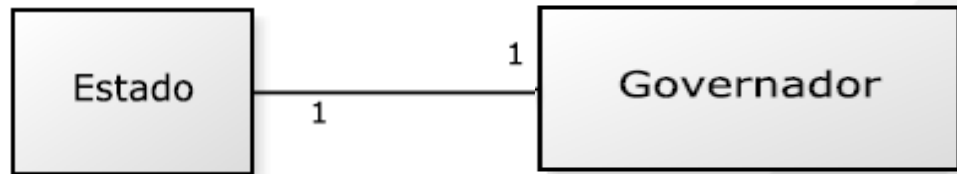
## ► Mapeamento

### ► Relacionamentos

#### ► Bidirecional

##### □ Solução

- **mappedBy**: Indicar em uma das classes que esse relacionamento bidirecional é a junção de dos relacionamentos unidirecionais

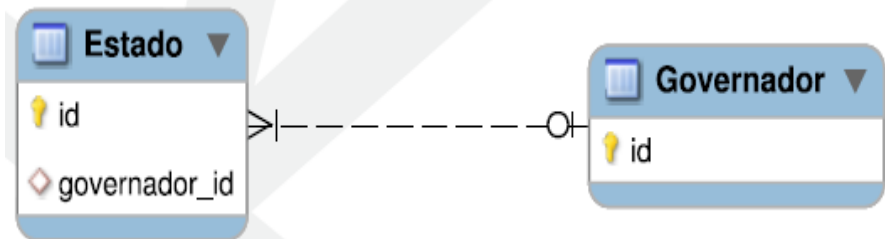


```
@Entity
class Governador {
    @Id
    @GeneratedValue
    private Long id;

    @OneToOne(mappedBy="governador")
    private Estado estado;

    // GETTERS E SETTERS
}
```

*O valor do mappedBy deve ser o nome do atributo que expressa o mesmo relacionamento na outra entidade*





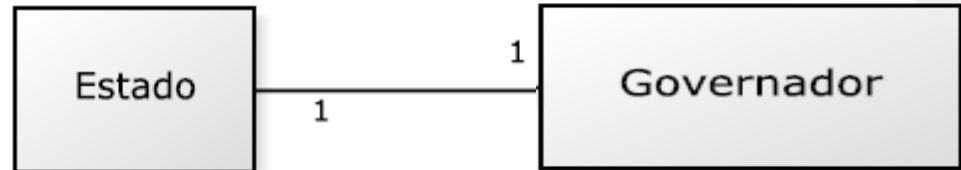
## Mapeamento Objeto Relacional

### ► Mapeamento

#### ► Relacionamentos

##### ► Atributo *Cascade*

- ❑ Operações do *EntityManager* são aplicadas somente ao objeto passado como parâmetro para o método que implementa a operação.
- ❑ Essas operações não são aplicadas aos objetos relacionados ao objeto passado como parâmetro.



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

Governador governador = new Governador();
governador.setNome("Rafael Cosentino");

Estado estado = new Estado();
estado.setNome("São Paulo");

governador.setEstado(estado);
estado.setGovernador(governador);

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

Os dois objetos precisam ser persistidos

```
manager.persist(estado);
manager.persist(governador);
```

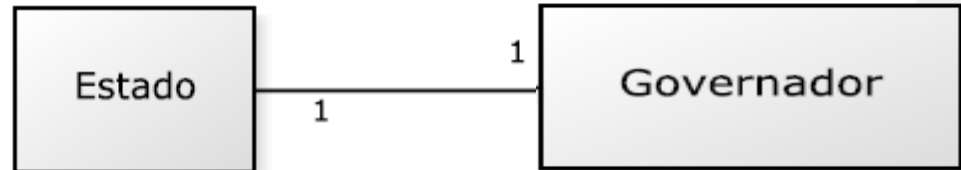
## Mapeamento Objeto Relacional

### ► Mapeamento

#### ► Relacionamentos

##### ► **Atributo *Cascade***

- Podemos configurar a operação para que seja aplicada em cascata nos objetos relacionados ao objeto passado como parâmetro.



```
@Entity
class Estado {
    @Id
    @GeneratedValue
    private Long id;

    @OneToOne(cascade=CascadeType.PERSIST)
    private Governador governador;

    // GETTERS E SETTERS
}
```

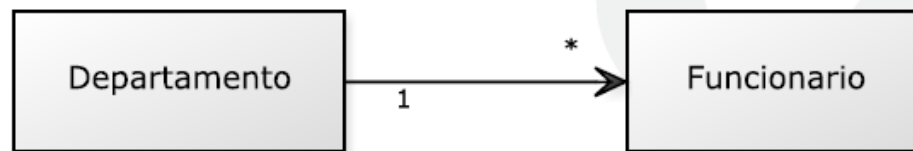
- CascadeType.PERSIST
- CascadeType.DETACH
- CascadeType.MERGE
- CascadeType.REFRESH
- CascadeType.REMOVE
- CascadeType.ALL

# Mapeamento Objeto Relacional

## ► Mapeamento

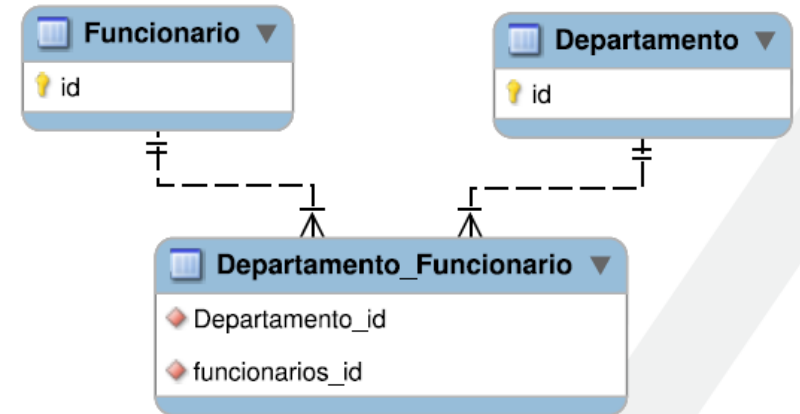
### ► Relacionamentos

#### ► *One To Many* (Um pra Muitos)



```
@Entity
class Departamento {
    @Id
    @GeneratedValue
    private Long id;

    @OneToMany
    private Collection<Funcionario> funcionarios;
}
```



# Mapeamento Objeto Relacional

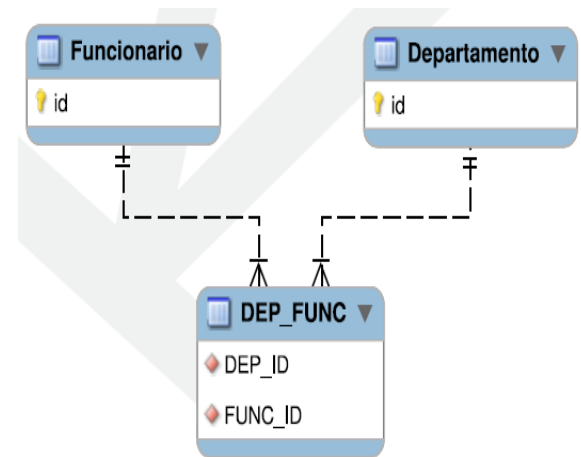
## ► Mapeamento

### ► Relacionamentos

#### ► *One To Many* (Um pra Muitos)

```
@Entity
class Departamento {
    @Id
    @GeneratedValue
    private Long id;

    @OneToMany
    @JoinTable(name="DEP_FUNC",
        joinColumns=@JoinColumn(name="DEP_ID"),
        inverseJoinColumns=@JoinColumn(name="FUNC_ID"))
    private Collection<Funcionario> funcionarios;
}
```

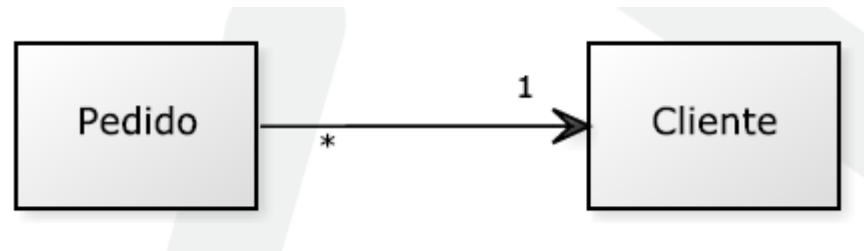


# Mapeamento Objeto Relacional

## ► Mapeamento

### ► Relacionamentos

#### ► *Many To One* (Muitos pra Um)



```
@Entity
class Pedido {
    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    @JoinColumn(name="cli_id")
    private Cliente cliente;
}
```

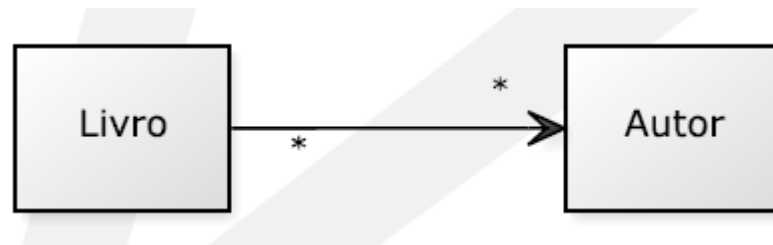


# Mapeamento Objeto Relacional

## ► Mapeamento

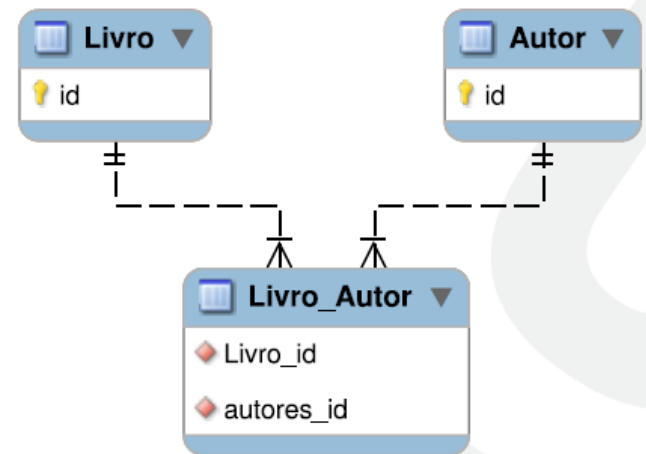
### ► Relacionamentos

#### ► *Many To Many* (Muitos pra Muitos)



```
@Entity
class Livro {
    @Id
    @GeneratedValue
    private Long id;

    @ManyToMany
    private Collection<Autor> autores;
}
```



# Mapeamento Objeto Relacional

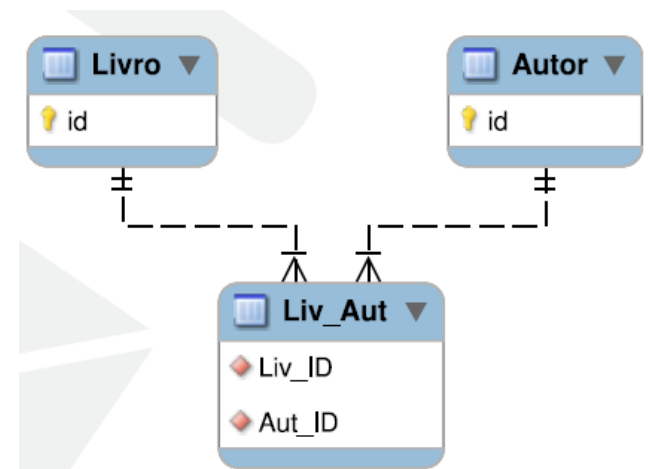
## ► Mapeamento

### ► Relacionamentos

#### ► *Many To Many* (Muitos pra Muitos)

```
@Entity
class Livro {
    @Id
    @GeneratedValue
    private Long id;

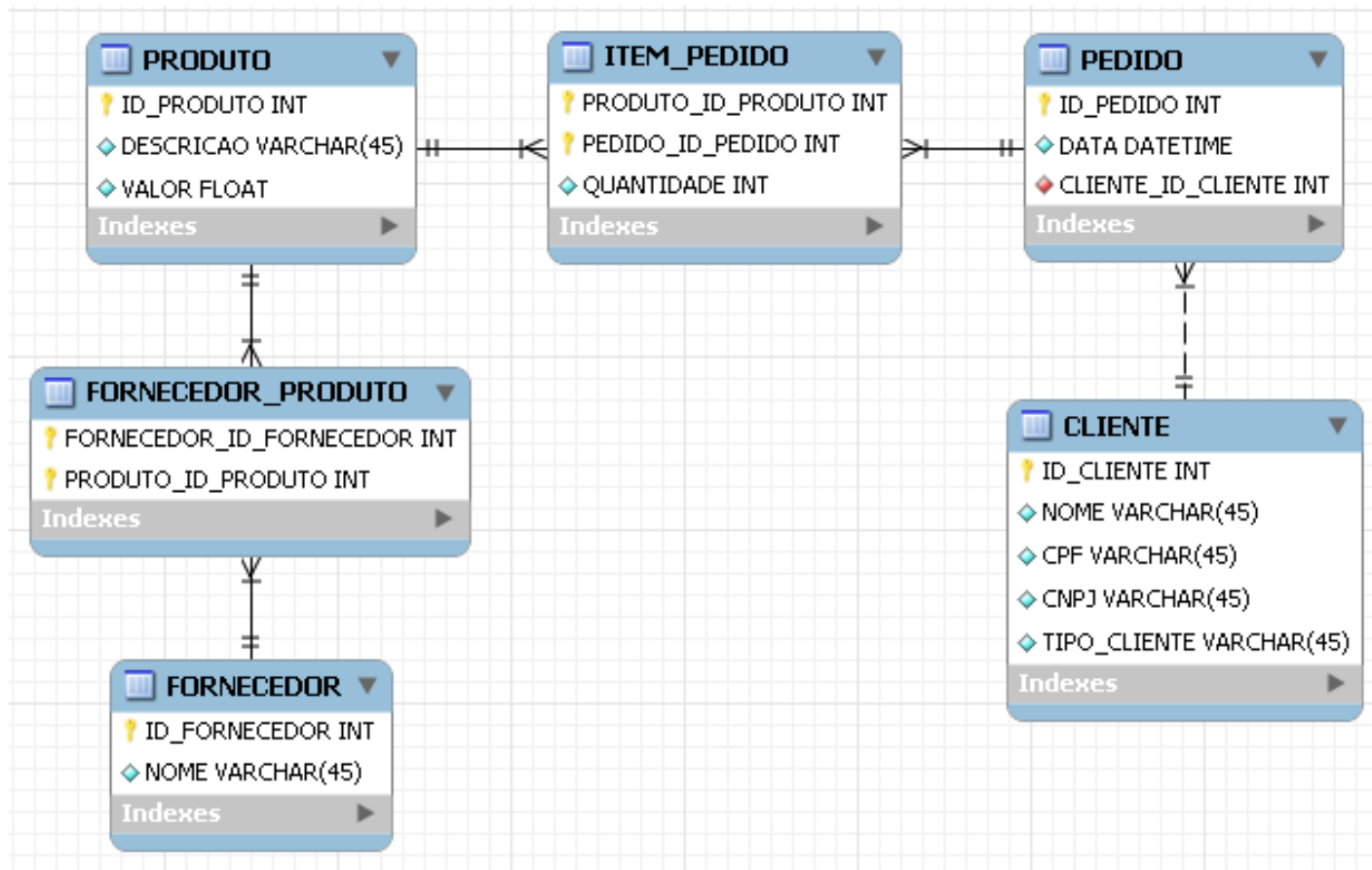
    @ManyToMany
    @JoinTable(name="Liv_Aut",
        joinColumns=@JoinColumn(name="Liv_ID"),
        inverseJoinColumns=@JoinColumn(name="Aut_ID"))
    private Collection<Autor> autores;
}
```



# Mapeamento Objeto Relacional



## ► Tarefa de Implementação



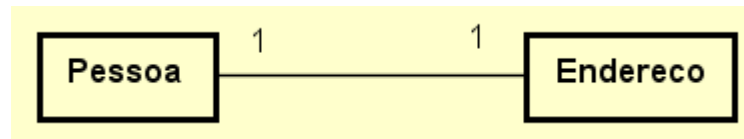


# Mapeamento Objeto Relacional

## ► Mapeamento

### ► Objetos Embutidos

- Nesse caso não queremos que uma tabela Endereço seja gerada, mas que os atributos pertencentes à classe endereço virem colunas na tabela Pessoa.



```
@Entity
class Pessoa {

    @Id
    @GeneratedValue
    private Long id;
    private String nome ;
    @Temporal ( TemporalType . DATE )
    private Calendar nascimento ;
    private Endereco endereco ;

}
```

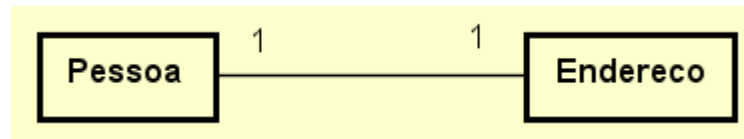
Não se aplica a anotação  
`@OneToOne`

# Mapeamento Objeto Relacional

## ► Mapeamento

### ► Objetos Embutidos

- Nesse caso não queremos que uma tabela Endereço seja gerada, mas que os atributos pertencentes à classe endereço virem colunas na tabela Pessoa.



```
@Embeddable
class Endereco {

    private String pais ;
    private String estado ;
    private String cidade ;
    private String logradouro ;
    private int numero ;
    private String complemento ;
    private int cep ;

}
```

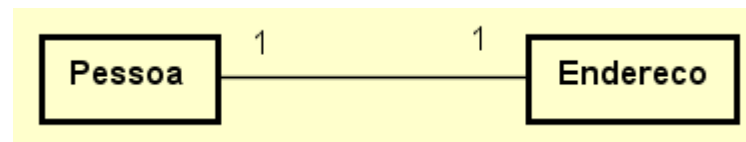
*Substitui @Entity por @Embedable, que indica que é uma classe embutida.*

*Não se deve definir uma chave, pois essa classe não define uma entidade.*

## Mapeamento Objeto Relacional

---

- ▶ Mapeamento
  - ▶ Objetos Embutidos
    - ▶ Resultado



## Mapeamento Objeto Relacional

---

### ► Mapeamento

#### ► Herança

- JPA define três estratégias para o mapeamento de herança.

*Single  
Table*

*Joined*

*Table per  
Class*

# Mapeamento Objeto Relacional

## ► Mapeamento

### ► Herança

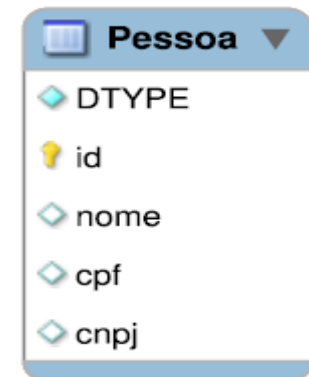
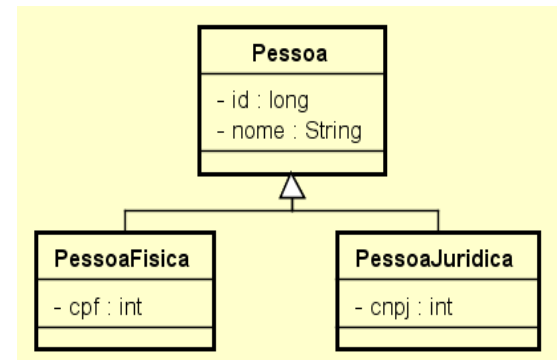
- Estratégia *Single Table*: Uma única tabela é gerada.

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
public class Pessoa {
    @Id @GeneratedValue
    private Long id;

    private String nome;
}
```

```
@Entity
public class PessoaJuridica extends Pessoa{
    private String cnpj;
}
```

```
@Entity
public class PessoaFisica extends Pessoa{
    private String cpf;
}
```



# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Herança

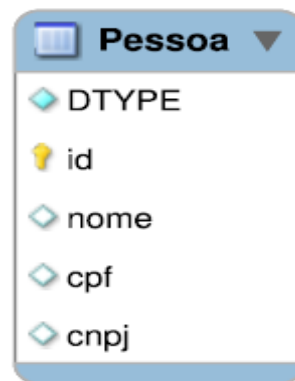
#### ► Estratégia *Single Table*

- Vantagem

- Possibilita melhor desempenho em relação à velocidade das consultas.

- Desvantagem

- Consumo desnecessário de espaço, já que nem todos os campos são utilizados para todos os registros.



# Mapeamento Objeto Relacional

## ► Mapeamento

### ► Herança

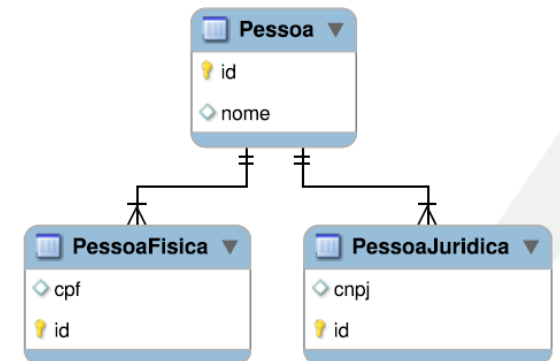
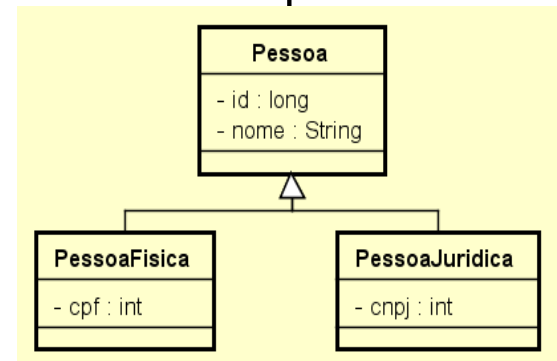
- Estratégia *Joined*: Uma tabela para cada classe é gerada. As tabelas referentes às classes filhas mantêm apenas os atributos específicos.

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Pessoa {
    @Id @GeneratedValue
    private Long id;

    private String nome;
}
```

```
@Entity
public class PessoaJuridica extends Pessoa {
    private String cnpj;
}
```

```
@Entity
public class PessoaFisica extends Pessoa {
    private String cpf;
}
```



# Mapeamento Objeto Relacional

---

## ► Mapeamento

### ► Herança

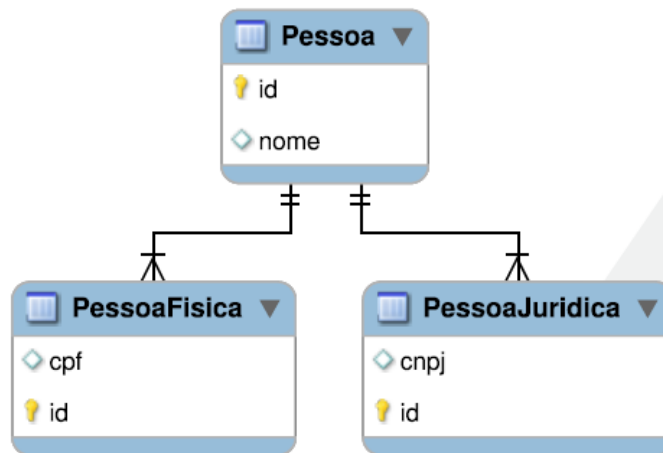
#### ► Estratégia *Joined*

- Vantagem

- Consumo de espaço menor que a *Single Table*.

- Desvantagem

- Consultas mais lentas por ser necessário fazer um *join* para recuperar os dados.





# Mapeamento Objeto Relacional

## ► Mapeamento

### ► Herança

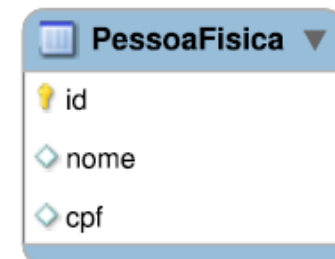
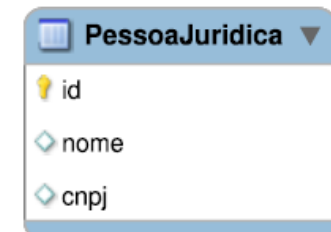
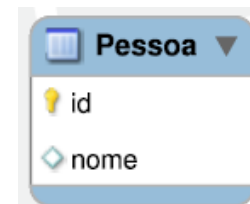
- Estratégia *Table per Class*: Uma tabela para cada classe **concreta** é gerada. Atributos da classe mãe são replicados nas tabelas filhas.

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Pessoa {
    @Id
    private Long id;

    private String nome;
}
```

```
@Entity
public class PessoaJuridica extends Pessoa {
    private String cnpj;
}
```

```
@Entity
public class PessoaFisica extends Pessoa {
    private String cpf;
}
```



## Mapeamento Objeto Relacional

---

### ► Controle de Concorrência

#### ► Fornecer Isolamento

- Conjunto de técnicas que tentam evitar que transações paralelas interfiram umas nas outras.

- ☐ Operações exteriores a uma dada transação jamais verão esta transação em estados intermediários.

- Quando dois *Entity Managers* manipulam objetos da mesma entidade com o o mesmo identificador, pode haver falha de isolamento no banco de dados.

## Mapeamento Objeto Relacional

### ► Controle de Concorrência

- **Problema:** Dependendo da ordem que essas linhas forem executadas, o resultado pode ser diferente.

```
manager1.getTransaction().begin();  
  
Conta x = manager1.find(Conta.class, 1L);  
  
x.setSaldo(x.getSaldo() + 500);  
  
manager1.getTransaction().commit();
```

```
manager2.getTransaction().begin();  
  
Conta y = manager2.find(Conta.class, 1L);  
  
y.setSaldo(y.getSaldo() - 500);  
  
manager2.getTransaction().commit();
```

```
Conta x = manager1.find(Conta.class, 1L) ;//x:saldo=2000  
x.setSaldo(x.getSaldo()+500) ;//x:saldo=2500  
  
Conta y = manager2.find(Conta.class, 1L) ;//y:saldo=2000  
y.setSaldo(y.getSaldo()-500) ;//y:saldo=1500  
  
manager1.getTransaction().commit() ;//Conta1:saldo=2500
```

## Mapeamento Objeto Relacional

---

### ► Controle de Concorrência

#### ► Solução (1) para Controle de Concorrência

##### ► Locking Otimista: Atributo *@Version*

- ❑ Toda vez que um registro for modificado, esse atributo será atualizado.
- ❑ Antes de haver uma nova modificação, a versão do registro do objeto será comparada com a versão do registro do banco de dados.

```
@Entity
public class Conta {

    @Id
    @GeneratedValue
    private Long id;

    private double saldo;

    @Version
    private Long versao;

    // GETTERS AND SETTERS
}
```

## Mapeamento Objeto Relacional

---

### ► Controle de Concorrência

#### ► Solução (2) para Controle de Concorrência

##### ► *Locking* Pessimista

- “Trava” um registro, fazendo com que os outros *Entity Managers* que desejam manipular o mesmo registro tenham que aguardar.

```
Conta x = manager.find(Conta.class, 1L, LockModeType.PESSIMISTIC_WRITE);
```

- Limitação: Pode gerar *deadlock*

```
Conta x = manager1.find(Conta.class, 1L, LockModeType.PESSIMISTIC_WRITE);  
Conta y = manager2.find(Conta.class, 1L, LockModeType.PESSIMISTIC_WRITE);  
manager1.commit();// NUNCA VAI EXECUTAR ESSA LINHA
```

## Mapeamento Objeto Relacional

---

### ► JPQL - *Java Persistence Query Language*

- Recurso para realizar consultas orientadas a objetos.
- Independe dos mecanismos de consulta dos bancos de dados.
- **Consultas Dinâmicas**

```
public void umMetodoQualquer() {  
    String jpql = "SELECT p FROM Pessoa p";  
    Query query = manager.createQuery(jpql);  
}
```

### ► Limitação

- ❑ Pode prejudicar a performance
  - ❑ Toda vez que o método for chamado, o código JPQL dessa consulta será processado pelo provedor JPA.
  - ❑ Alternativa: *Named Query*

## Mapeamento Objeto Relacional

---

### ► JPQL - *Java Persistence Query Language*

#### ► Consultas Dinâmicas

##### ► *@NamedQuery*

- ❑ Só é processada no momento da inicialização da unidade de persistência.
- ❑ Provedores JPA podem mapear *Named Queries* para *Stored Procedures* pré-compiladas no banco de dados para melhorar a performance das consultas.

```
@NamedQuery(name="Pessoa.findAll", query="SELECT p FROM Pessoa p")
class Pessoa {
    ...
}
```

```
@NamedQueries({
    @NamedQuery(name="Pessoa.findAll", query="SELECT p FROM Pessoa p"),
    @NamedQuery(name="Pessoa.count", query="SELECT COUNT(p) FROM Pessoa p")
})
class Pessoa {
    ...
}
```

## Mapeamento Objeto Relacional

---

### ► JPQL - *Java Persistence Query Language*

#### ► Consultas Dinâmicas

##### ► *@NamedQuery*

- Execução

- *createNamedQuery("...")*

```
@NamedQuery(name="Pessoa.findAll", query="SELECT p FROM Pessoa p")  
class Pessoa {  
    ...  
}
```

```
public void listaPessoas() {  
    Query query = manager.createNamedQuery("Pessoa.findAll");  
    List<Pessoa> pessoas = query.getResultList();  
}
```



## Mapeamento Objeto Relacional

---

### ► JPQL - *Java Persistence Query Language*

#### ► Parâmetros

- Tornar consultas genéricas e evitar problemas de *SQL Injection*.
- Utilizar o caracter : seguido do nome do argumento.

```
@NamedQuery(name="Pessoa.findByIdade",  
            query="SELECT p FROM Pessoa p WHERE p.idade > :idade")
```

#### ► Execução

```
public void listaPessoas() {  
    Query query = manager.createNamedQuery("Pessoa.findByIdade");  
    query.setParameter("idade", 18);  
    List<Pessoa> pessoasComMaisDe18 = query.getResultList();  
}
```

## Mapeamento Objeto Relacional

---

### ▶ JPQL - *Java Persistence Query Language*

#### ▶ ***Typed Query***

- ▶ Lista de Objetos Comuns: *getResultList()*

```
String query = "SELECT p FROM Pessoa p";  
Query query = manager.createQuery(query);  
List<Departamento> departamentos = query.getResultList();
```

```
String query = "SELECT p FROM Pessoa p";  
TypedQuery<Pessoa> query = manager.createQuery(query, Pessoa.class);  
List<Pessoa> pessoas = query.getResultList();
```

## Mapeamento Objeto Relacional

---

### ► JPQL - *Java Persistence Query Language*

#### ► ***Typed Query***

- Valores Únicos: *getSingleResult()*

AVG
COUNT
MAX
MIN
SUM

```
String query = "SELECT COUNT(p) FROM Pessoa p";  
TypedQuery<Long> query = manager.createQuery(query, Long.class);  
Long numeroDePessoas = query.getSingleResult();
```

```
String query = "SELECT MAX(p.idade) FROM Pessoa p";  
TypedQuery<Integer> query = manager.createQuery(query, Integer.class);  
Integer maiorIdade = query.getSingleResult();
```

## Mapeamento Objeto Relacional

---

### ► JPQL - *Java Persistence Query Language*

#### ► Resultados Especiais

##### ► **List<Object[]>**

```
"SELECT f.nome, f.departamento.nome FROM Funcionario f";
```

```
String query = "SELECT f.nome, f.departamento.nome FROM Funcionario f";  
Query query = manager.createQuery(query);  
List<Object[]> lista = query.getResultList();  
  
for(Object[] tupla : lista) {  
    System.out.println("Funcionário: " + tupla[0]);  
    System.out.println("Departamento: " + tupla[1]);  
}
```

## Mapeamento Objeto Relacional

---

### ► JPQL - *Java Persistence Query Language*

#### ► Resultados Especiais

##### ► Operador NEW

```
class FuncionarioDepartamento {  
    private String funcionarioNome;  
    private String departamentoNome;  
  
    public FuncionarioDepartamento(String funcionarioNome, String departamentoNome) {  
        this.funcionarioNome = funcionarioNome;  
        this.departamentoNome = departamentoNome;  
    }  
  
    // GETTERS E SETTERS  
}
```

```
String query = "SELECT NEW resultado.FuncionarioDepartamento(f.nome,  
    f.departamento.nome) FROM Funcionario f";  
  
Query query = manager.createQuery(query);  
  
List<FuncionarioDepartamento> resultados = query.getResultList();  
  
for(FuncionarioDepartamento resultado : resultados) {  
    System.out.println("Funcionário: " + resultado.getFuncionarioNome());  
    System.out.println("Departamento: " + resultado.getDepartamentoNome());  
}
```

## Mapeamento Objeto Relacional

---

### ► JPQL - *Java Persistence Query Language*

#### ► *Stored Procedure*

```
@NamedStoredProcedureQuery(  
    name = "BuscaProdutos",  
    resultClasses = Produto.class,  
    procedureName = "BUSCA_PRODUTOS",  
    parameters = {  
        @StoredProcedureParameter(  
            mode=ParameterMode.IN,  
            name="PRECO_MINIMO",  
            type=Double.class)  
    }  
)  
@Entity  
public class Produto {  
    ...  
}
```

```
StoredProcedureQuery query =  
    manager.createNamedStoredProcedureQuery("BuscaProdutos");  
query.setParameter("PRECO_MINIMO", 1000.0);  
List<Produto> produtos = query.getResultList();
```