

Desenvolvimento Rápido de Aplicações

Mapeamento Objeto Relacional

Profa. Joyce Miranda

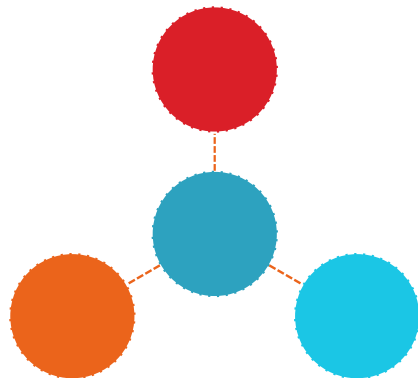
Mapeamento Objeto Relacional

► 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.



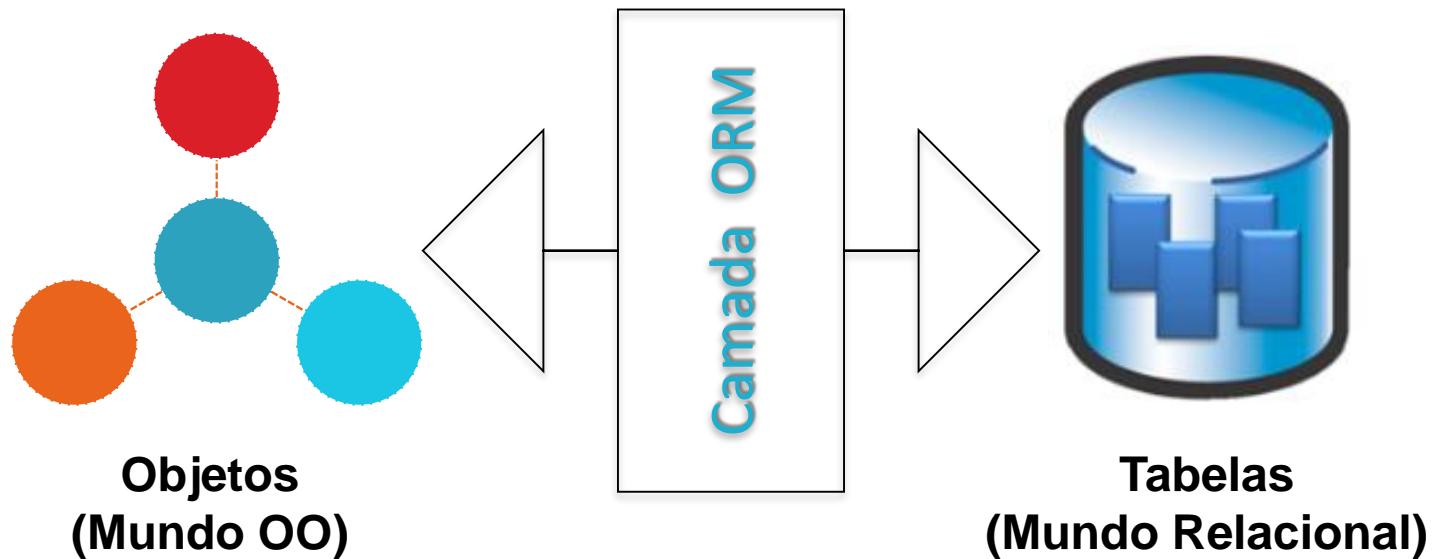
**Objetos
(Mundo OO)**



**Tabelas
(Mundo Relacional)**

Mapeamento Objeto Relacional

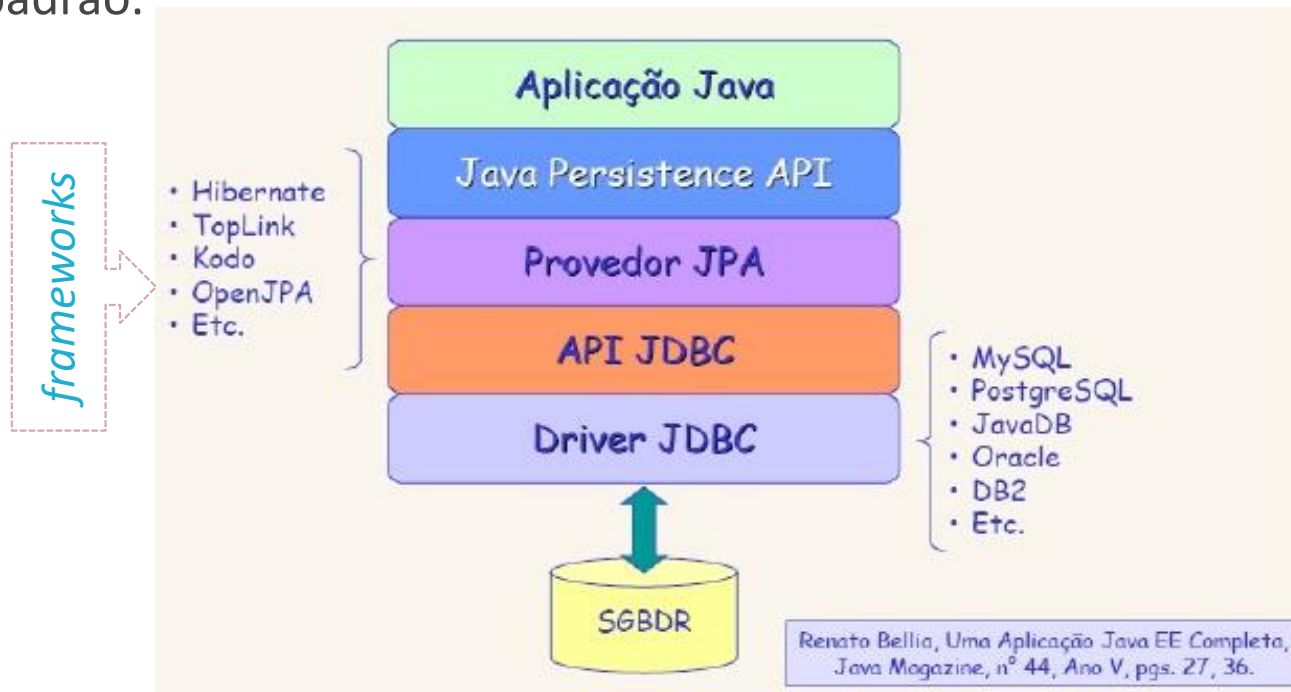
- ▶ Ferramentas ORM (*Object Relational Mapping*)
 - ▶ Representam objetos de maneira relacional na gravação do banco de dados, e conseguem fazer o caminho inverso sem perder informação.



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.



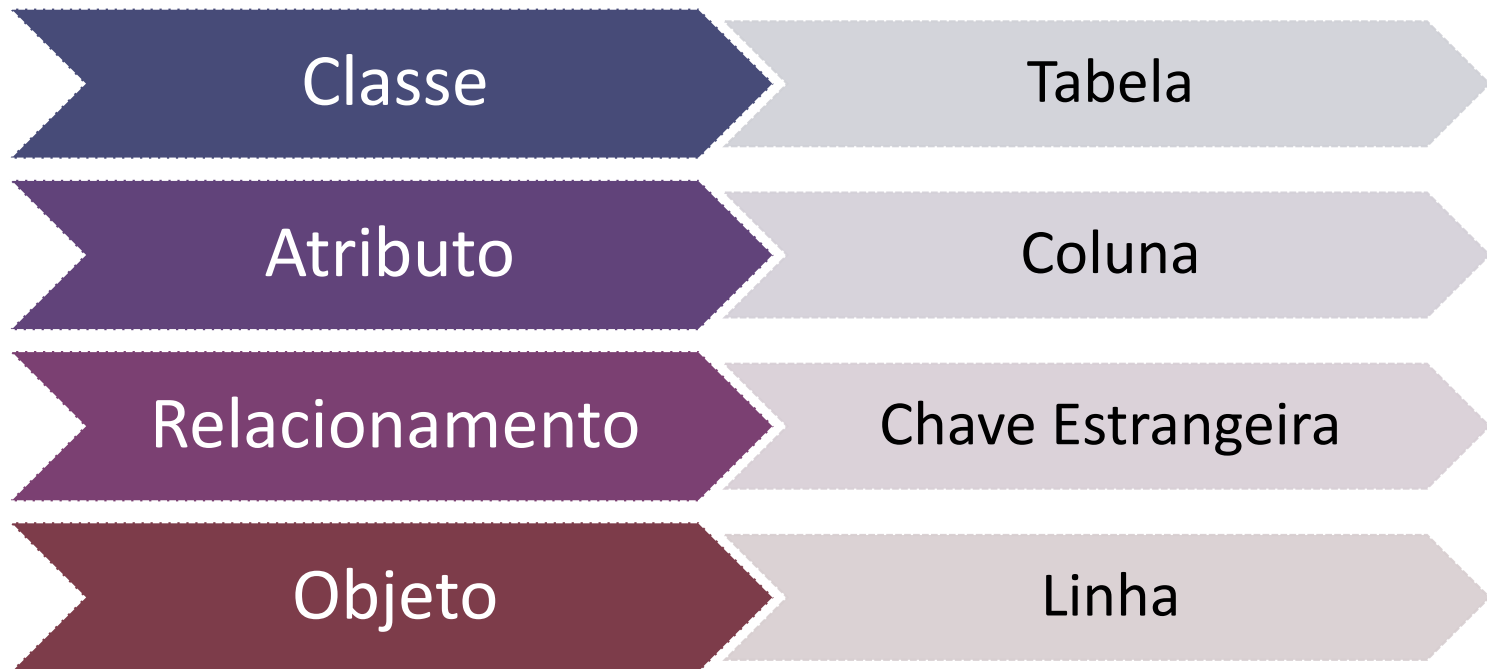
Mapeamento Objeto Relacional

- ▶ JPA – *Java Persistence API*
 - ▶ Vantagens *Frameworks JPA*
 - ▶ Independência de SGBD
 - ▶ Abstração de código SQL
 - ▶ Portabilidade de código
 - ▶ 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

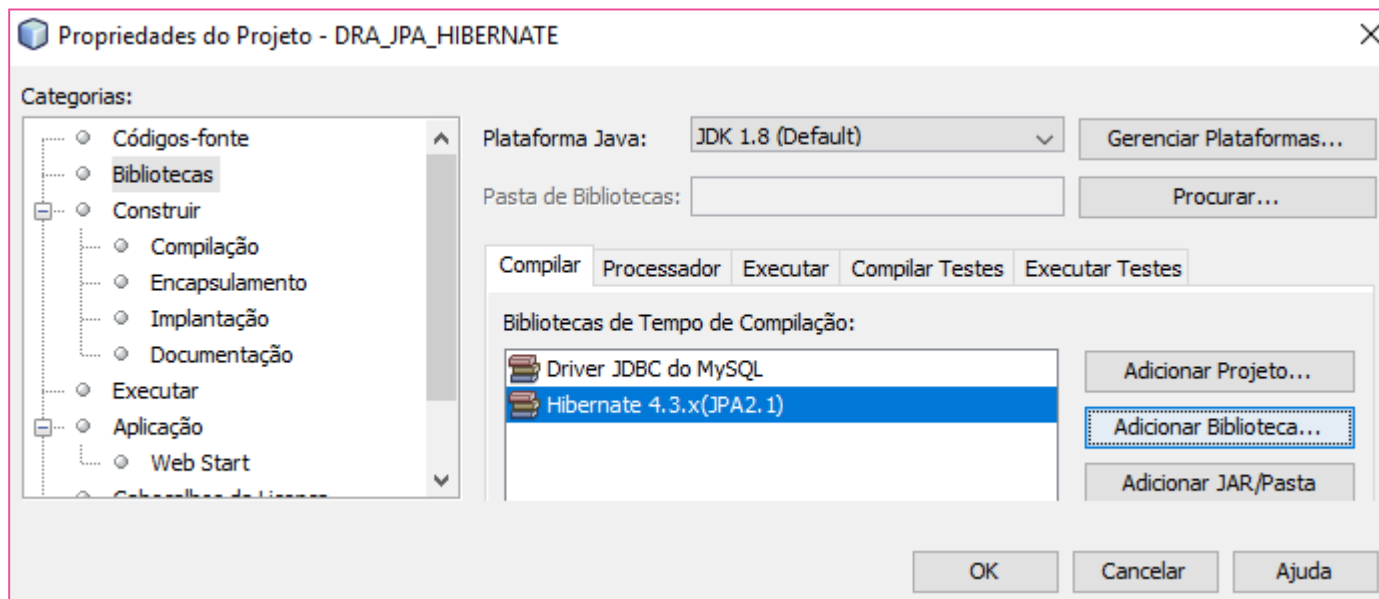
□ <https://dev.mysql.com/downloads/connector/j/>

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

Configurar bibliotecas do projeto

▶ Pratique!

- ▶ No Netbeans, crie o projeto JAVA “SysControleAcademico”.
- ▶ Adicione ao projeto as bibliotecas:
 - ▶ Driver JDBC do MySQL
 - ▶ Hibernate (JPA)
 - ▶ Jandex (arquivo .jar externo)
 - Processa anotações JAVA
- ▶ No MySQL, crie o Banco de Dados “sysControleAcademico”.

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

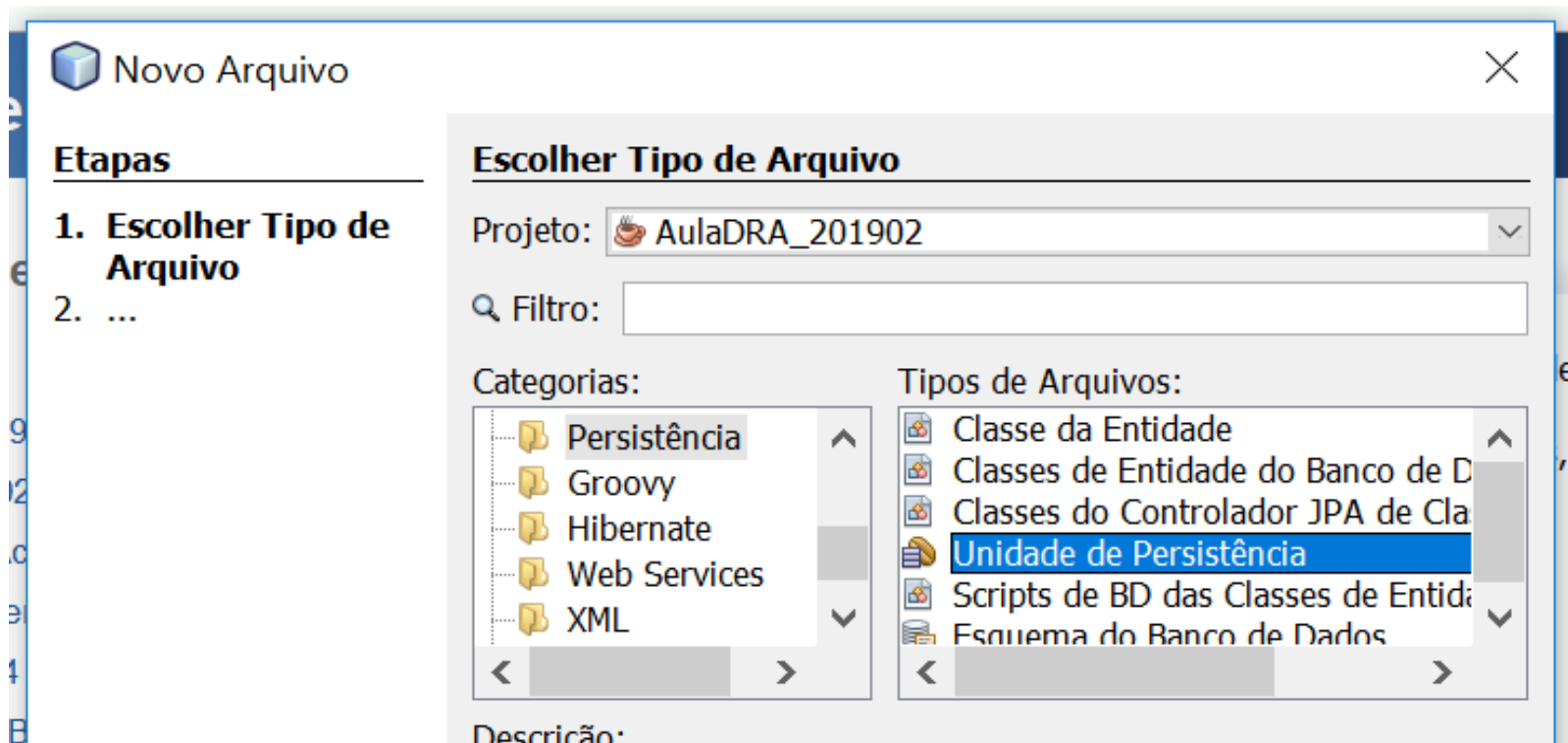
▶ É representada pelo arquivo “persistence.xml”

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

Mapeamento Objeto Relacional

► Como usar JPA - Passo a Passo

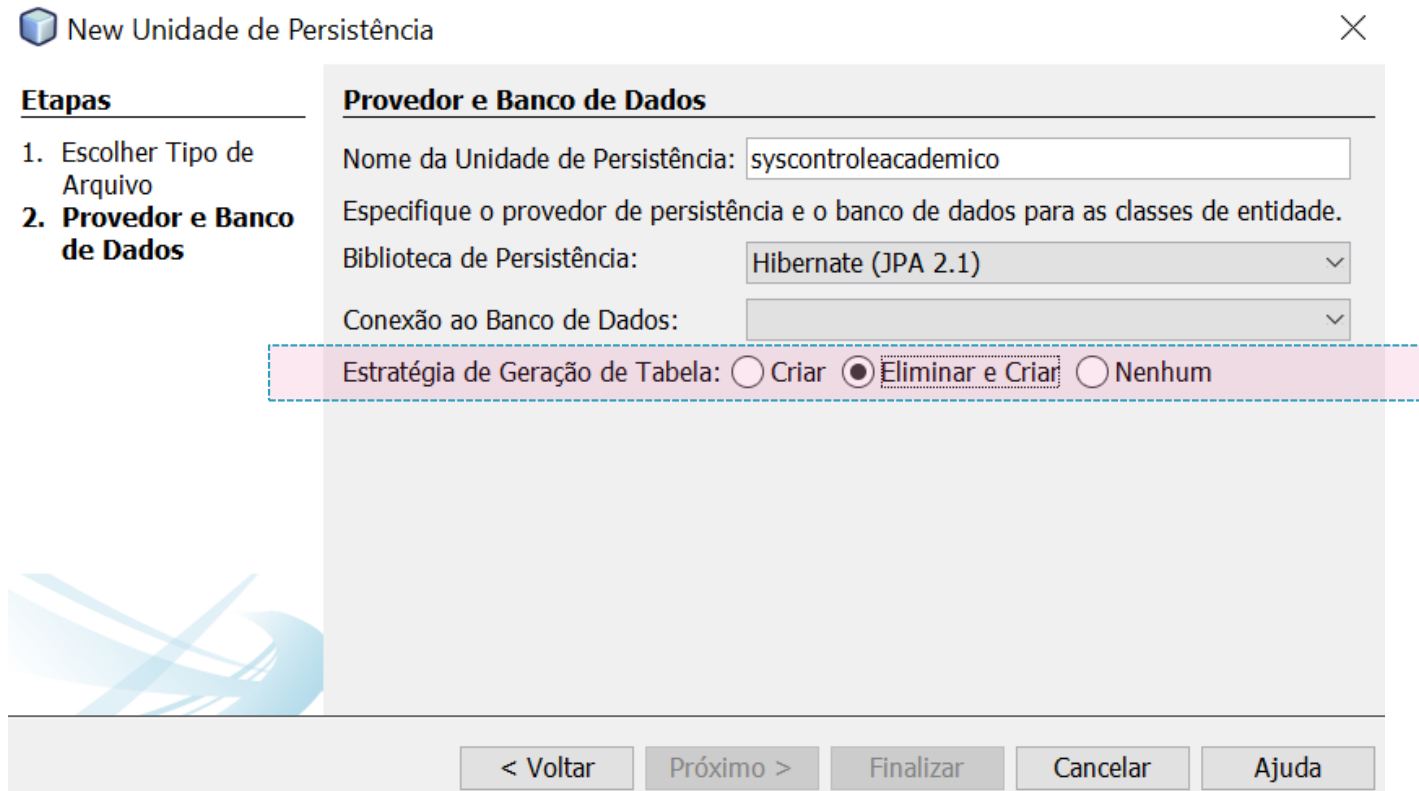
Configurar unidade de persistência



Mapeamento Objeto Relacional

► Como usar JPA - Passo a Passo

Configurar unidade de persistência



New Unidade de Persistência

Etapas

1. Escolher Tipo de Arquivo
2. **Provedor e Banco de Dados**

Provedor e Banco de Dados

Nome da Unidade de Persistência: syscontroleacademico

Especifique o provedor de persistência e o banco de dados para as classes de entidade.

Biblioteca de Persistência: Hibernate (JPA 2.1)

Conexão ao Banco de Dados:

Estratégia de Geração de Tabela: ☐ Criar ☒ Eliminar e Criar ☐ Nenhum

< Voltar Próximo > Finalizar Cancelar Ajuda

Mapeamento Objeto Relacional

► Como usar JPA - Passo a Passo

Configurar unidade de persistência

New Unidade de Persistência

Etapas

1. Escolher Tipo de Arquivo
2. **Provedor e Banco de Dados**

Provedor e Banco de Dados

Nome da Unidade de Persistência: syscontroleacademico

Especifique o provedor de persistência e o banco de dados para as classes de entidade.

Biblioteca de Persistência: Hibernate (JPA 2.1)

Conexão ao Banco de Dados: Nova Conexão de Banco de Dados...

Estratégia de Geração de Tabela: ☐ Criar ☒ Eliminar e Criar ☐ Nenhum

Assistente de Nova Conexão

Localizar Driver

Driver: MySQL (Driver Connector/J)

Arquivo(s) de driver: alho\mysql-connector-java-5.1.47.jar

Adicionar... Remove

< Voltar Próximo >

Mapeamento Objeto Relacional

► Como usar JPA - Passo a Passo

Configurar unidade de persistência

Assistente de Nova Conexão

Personalizar Conexão

Nome do Driver: MySQL (Driver Connector/J)

Host: localhost Porta: 3306

Banco de dados: syscontroleacademico

Nome do Usuário: root

Senha:

☐ Lembrar senha

Propriedades da Conexão Testar Conexão

JDBC URL: jdbc:mysql://localhost:3306/syscontroleacademico?useSSL=false

Conexão Bem-sucedida.

Configurar unidade de persistência

*Define
Unidade de
Persistência*

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

```
<provider>org.hibernate.ejb.HibernatePersistence</provider>
```

```
<properties>
```

```
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico?useSSL=false"/>
```

```
<property name="javax.persistence.jdbc.user" value="root"/>
```

```
<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
```

```
<property name="javax.persistence.jdbc.password" value="root"/>
```

```
<property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
```

```
<property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
```

```
</properties>
```

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


Configurar unidade de persistência

*Define o provedor
JPA*

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

```
<provider>org.hibernate.ejb.HibernatePersistence</provider>
```

```
<properties>
```

```
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico?useSSL=false"/>
```

```
<property name="javax.persistence.jdbc.user" value="root"/>
```

```
<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
```

```
<property name="javax.persistence.jdbc.password" value="root"/>
```

```
<property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
```

```
<property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
```

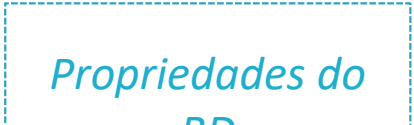
```
</properties>
```

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

Configurar unidade de persistência

```
<persistence-unit name="syscontroleacademico" transaction-type="RESOURCE_LOCAL">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>

  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico?useSSL=false"/>
    <property name="javax.persistence.jdbc.user" value="root"/>
    <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
    <property name="javax.persistence.jdbc.password" value="root"/>
    <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
    <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
  </properties>
</persistence-unit>
```



*Propriedades do
BD*

Configurar unidade de persistência

```
<persistence-unit name="syscontroleacademico" transaction-type="RESOURCE_LOCAL">  
  <provider>org.hibernate.ejb.HibernatePersistence</provider>  
  
  <properties>  
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico?useSSL=false"/>  
    <property name="javax.persistence.jdbc.user" value="root"/>  
    <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>  
    <property name="javax.persistence.jdbc.password" value="root"/>  
    <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>  
    <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>  
  </properties>  
</persistence-unit>
```

*Esquema de
Criação do BD*

Configurar unidade de persistência

Table 37-3 Schema Creation Actions

Setting	Description
<code>none</code>	No schema creation or deletion will take place.
<code>create</code>	The provider will create the database artifacts on application deployment. The artifacts will remain unchanged after application redeployment.
<code>drop-and-create</code>	Any artifacts in the database will be deleted, and the provider will create the database artifacts on deployment.
<code>drop</code>	Any artifacts in the database will be deleted on application deployment.



Configurar unidade de persistência

Adicionar Propriedade

Nome:

Valor:

OK Cancelar

Adicionar Propriedade

Nome:

Valor:

OK Cancelar

Adicionar Propriedade

Nome:

Valor:

OK Cancelar

Configurar unidade de persistência

```
<properties>
  <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico?useSSL=false"/>
  <property name="javax.persistence.jdbc.user" value="root"/>
  <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
  <property name="javax.persistence.jdbc.password" value="root"/>
  <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
  <property name="hibernate.hbm2ddl.auto" value="update"/>
  <property name="hibernate.show_sql" value="true"/>
  <property name="hibernate.format_sql" value="true"/>
</properties>
```

*Atualiza Schema
Mostra SQL
Formata SQL*

Configurar unidade de persistência

Enum Constant Summary

Enum Constants

Enum Constant and Description

CREATE

Database dropping will be generated followed by database creation.

CREATE_DROP

Drop the schema and recreate it on SessionFactory startup.

CREATE_ONLY

Database creation will be generated.

DROP

Database dropping will be generated.

NONE

No action will be performed.

UPDATE

"update" (Hibernate only) - update (alter) the database schema

VALIDATE

"validate" (Hibernate only) - validate the database schema

Mapeamento Objeto Relacional

Configurar unidade de persistência

► Pratique!

- Para o projeto “SysControleAcademico”, crie e configure uma nova **Unidade de Persistência** para o Banco de dados “SysControleAcademico”.

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 da classe **Persistence**.
 - Método Estático

□ **createEntityManagerFactory(String persistenceUnit)**

❖ persistenceUnit: unidade de persistência definida no arquivo [persistence.xml](#).

```
EntityManagerFactory factory =  
    Persistence  
        .createEntityManagerFactory(  
            "syscontroleacademico");  
factory.close();
```

Mapeamento Objeto Relacional

► Como usar JPA - Passo a Passo

Criar classes de gerenciamento de objetos

► Geração automática de tabelas no banco de dados.

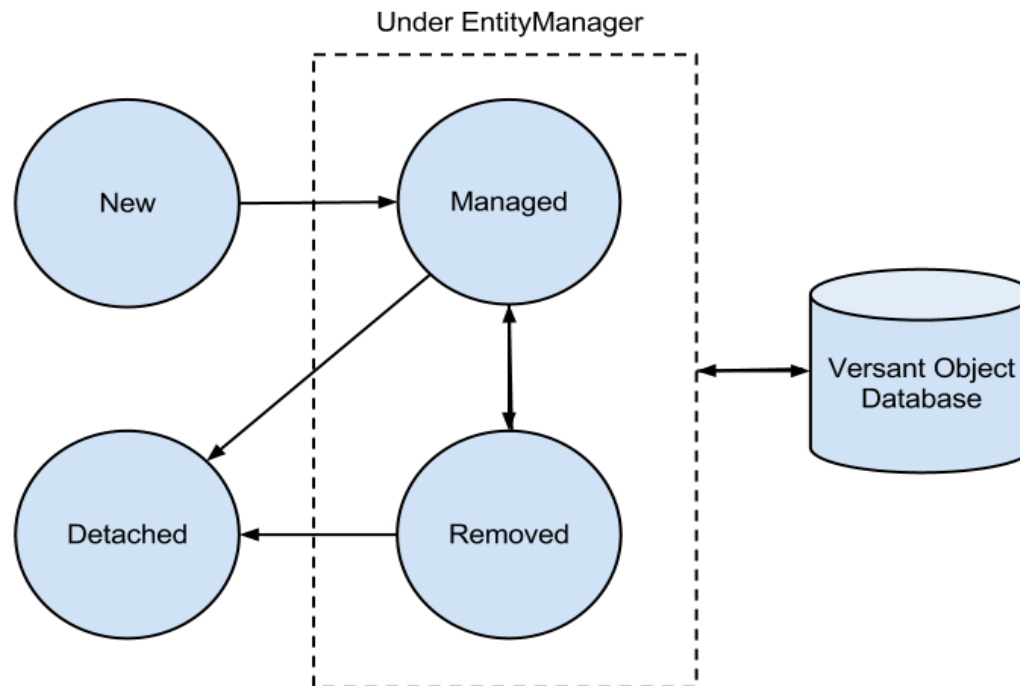
```
public class CreateTables {  
  
    public static void main(String[] args) {  
        EntityManagerFactory factory =  
            Persistence  
                .createEntityManagerFactory(  
                    "syscontroleacademico");  
        factory.close();  
    }  
}
```

Mapeamento Objeto Relacional

► *EntityManager*

► Responsabilidades

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



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

► Transações

- As modificações (persist/merge/remove) 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

► 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 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();  
    }  
}
```

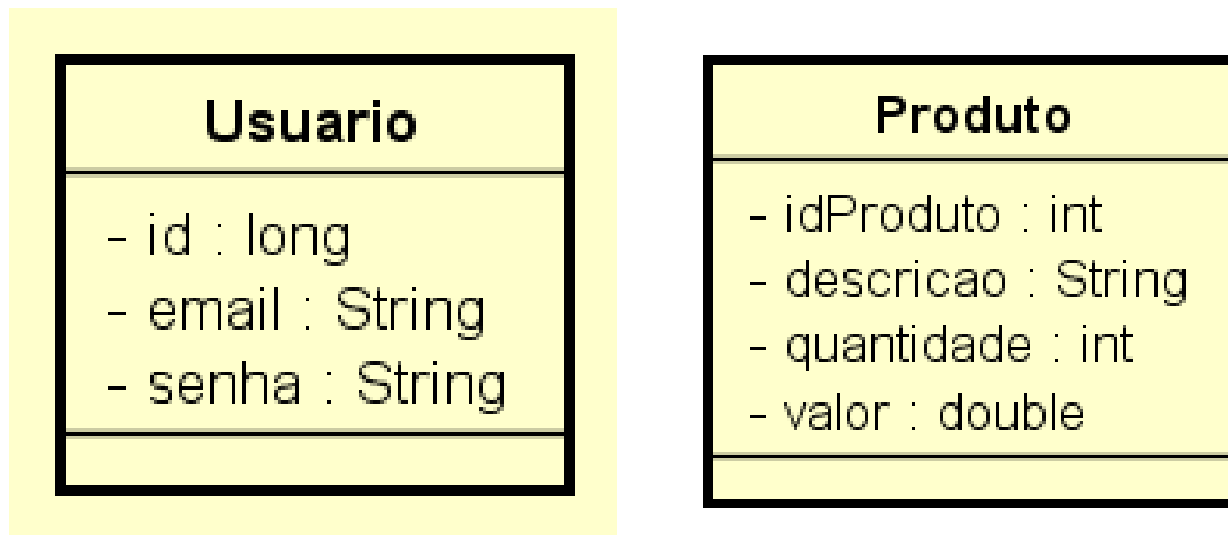
Inicia Transação e Sincroniza com BD

Mapeamento Objeto Relacional

► Pratique!

► Manipulando Entidades

- Considerando as classes abaixo, utilizando o Hibernate:
 - ❑ Faça o mapeamento objeto-relacional;
 - ❑ Implemente a classe de gerenciamento de objetos;
 - ❑ Implemente o método de inserção (*insert*) do objeto.



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

► 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);  
}
```

```
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();  
    }  
}
```

**JPQL - Java Persistence Query
Language**

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[]>**

- Algumas consultas possuem resultados complexos.

```
String query = "SELECT f.nome, f.departamento.nome FROM Funcionario f";
Query query = manager.createQuery(query);
List<Object[]> lista = query.getResultList();

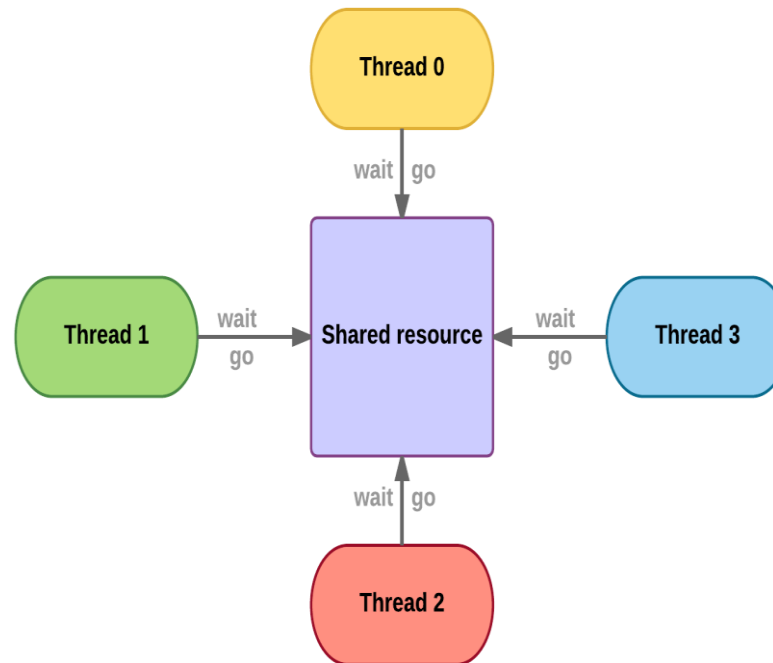
for(Object[] tupla : lista) {
    System.out.println("Funcionário: " + tupla[0]);
    System.out.println("Departamento: " + tupla[1]);
}
```

- Nesse caso, o resultado será uma lista de array de Object. Para manipular essa lista, devemos
- lidar com o posicionamento dos dados nos arrays.

Mapeamento Objeto Relacional

► Controle de Concorrência (Isolamento)

- Evitar que transações paralelas interfiram umas nas outras.
- Quando dois *Entity Managers* manipulam o mesmo objeto, 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); //saldo=1000  
  
x.setSaldo(x.getSaldo() + 500); //1500  
  
Conta y = manager2.find(Conta.class, 1L); //saldo=1000  
  
x.setSaldo(x.getSaldo() - 500); //saldo=500  
  
manager1.getTransaction().commit(); //saldo=1500  
  
manager2.getTransaction().commit(); //saldo=500
```

Mapeamento Objeto Relacional

► Controle de Concorrência

► Solução

► Locking Otimista: **@Version**

- ❑ Acrescenta um atributo para o controle de versão.
- ❑ 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.
- ❑ Caso as versões sejam diferentes uma exceção é lançada.

```
@Entity
public class Conta {

    @Id
    @GeneratedValue
    private Long id;

    private double saldo;

    @Version
    private Long versao;

    // GETTERS AND SETTERS
}
```


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)

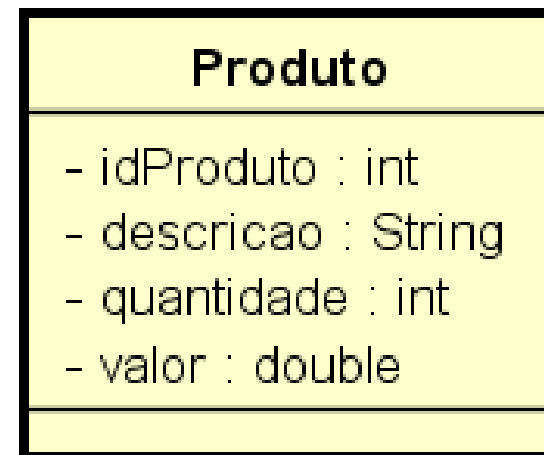
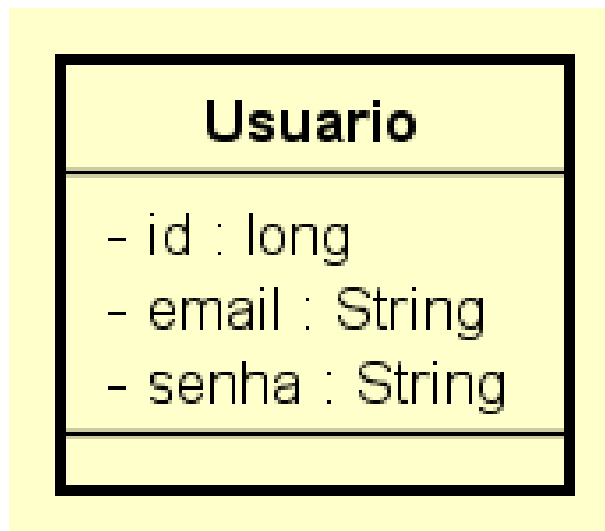
```
public class ExcluindoCurso {  
    public static void main(String args[]){  
        EntityManagerFactory factory =  
            Persistence.createEntityManagerFactory("SysControleAcademicoJPA");  
  
        EntityManager manager = factory.createEntityManager();  
  
        Curso curso = manager.find(Curso.class,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

Mapeamento Objeto Relacional

► Pratique!

► Manipulando Entidades

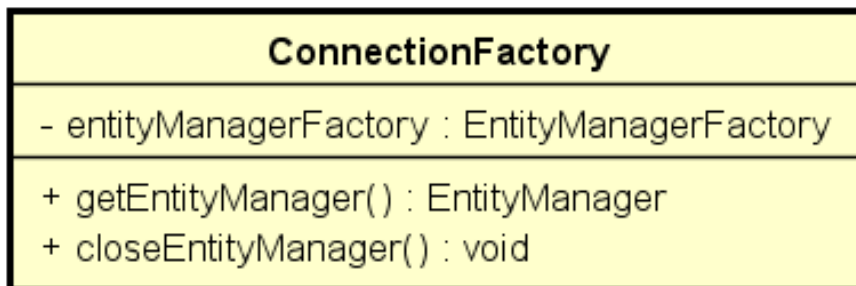
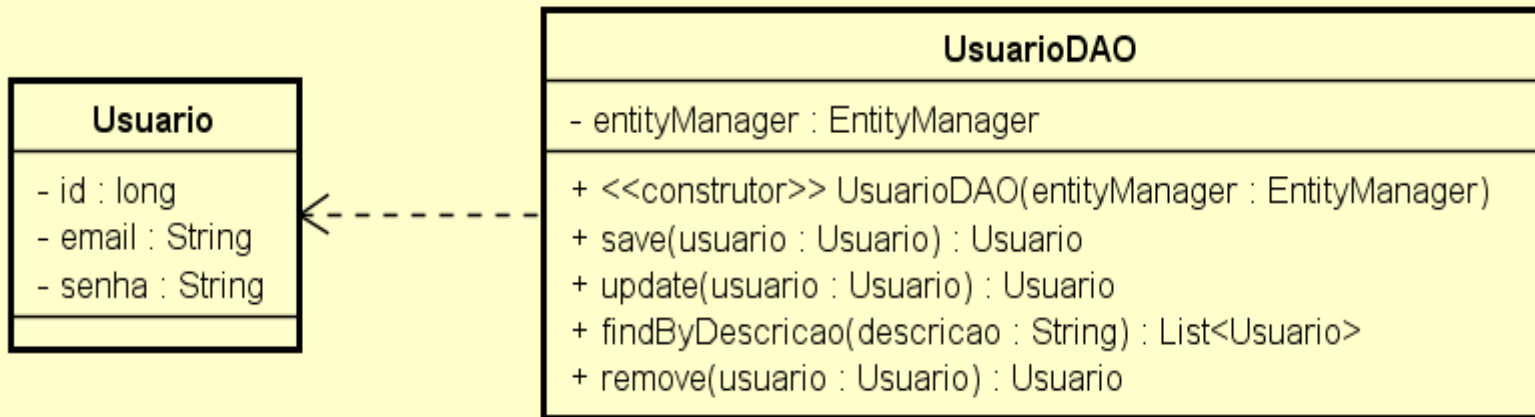
- Considerando as classes abaixo, utilizando o Hibernate:
 - ❑ Faça o mapeamento objeto-relacional;
 - ❑ Implemente a classe de gerenciamento de objetos;
 - ❑ Implemente os métodos de alteração (*merge*) e exclusão (*remove*) do objeto.



Mapeamento Objeto Relacional

► Pratique!

- De acordo com o modelo abaixo, implemente as classes aplicando JPA + *Hibernate*.



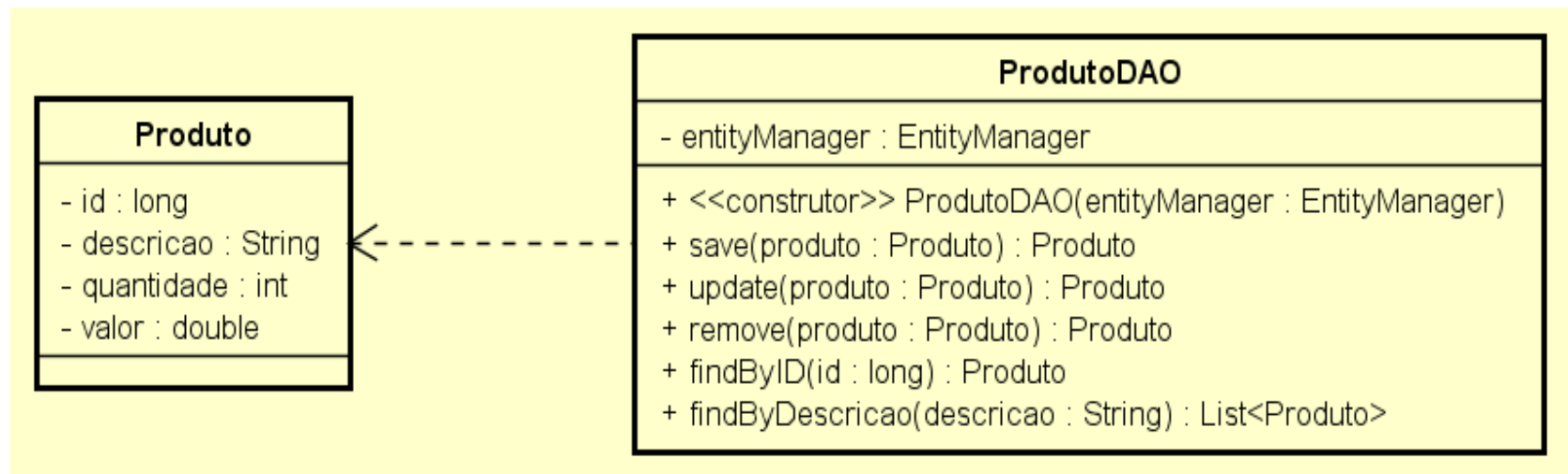
https://github.com/joyceMiranda/codigosDeExemplo/tree/master/DRA_JPA_HIBERNATE/src/jpa/basics/connectionfactory

Mapeamento Objeto Relacional

PROGRAMAÊ!

► Tarefa de Implementação

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



Mapeamento Objeto Relacional

► Manipulando Entidades

► Apresentação do DAO Genérico

► **Problema**

- Levando em consideração que uma classe X pode ter uma representação X_DAO responsável por executar as suas lógicas de persistência. Imagine se houver 10, 100, 200 entidades que necessitam de integração com o banco de dados.
- Seria necessário criar a mesma quantidade de classes DAO para executar as regras de persistência destas entidades.

Mapeamento Objeto Relacional

► Manipulando Entidades

► Apresentação do DAO Genérico

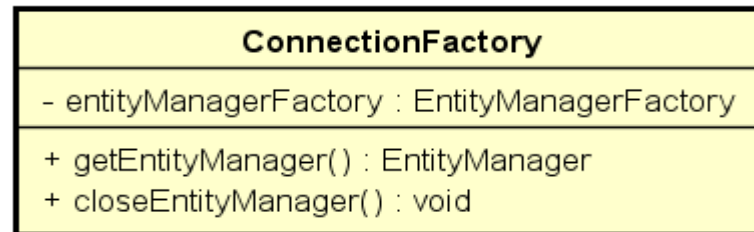
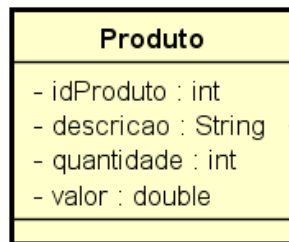
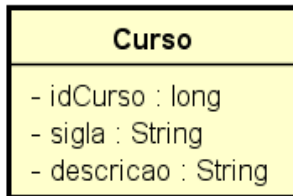
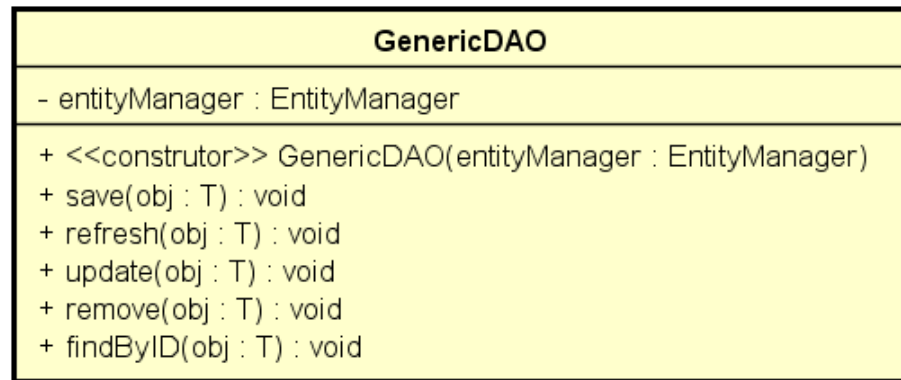
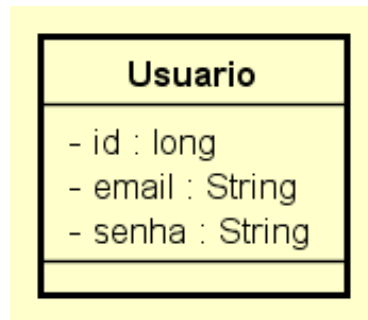
► **A solução**

- O conceito de **Generics** do Java permite criar uma classe DAO capaz de abstrair um tipo qualquer de entidade e executar comandos específicos, eliminando assim os chamados códigos clichês.
- Desta forma, se um número N de entidades do sistema tem características semelhantes, ao invés de se criar N classes DAO, a aplicação terá apenas **uma classe genérica** abstraindo as funcionalidades em comum entre um grupo específico de objetos que necessitam de comunicação com o banco de dados.

Mapeamento Objeto Relacional

► Manipulando Entidades

► Apresentação do DAO Genérico



Conteúdo disponível em:

https://github.com/joyceMiranda/codigosDeExemplo/tree/master/DRA_JPA_HIBERNATE/src/jpa/basics/daogenerico

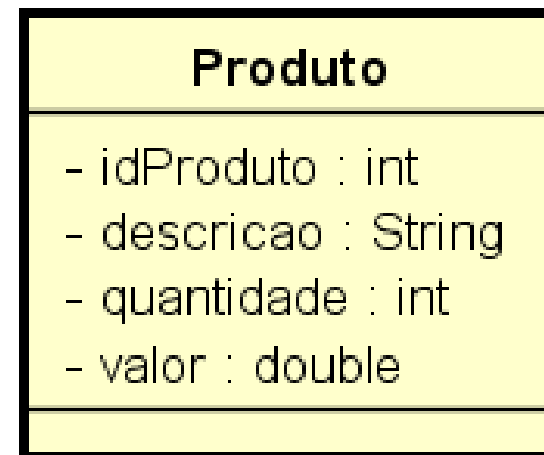
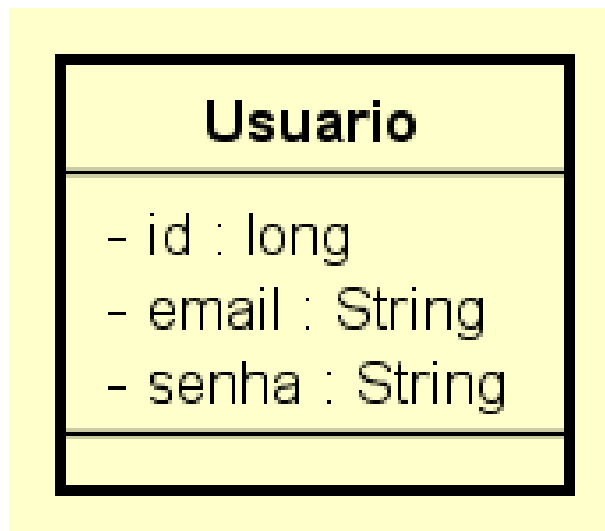
Mapeamento Objeto Relacional

► Pratique!

► Manipulando Entidades com GenericDAO

► Considerando as classes abaixo, utilizando o Hibernate:

- ☐ Faça o mapeamento objeto-relacional;
- ☐ Implemente a classe de gerenciamento de objetos;
- ☐ Implemente o CRUD a partir da classe GenericDAO.

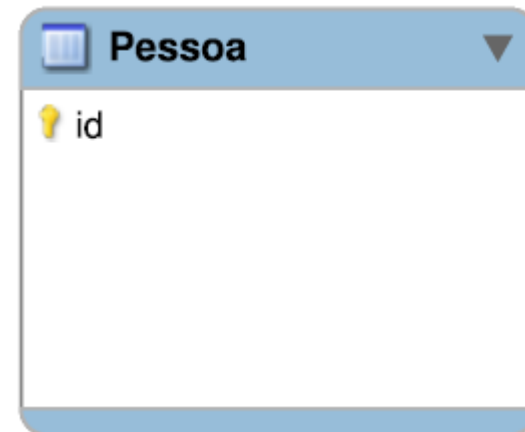


Mapeamento Objeto Relacional

► Mapeamento

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

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

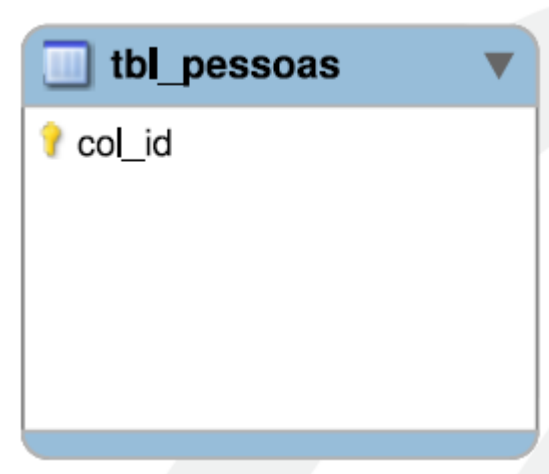


Mapeamento Objeto Relacional

► 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

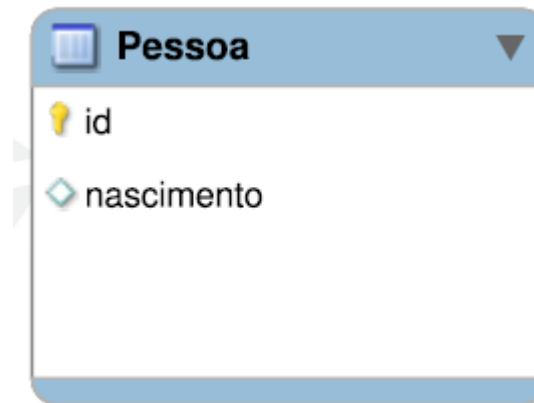
► @Transient

- 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;
}
```

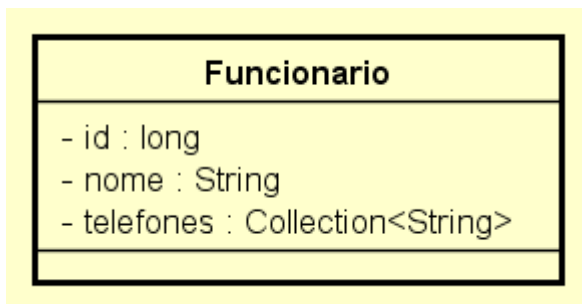
```
public enum Periodo {
    MATUTINO,
    VESPERTINO,
    NOTURNO
}
```

Mapeamento Objeto Relacional

► Mapeamento

► Coleções

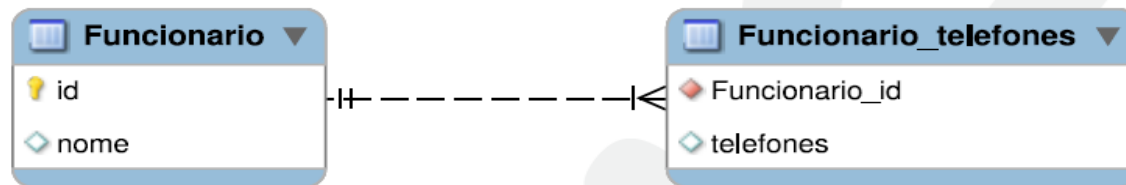
► @ElementCollection



```
@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 {

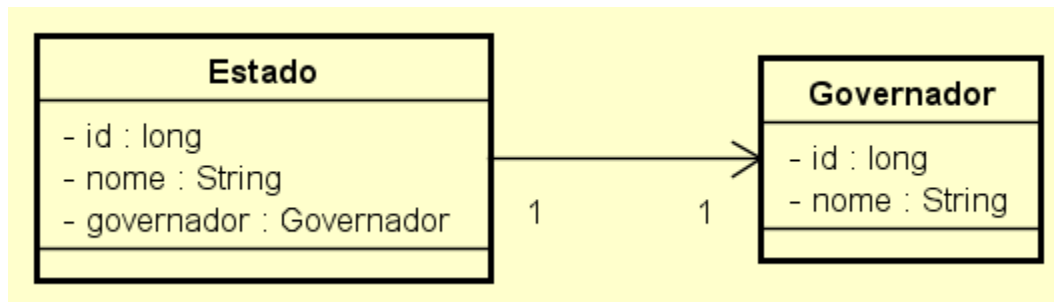
    @Id @GeneratedValue
    private Long id;

    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 – Relacionamento – **OneToOne**



```
@Entity
public class Estado {

    @Id @GeneratedValue
    private long id;
    private String nome;

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

}
```

```
@Entity
public class Governador {

    @Id @GeneratedValue
    private long id;
    private String nome;

}
```

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **OneToOne**

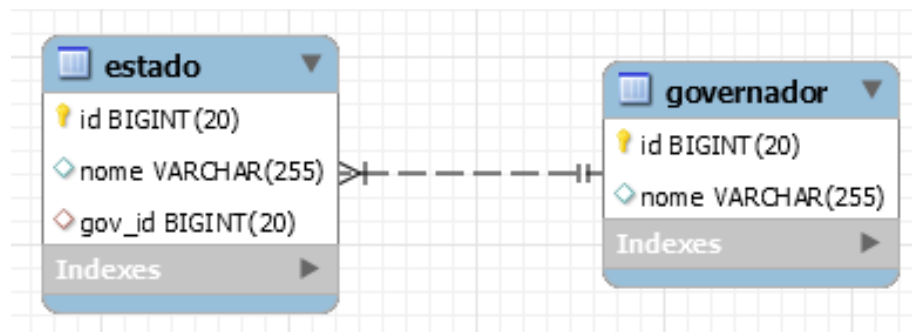
```
@Entity
public class Estado {

    @Id @GeneratedValue
    private long id;
    private String nome;

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

```
@Entity
public class Governador {

    @Id @GeneratedValue
    private long id;
    private String nome;
}
```



Mapeamento Objeto Relacional

► Mapeamento - Relacionamentos

- Operações do *EntityManager* são aplicadas somente ao objeto passado como parâmetro, não sendo aplicadas a objetos relacionados.



```
Governador governador = new Governador("Fulaninho");  
Estado estado = new Estado("Amazonas", governador);  
manager.persist(estado);
```

Os dois objetos precisam ser persistidos



```
Governador governador = new Governador("Fulaninho");  
manager.persist(governador);  
  
Estado estado = new Estado("Amazonas", governador);  
manager.persist(estado);
```

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.

- CascadeType.PERSIST
- CascadeType.MERGE
- CascadeType.REMOVE
- CascadeType.ALL

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **OneToOne**

```
@Entity
public class Estado_Cascade {

    @Id @GeneratedValue
    private long id;
    private String nome;

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

}
```

```
@Entity
public class Governador_Cascade {


    @Id @GeneratedValue
    private long id;
    private String nome;

}
```

```
Governador_Cascade governador = new Governador_Cascade("Fulaninho");

Estado_Cascade estado = new Estado_Cascade("Amazonas", governador);

manager.persist(estado);
```

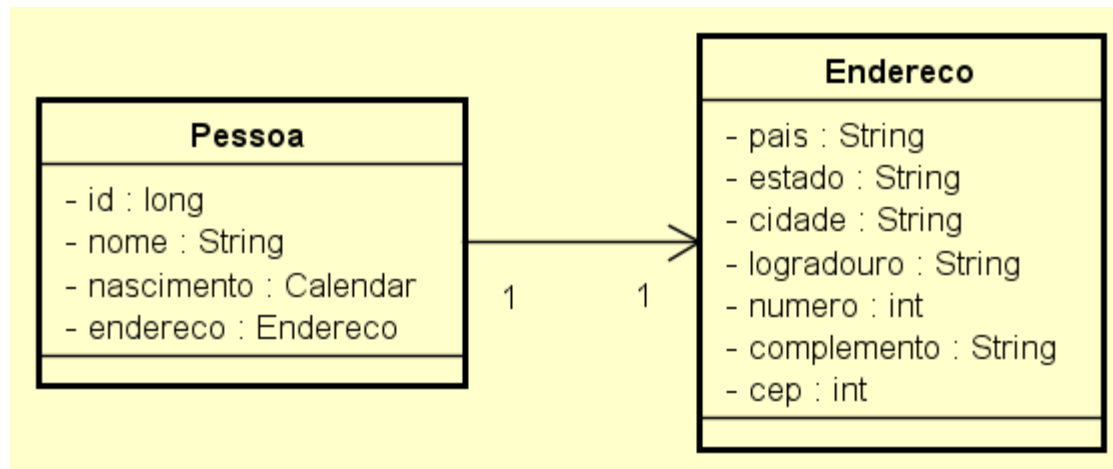


Mapeamento Objeto Relacional

► 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.



pessoa	
col_id	BIGINT(20)
cidade	VARCHAR(255)
estado	VARCHAR(255)
logradouro	VARCHAR(255)
numero	VARCHAR(255)
pais	VARCHAR(255)
foto	LONGBLOB
nascimento	DATE
nome	VARCHAR(30)
Indexes	

Mapeamento Objeto Relacional

► Mapeamento

► Objetos Embutidos

```
@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`

```
@Embeddable
class Endereco {

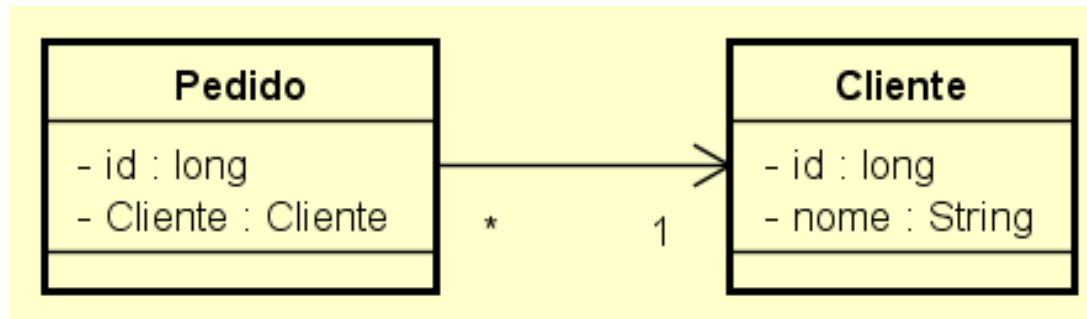
    private String pais ;
    private String estado ;
    private String cidade ;
    private String logradouro ;
    private int numero ;
    private String complemento ;
    private int cep ;
}
```

Substitui `@Entity` por
`@Embeddable`, que indica que é
uma classe embutida.

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

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **ManyToOne**



```
@Entity
public class Pedido {

    @Id @GeneratedValue
    private long id;

    @ManyToOne(cascade = CascadeType.PERSIST)
    private Cliente cliente;

}
```

```
@Entity
public class Cliente {

    @Id @GeneratedValue
    private long id;
    private String nome;

}
```

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **ManyToOne**

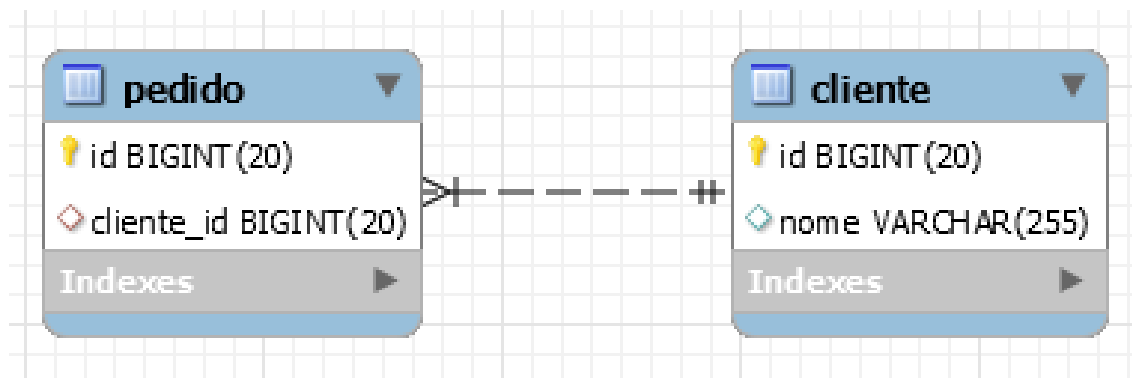
```
@Entity
public class Pedido {

    @Id @GeneratedValue
    private long id;

    @ManyToOne(cascade = CascadeType.PERSIST)
    private Cliente cliente;
}
```

```
@Entity
public class Cliente {

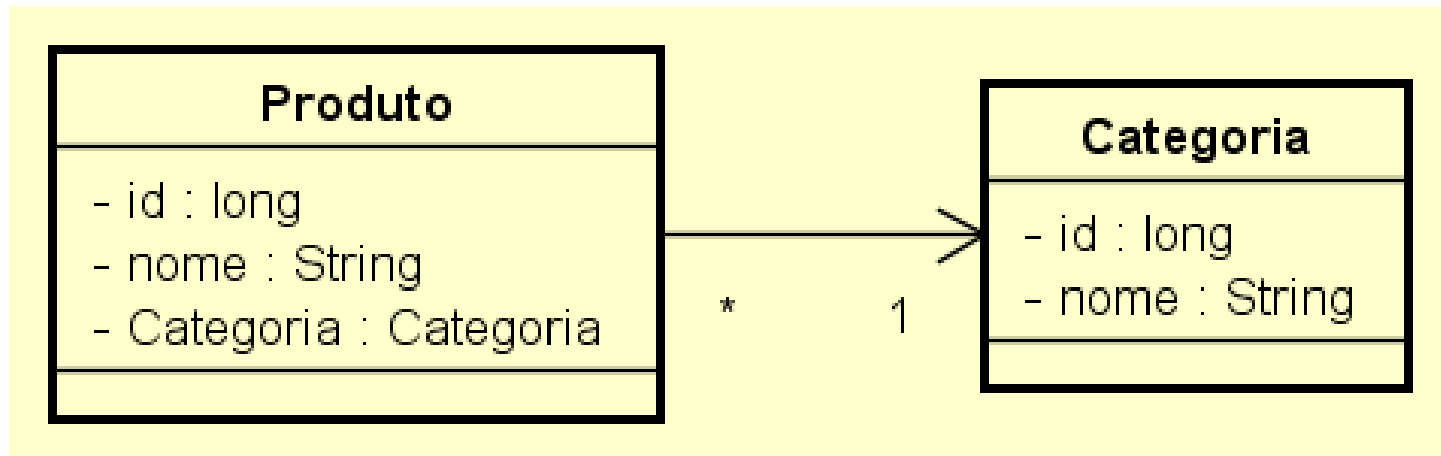
    @Id @GeneratedValue
    private long id;
    private String nome;
}
```



Mapeamento Objeto Relacional

► Pratique!

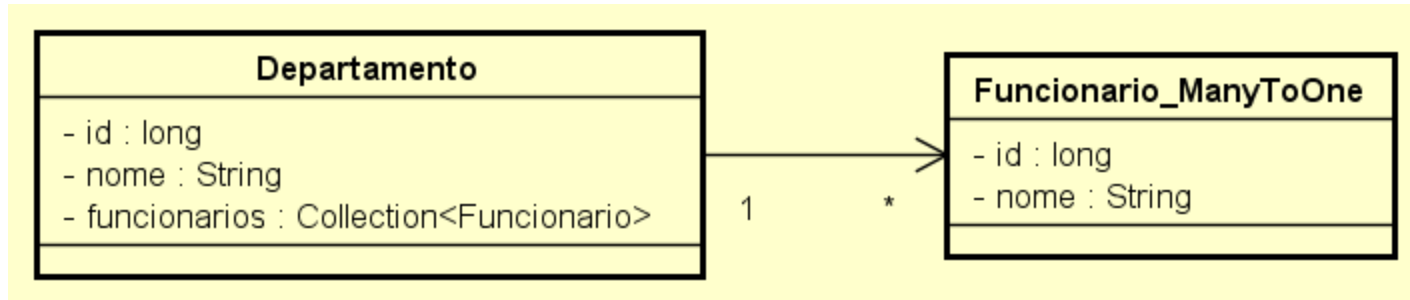
► Mapeamento – Relacionamentos – *Many To One*



**Crie uma classe de persistência para inserir no BD
objetos de cada Classe do Modelo acima**

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **OneToMany**



```
@Entity
public class Departamento {

    @Id @GeneratedValue
    private long id;
    private String nome;

    @OneToMany(cascade = CascadeType.PERSIST)
    private Collection<Funcionario_ManyToOne> funcionarios;
}
```

```
@Entity
public class Funcionario_ManyToOne {

    @Id @GeneratedValue
    private long id;
    private String nome;
}
```

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **OneToMany**

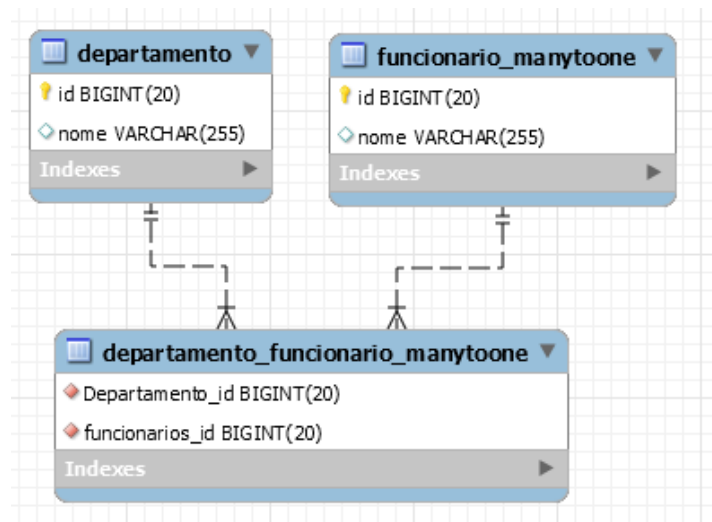
```
@Entity
public class Departamento {

    @Id @GeneratedValue
    private long id;
    private String nome;

    @OneToMany(cascade = CascadeType.PERSIST)
    private Collection<Funcionario_ManyToOne> funcionarios;
}
```

```
@Entity
public class Funcionario_ManyToOne {

    @Id @GeneratedValue
    private long id;
    private String nome;
}
```



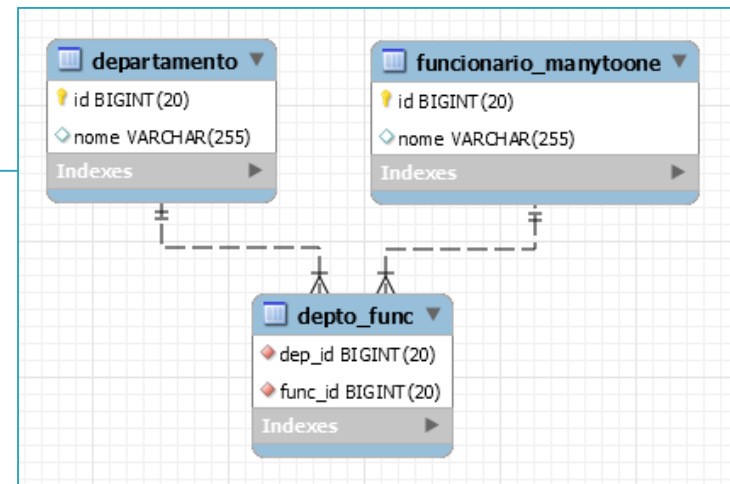
Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **OneToMany**

```
@Entity
public class Departamento {

    @Id @GeneratedValue
    private long id;
    private String nome;

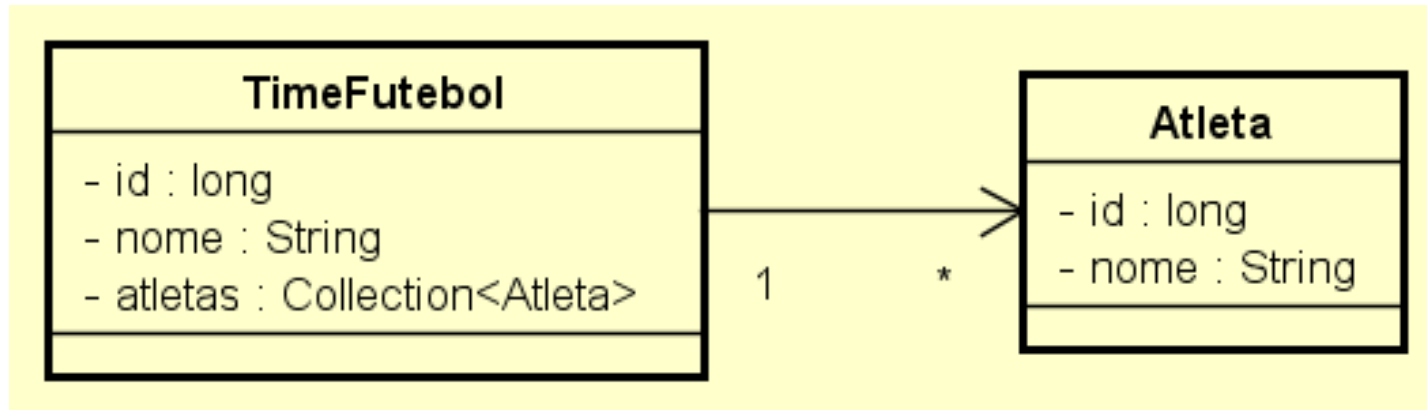
    @OneToMany(cascade = CascadeType.PERSIST)
    @JoinTable(name = "depto_func",
               joinColumns = @JoinColumn(name="dep_id"),
               inverseJoinColumns = @JoinColumn(name="func_id"))
    private Collection<Funcionario_ManyToOne> funcionarios;
}
```



Mapeamento Objeto Relacional

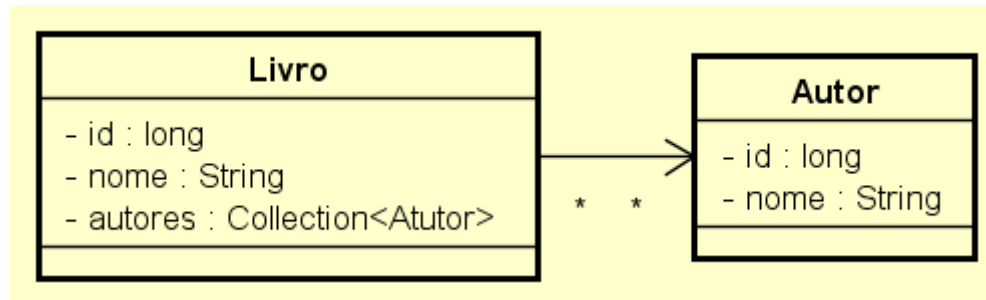
► Pratique!

► Mapeamento – Relacionamentos – *One To Many*



Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **ManyToMany**



```
@Entity
public class Livro {

    @Id @GeneratedValue
    private long id;
    private String nome;

    @ManyToMany(cascade = CascadeType.PERSIST)
    private Collection<Autor> autores;
}
```

```
@Entity
public class Autor {

    @Id @GeneratedValue
    private long id;

    private String nome;
}
```

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – **ManyToMany**

```
@Entity
public class Livro {

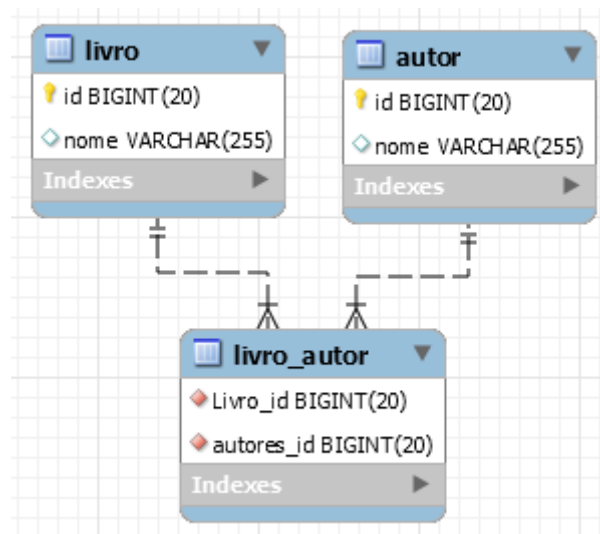
    @Id @GeneratedValue
    private long id;
    private String nome;

    @ManyToMany(cascade = CascadeType.PERSIST)
    private Collection<Autor> autores;
}
```

```
@Entity
public class Autor {

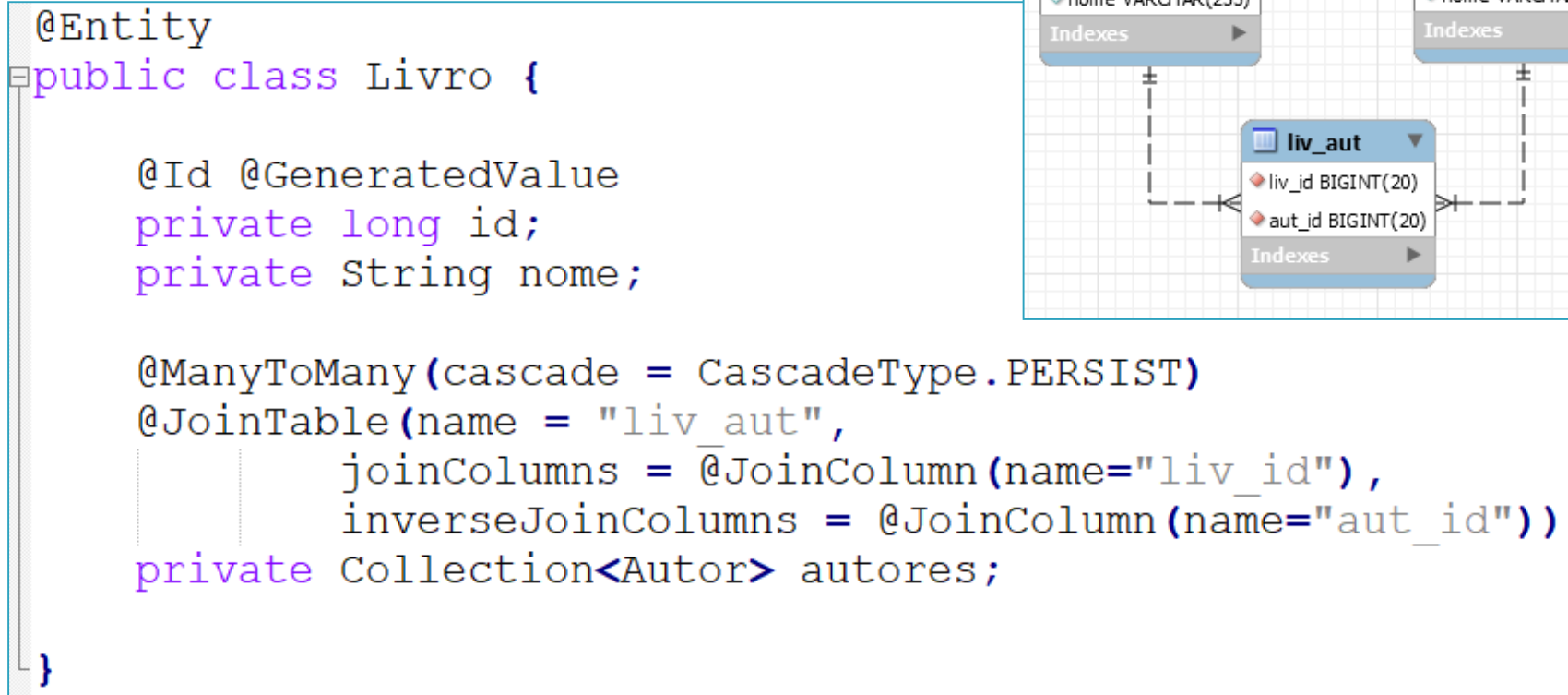
    @Id @GeneratedValue
    private long id;

    private String nome;
}
```



Mapeamento Objeto Relacional

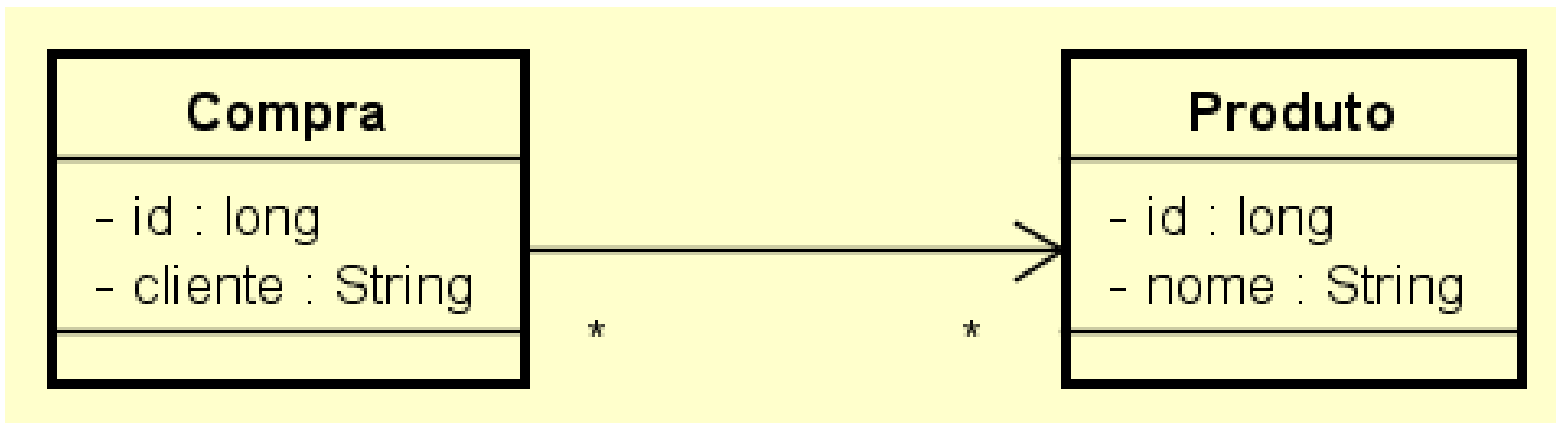
► Mapeamento – Relacionamento – *ManyToMany*



Mapeamento Objeto Relacional

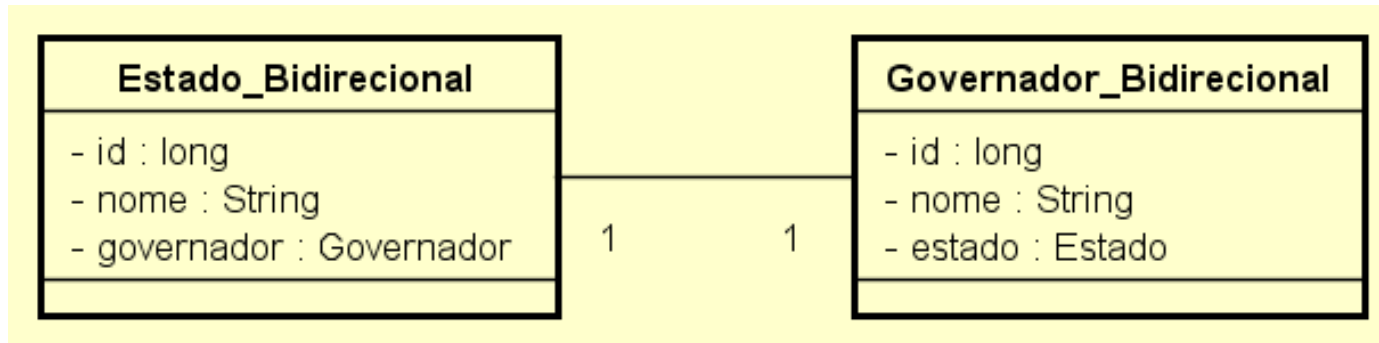
► Pratique!

► Mapeamento – Relacionamento – *ManyToMany*



Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – Bidirecional

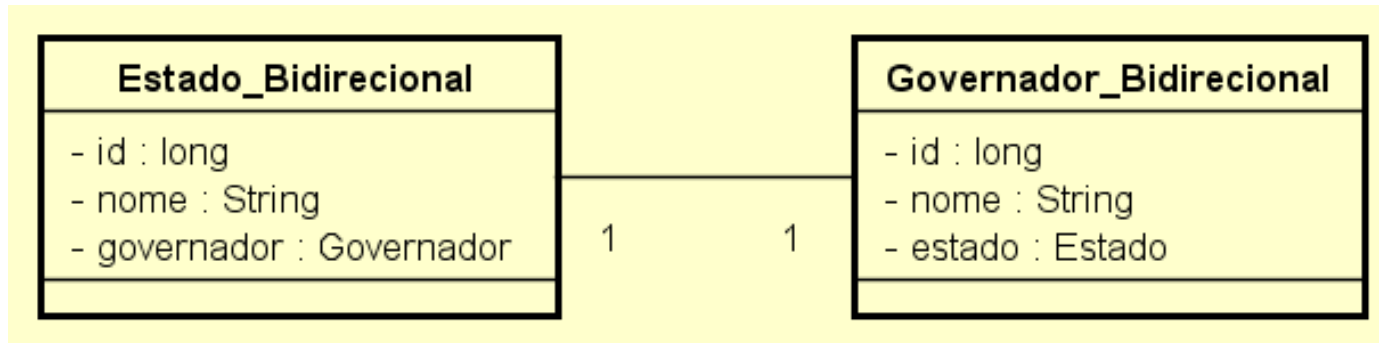


► Sentido 1

```
Estado_Bidirecional estado =
    manager.find(Estado_Bidirecional.class, 1L);
Governador_Bidirecional governador =
    estado.getGovernador();
```

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – Bidirecional

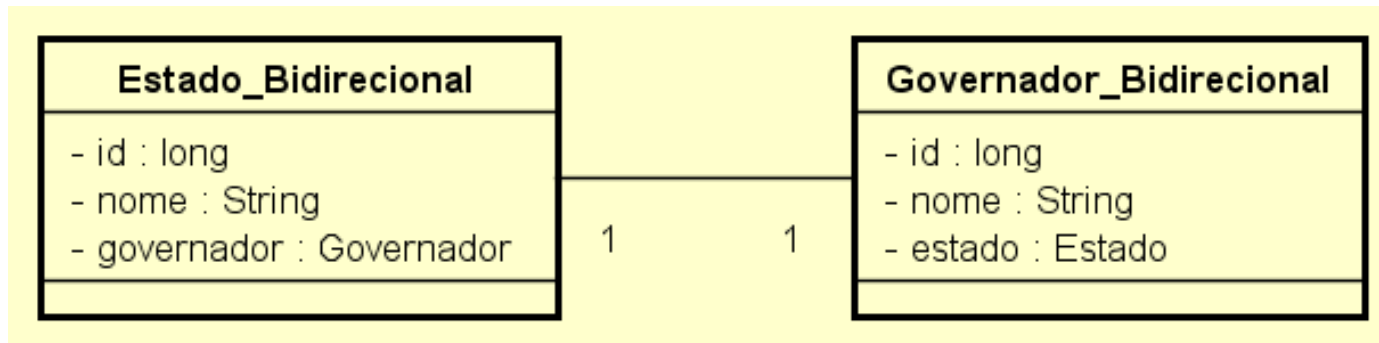


► Sentido 2

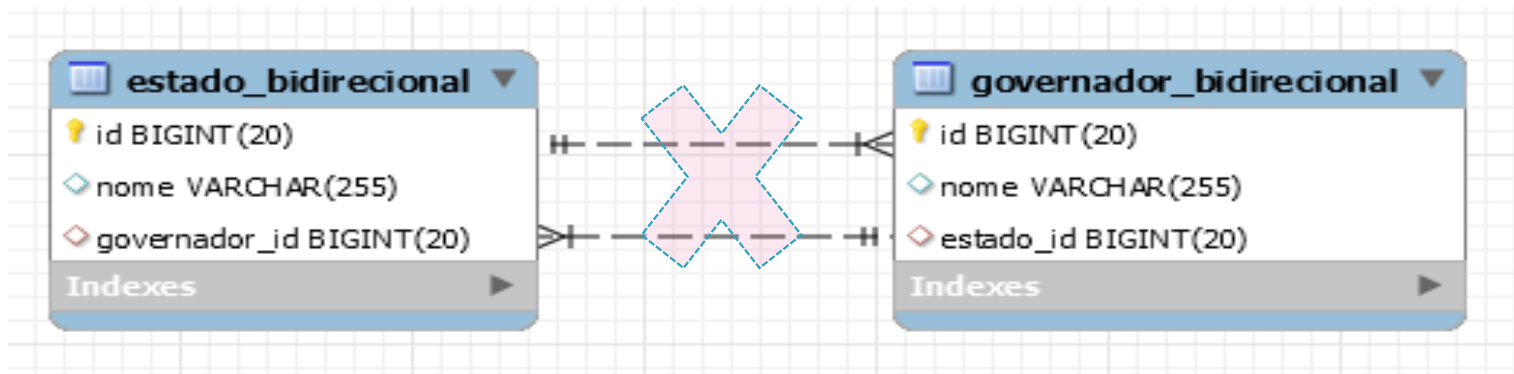
```
Governador_Bidirecional governador =  
    manager.find(Governador_Bidirecional.class, 1L);  
Estado_Bidirecional estado =  
    governador.getEstado();
```

Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – Bidirecional



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



Mapeamento Objeto Relacional

► Mapeamento – Relacionamento – Bidirecional

► Solução

- **mappedBy**: Indica que o relacionamento já foi mapeado em outra classe.
- *O valor do mappedBy deve ser o nome do atributo que expressa o mesmo relacionamento na outra entidade*

```
@Entity
public class Estado_Bidirecional {

    @Id @GeneratedValue
    private long id;
    private String nome;

    @OneToOne(cascade = CascadeType.PERSIST)
    private Governador_Bidirecional governador;
}
```

```
@Entity
public class Governador_Bidirecional {

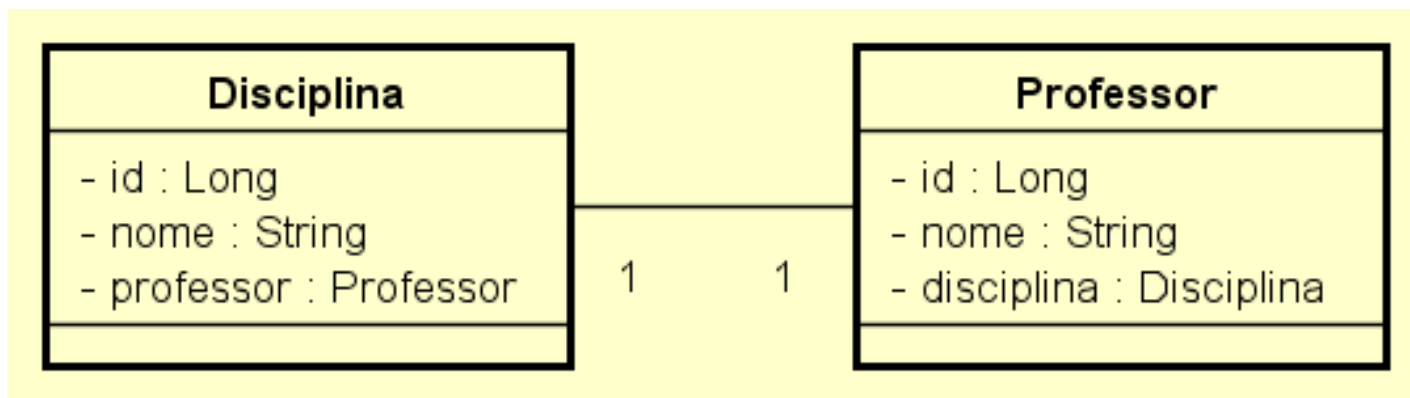
    @Id @GeneratedValue
    private long id;
    private String nome;

    @OneToOne(mappedBy = "governador")
    private Estado_Bidirecional estado;
}
```

Mapeamento Objeto Relacional

► Pratique!

- Mapeamento – Relacionamentos – ***Bidirecional***
 - * ***Classe dominante = Disciplina***

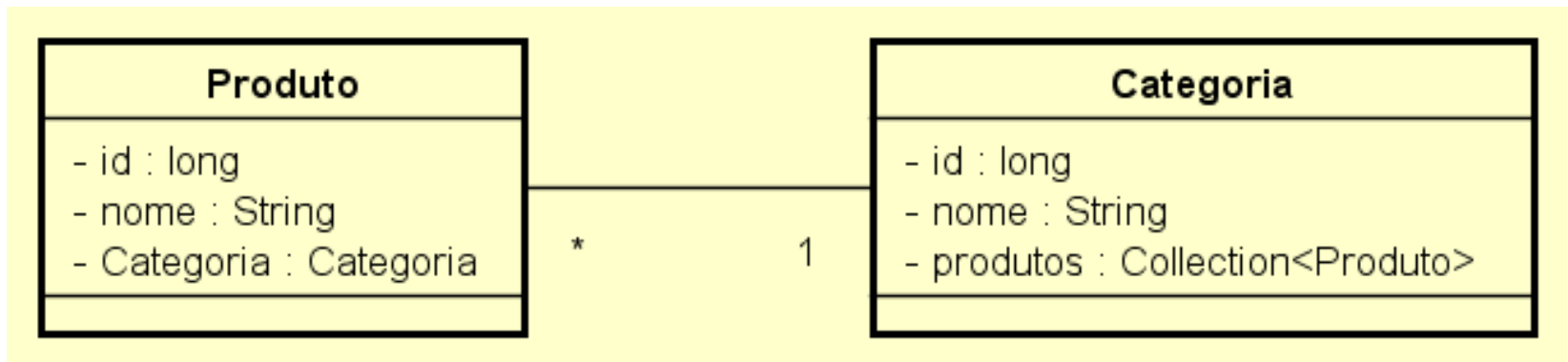


Crie uma classe de persistência para inserir e recuperar objetos do BD para cada classe do modelo

Mapeamento Objeto Relacional

► Pratique!

- Mapeamento – Relacionamento – ***Bidirecional***
 - * ***Classe dominante = Produto***

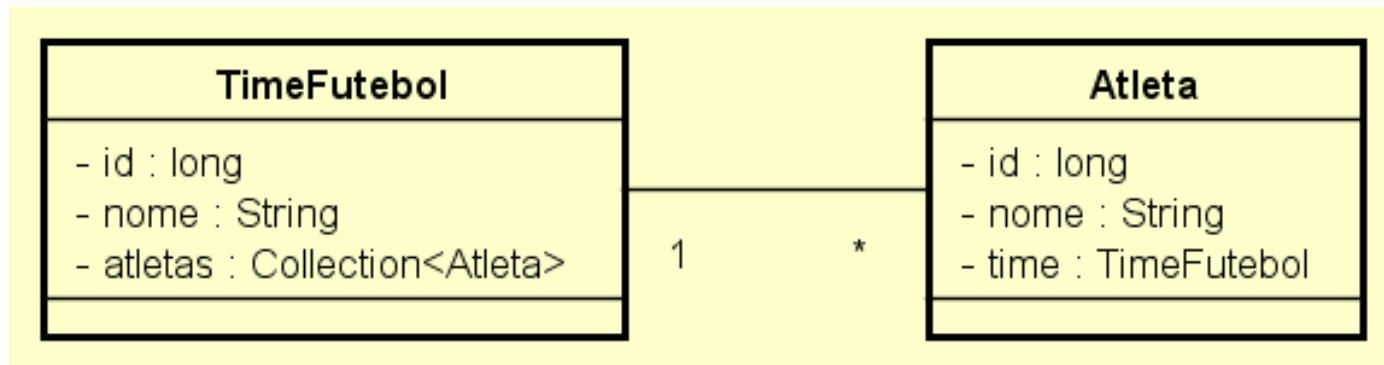


Crie uma classe de persistência para inserir e recuperar objetos do BD para cada classe do modelo

Mapeamento Objeto Relacional

► Pratique!

- Mapeamento – Relacionamento – ***Bidirecional***
 - * ***Classe dominante = TimeFutebol***

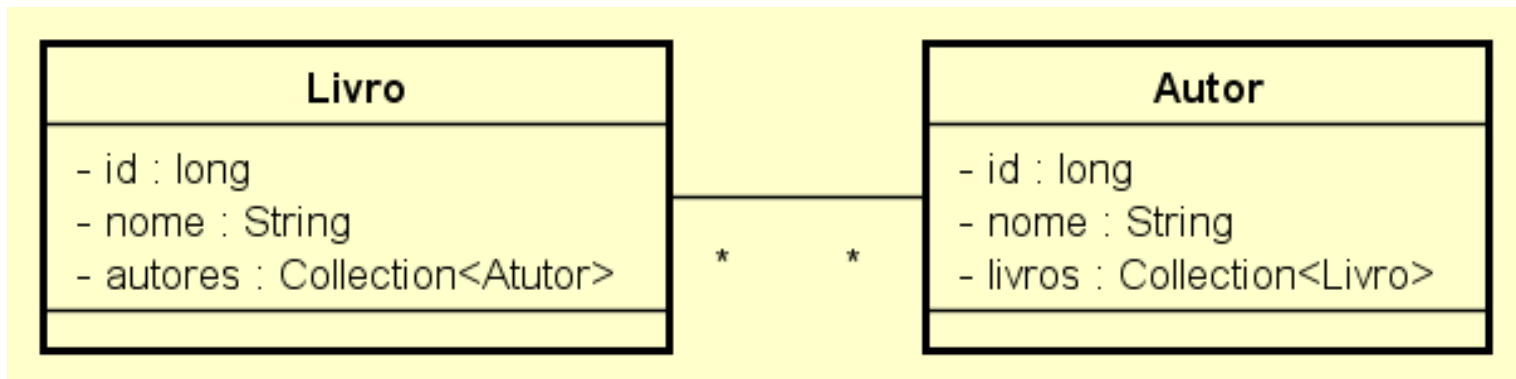


Crie uma classe de persistência para inserir e recuperar objetos do BD para cada classe do modelo

Mapeamento Objeto Relacional

► Pratique!

- Mapeamento – Relacionamento – ***Bidirecional***
 - * ***Classe dominante = Livro***



Crie uma classe de persistência para inserir e recuperar objetos do BD para cada classe do modelo

Mapeamento Objeto Relacional

► Mapeamento

► Herança

- JPA define três estratégias para o mapeamento de herança.

Single Table

Joined

Table per Class

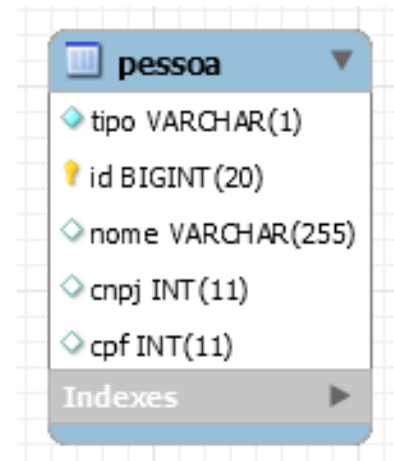
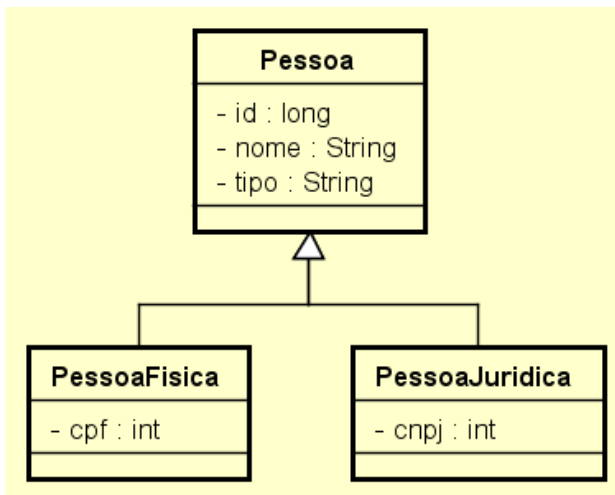
Mapeamento Objeto Relacional

► Mapeamento

► Herança

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

Single Table



Single Table

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="tipo", length=1, discriminatorType=DiscriminatorType.STRING)
public class Pessoa {

    @Id
    @GeneratedValue
    private Long id;
    private String nome;
    @Column(insertable=false, updatable=false)
    private String tipo;
}
```

```
@Entity
@DiscriminatorValue(value = "F")
public class PessoaFisica extends Pessoa {

    private int cpf;
}
```

```
@Entity
@DiscriminatorValue(value = "J")
public class PessoaJuridica extends Pessoa {

    private int cnpj;
}
```

Mapeamento Objeto Relacional

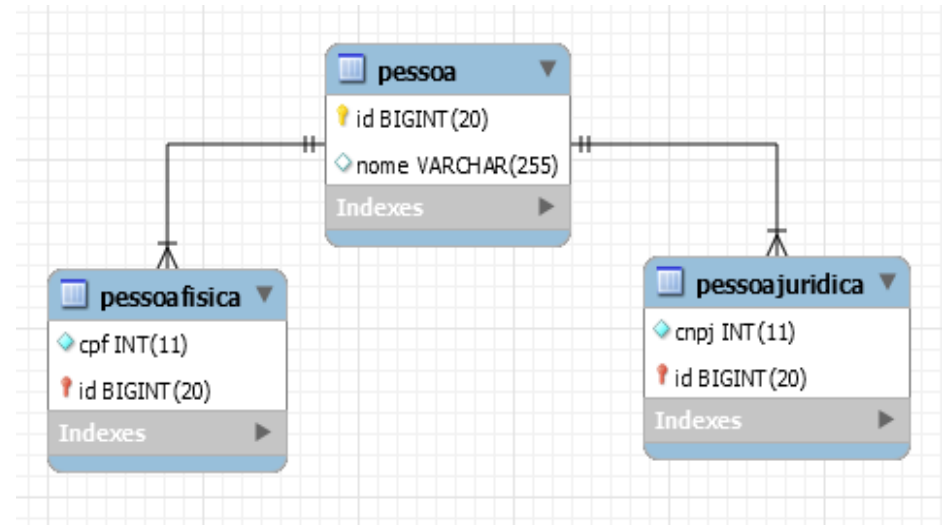
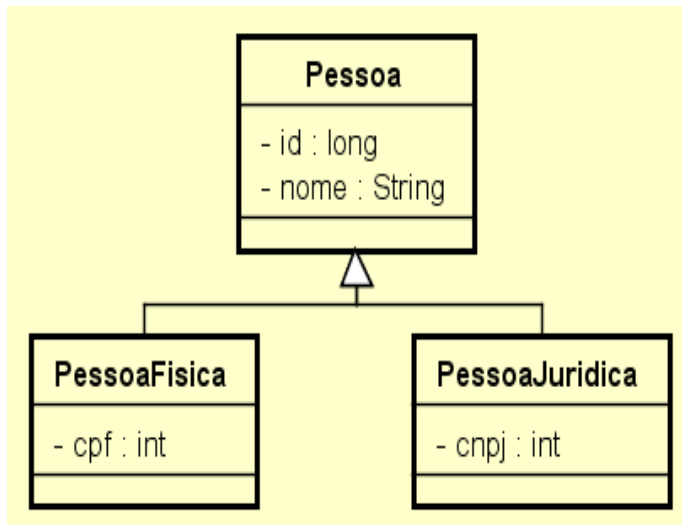
► Mapeamento

► Herança

► Estratégia *Joined*:

- ❑ Classe Mãe e Classes Filhas são geradas no BD, sendo que em todas as classes filhas haverá uma chave estrangeira que apontará para a classe mãe.

Joined



Joined

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Pessoa {

    @Id
    @GeneratedValue
    private Long id;
    private String nome;
}
```

```
@Entity
public class PessoaFisica extends Pessoa {

    private int cpf;
}
```

```
@Entity
public class PessoaJuridica extends Pessoa {

    private int cnpj;
}
```

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.

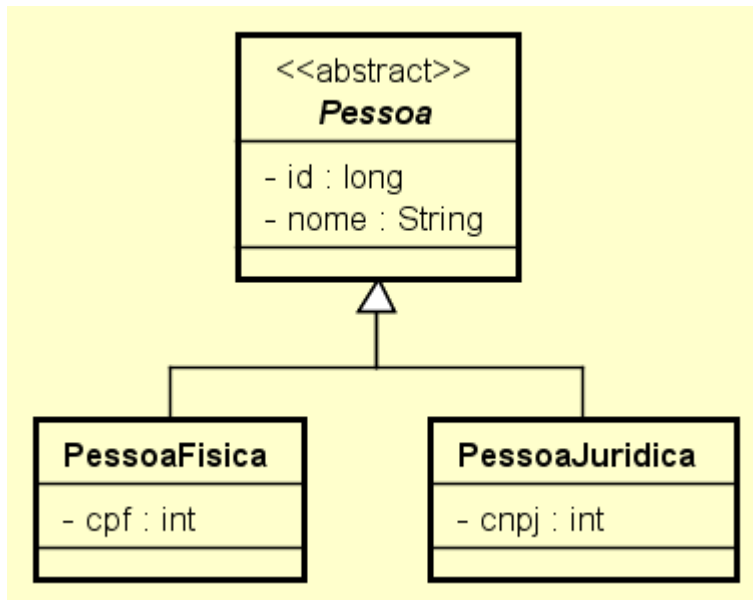


Table per Class

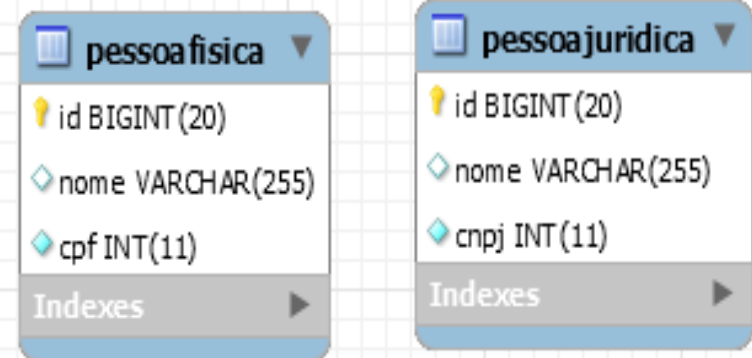


Table per Class

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Pessoa_Heranca {

    @Id @GeneratedValue(strategy = GenerationType.TABLE)
    private long id;
    private String nome;

}
```

<pre>@Entity public class PessoaFisica extends Pessoa { private int cpf; }</pre>	<pre>@Entity public class PessoaJuridica extends Pessoa { private int cnpj; }</pre>
--	---

Mapeamento Objeto Relacional



► Tarefa de Implementação

