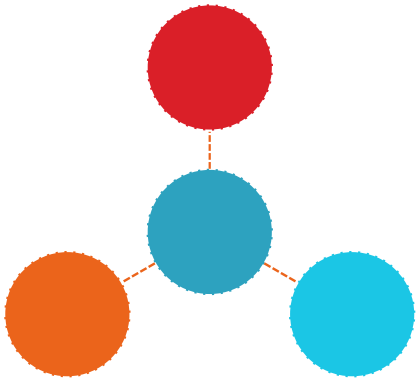
Desenvolvimento Rápido de Aplicações

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.

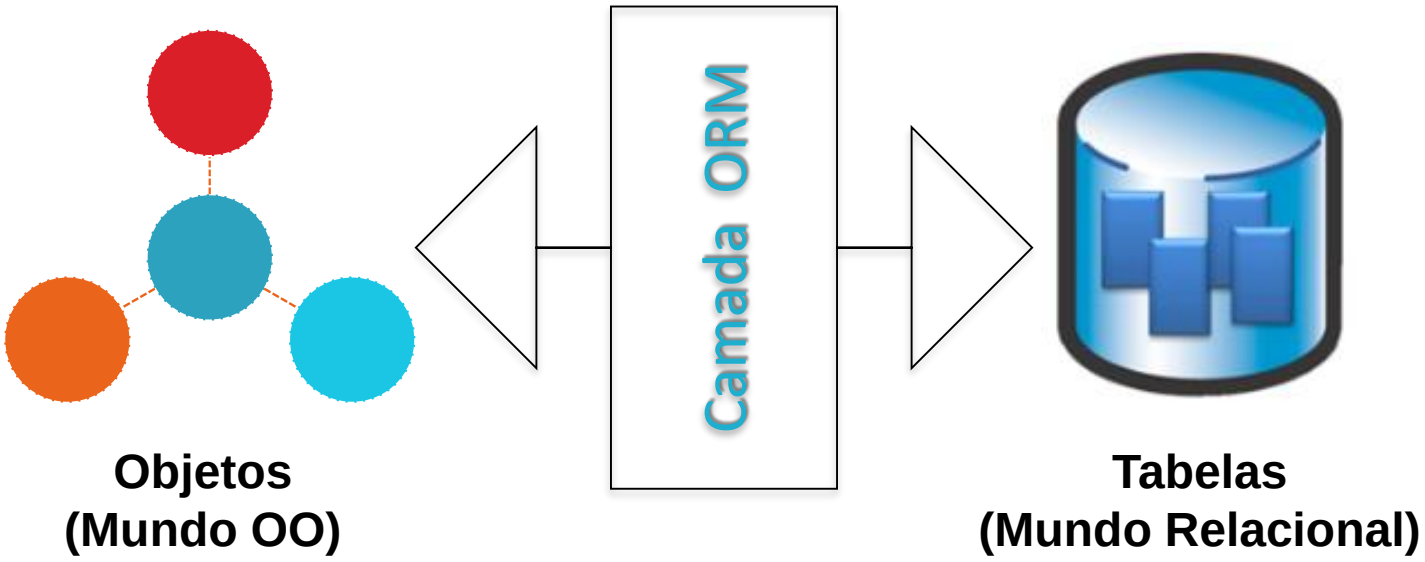


Objetos
(Mundo OO)

Tabelas
(Mundo Relacional)

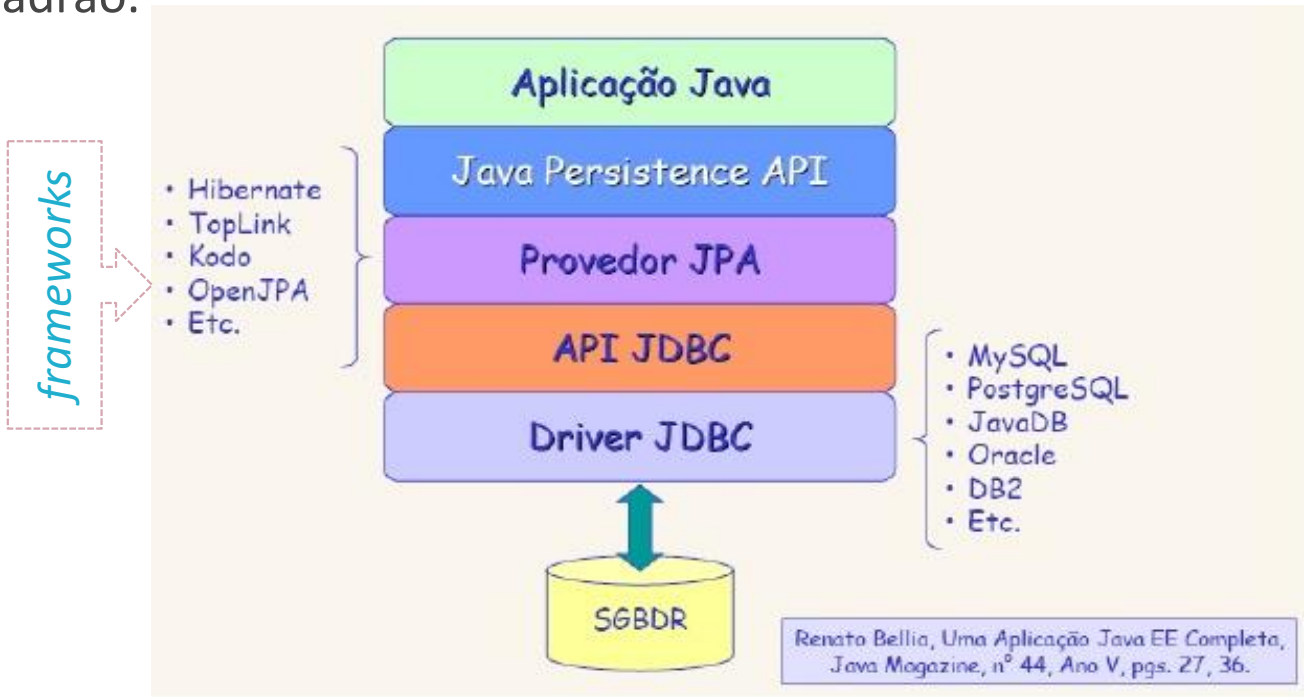
Mapeamento Objeto Relacional

- 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.

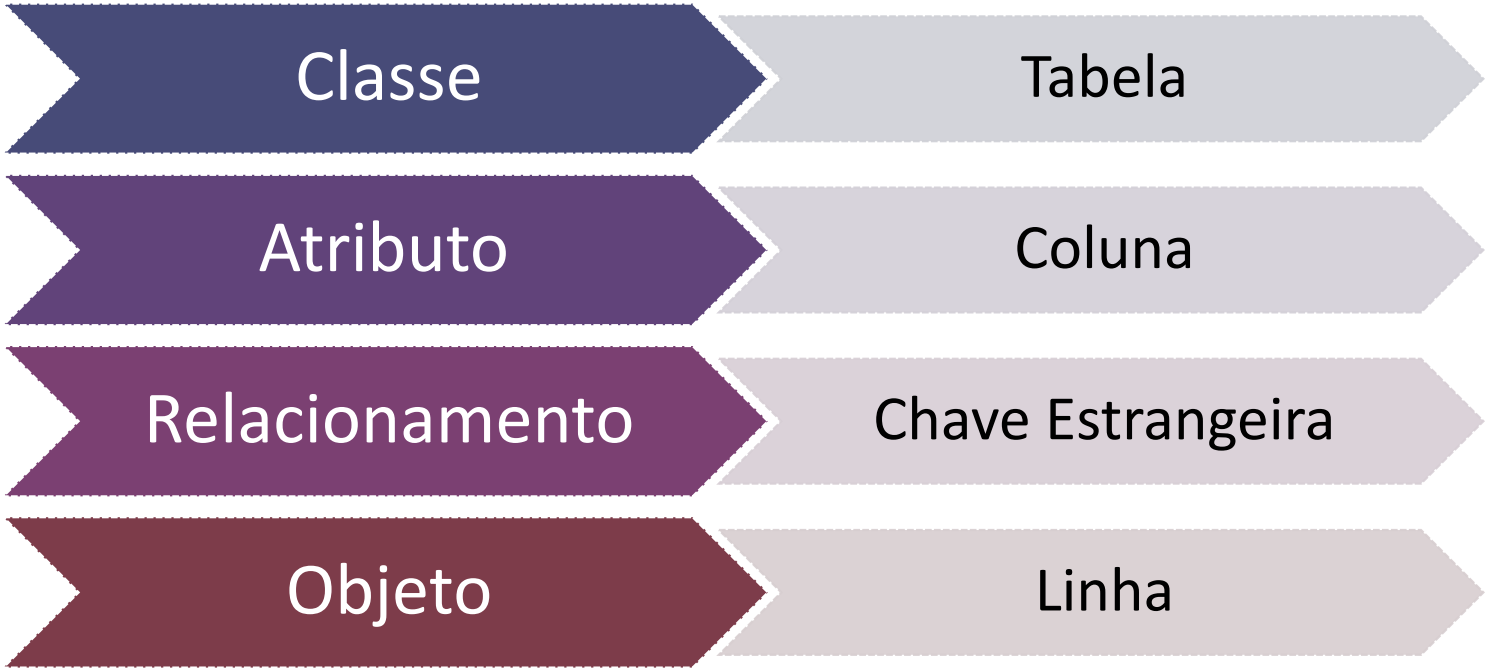


Mapeamento Objeto Relacional

- 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

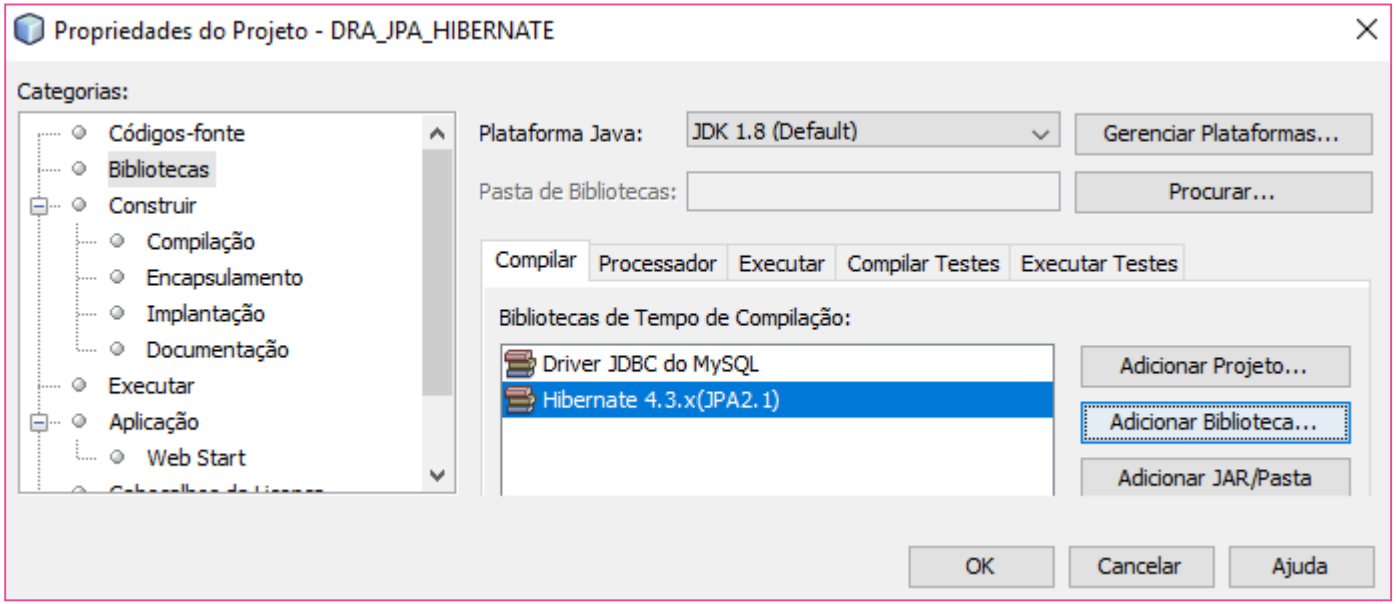
  - Provedores JPA
    - Hibernate
      - http://jpa.hibernate.org
    - Driver JDBC MySQL
      - https://dev.mysql.com/downloads/connector/j/

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

Configurar unidade de persistência

Define Unidade de Persistência

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

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

  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>
    <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.hbm2ddl.auto" value="create-drop"/>
    <property name="hibernate.show_sql" value="true"/>
  </properties>

</persistence-unit>
```


persistence.xml

Configurar unidade de persistência

Define o provedor JPA

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

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

  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>
    <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.hbm2ddl.auto" value="create-drop"/>
    <property name="hibernate.show_sql" value="true"/>
  </properties>

</persistence-unit>
```


persistence.xml

Configurar unidade de persistência

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

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

  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>
    <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.hbm2ddl.auto" value="create-drop"/>
    <property name="hibernate.show_sql" value="true"/>
  </properties>

</persistence-unit>
```

Propriedades do BD

persistence.xml

Configurar unidade de persistência

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

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

  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>
    <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.hbm2ddl.auto" value="create-drop"/>
    <property name="hibernate.show_sql" value="true"/>
  </properties>

</persistence-unit>
```

Update
Create-Drop
Validate

persistence.xml

Configurar unidade de persistência

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

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

  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/syscontroleacademico"/>
    <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.hbm2ddl.auto" value="create-drop"/>
    <property name="hibernate.show_sql" value="true"/>
  </properties>

</persistence-unit>
```

Mostra SQL gerado

persistence.xml

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

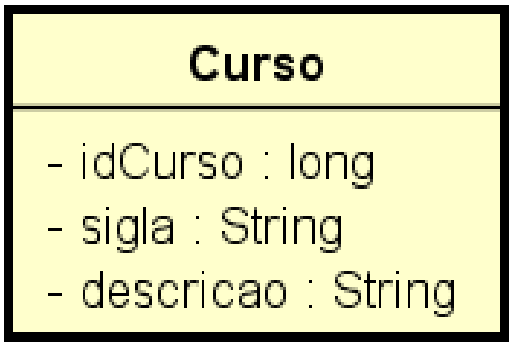
  - Indica que o valor do atributo que compõe a chave primária deve ser gerado automaticamente pelo banco de dados.
  - Geralmente vem acompanhado pela anotação @Id

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;

}
```



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(
                "SysControleAcademicoEstrutural");
factory.close();
```

Mapeamento Objeto Relacional

- Como usar JPA - Passo a Passo

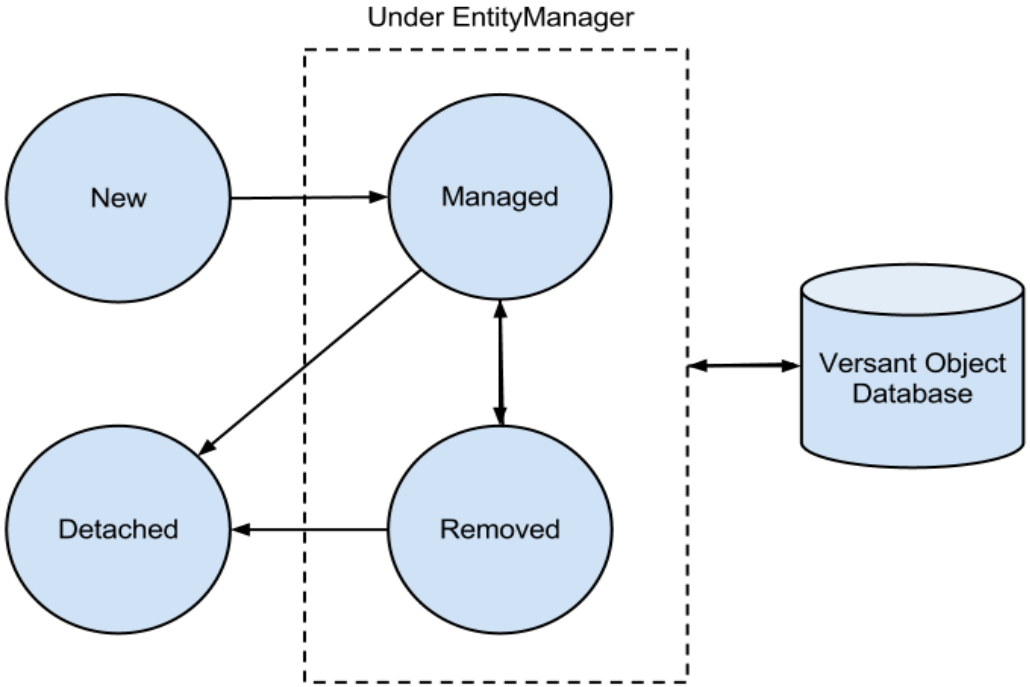
Criar classes de gerenciamento de objetos

- Geração automática de tabelas no banco de dados.

```
public class GeraTabelas {

    public static void main(String[] args) {

        EntityManagerFactory factory =
                Persistence.createEntityManagerFactory(
                        "SysControleAcademicoEstrutural");
        factory.close();

    }

}
```

Mapeamento Objeto Relacional

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

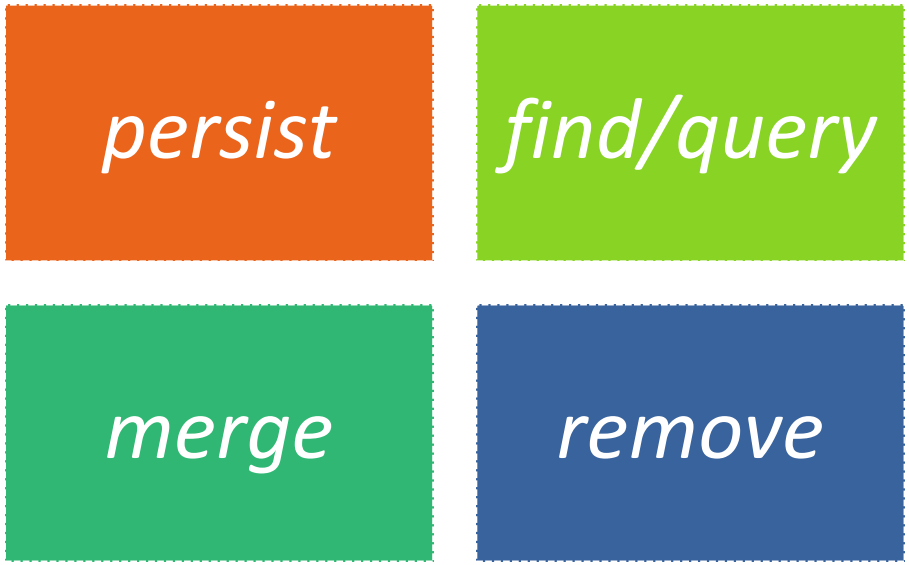


Mapeamento Objeto Relacional

- Como usar JPA - Passo a Passo

Criar classes de gerenciamento de objetos

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



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 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();
    }
}
```

Define Unidade de Persistência

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();
    }
}
```

Cria o EntityManager para manipular entidades

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();
    }
}
```

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

Mapeamento Objeto Relacional

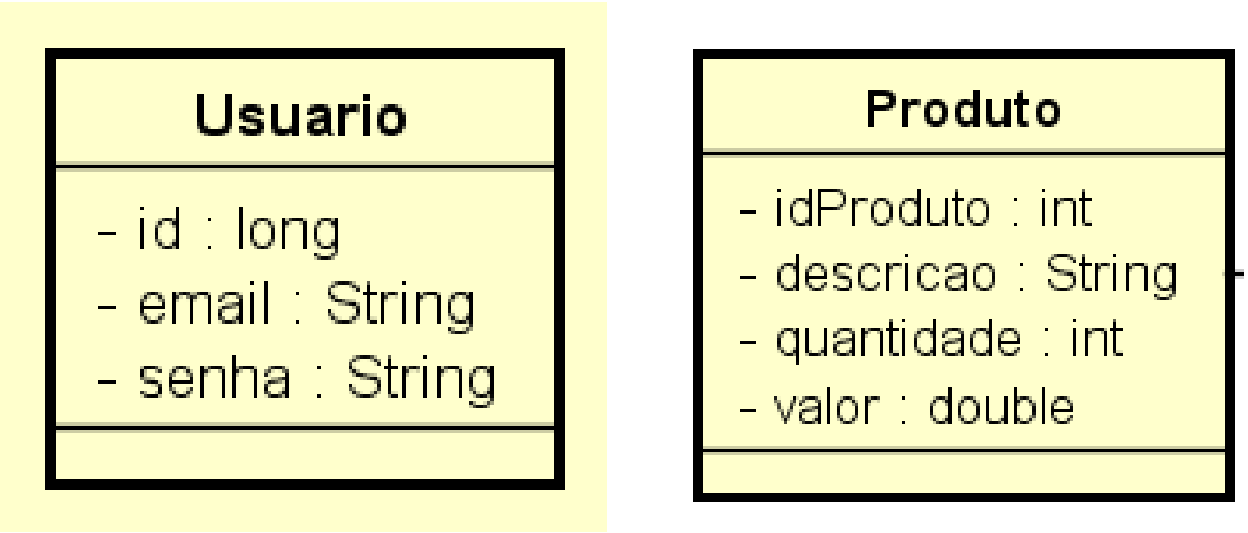
- Manipulando Entidades
  - Inserindo (*persist*)

```
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

- 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);
}
```

```java
public class BusancandoComQueryCurso {
    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

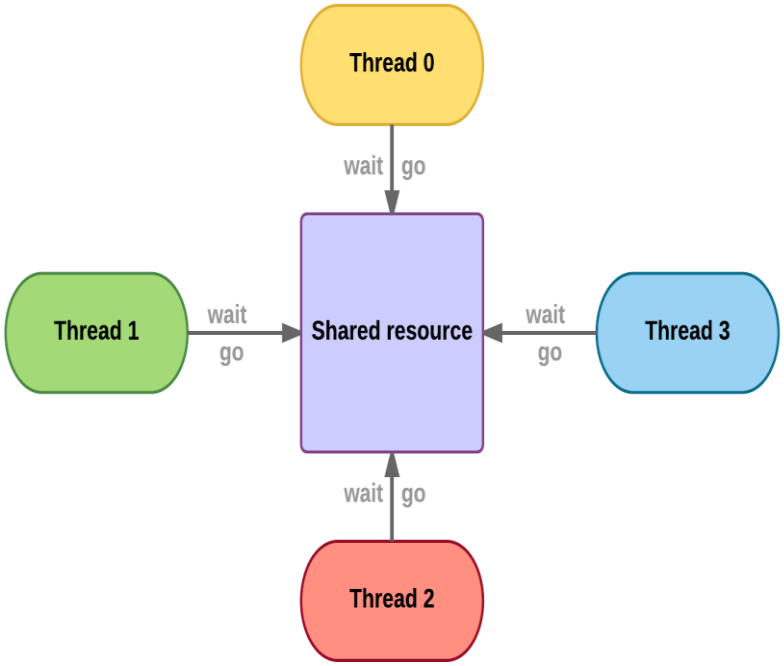
- JPQL - *Java Persistence Query Language*
  - **Resultados Especiais**
    - **List<Object[]>**
      - Algumas consultas possuem resultados complexos.

```
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();
    }
}
```

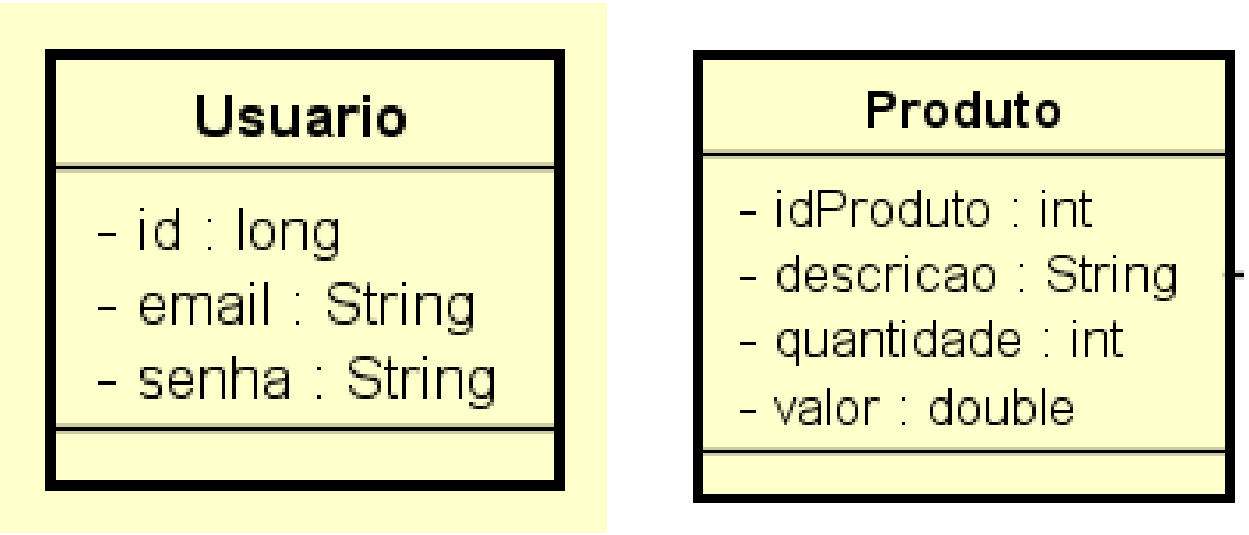
Mapeamento Objeto Relacional

- 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();
    }
}
```

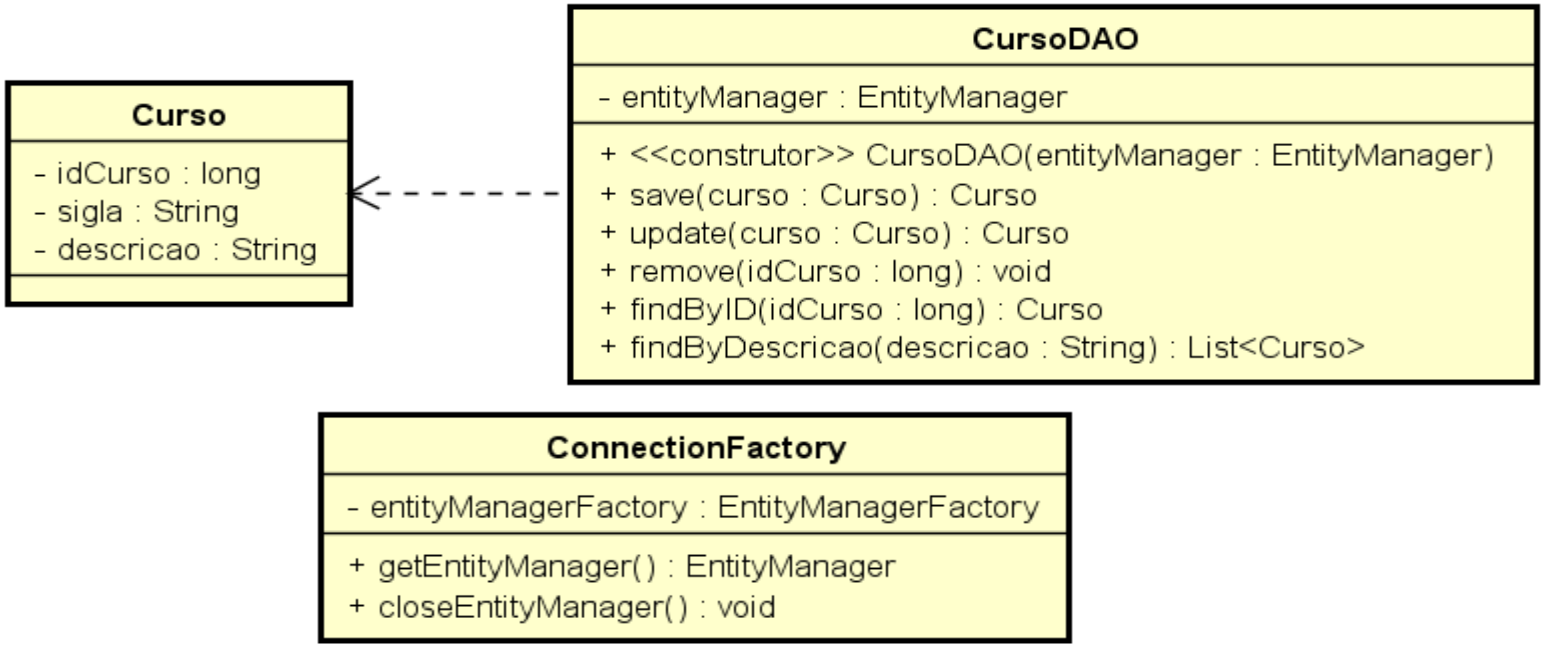
Mapeamento Objeto Relacional

- 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

- Manipulando Entidades
  - Discussão sobre classe DAO

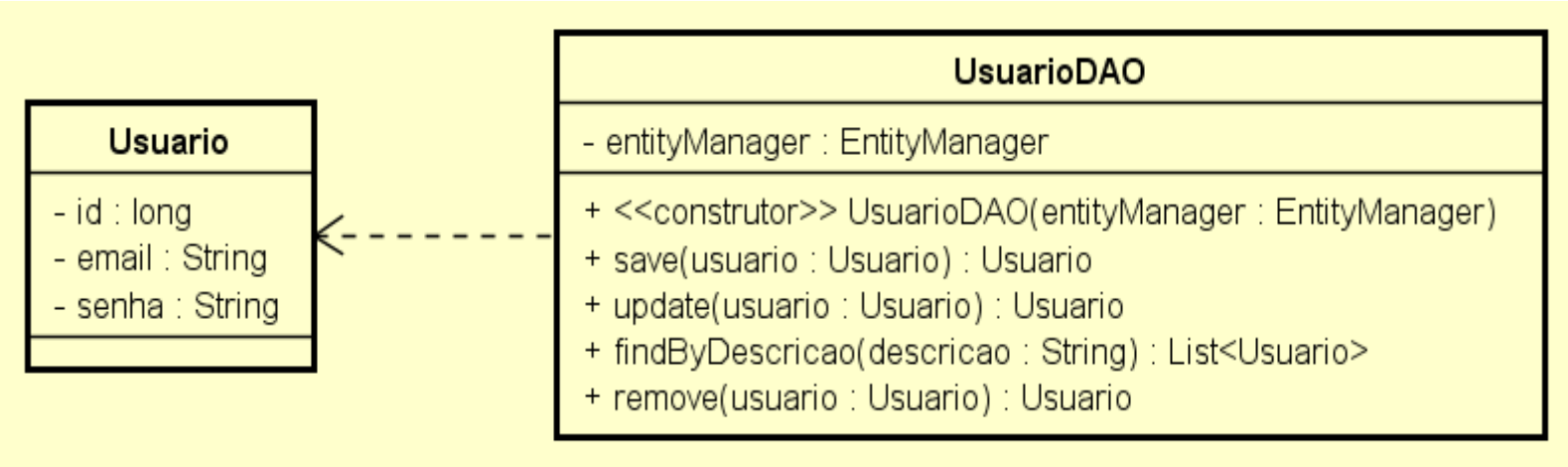


Conteúdo disponível em:

- https://github.com/joyceMiranda/codigosDeExemplo/tree/master/DRA_JPA_HIBERNATE

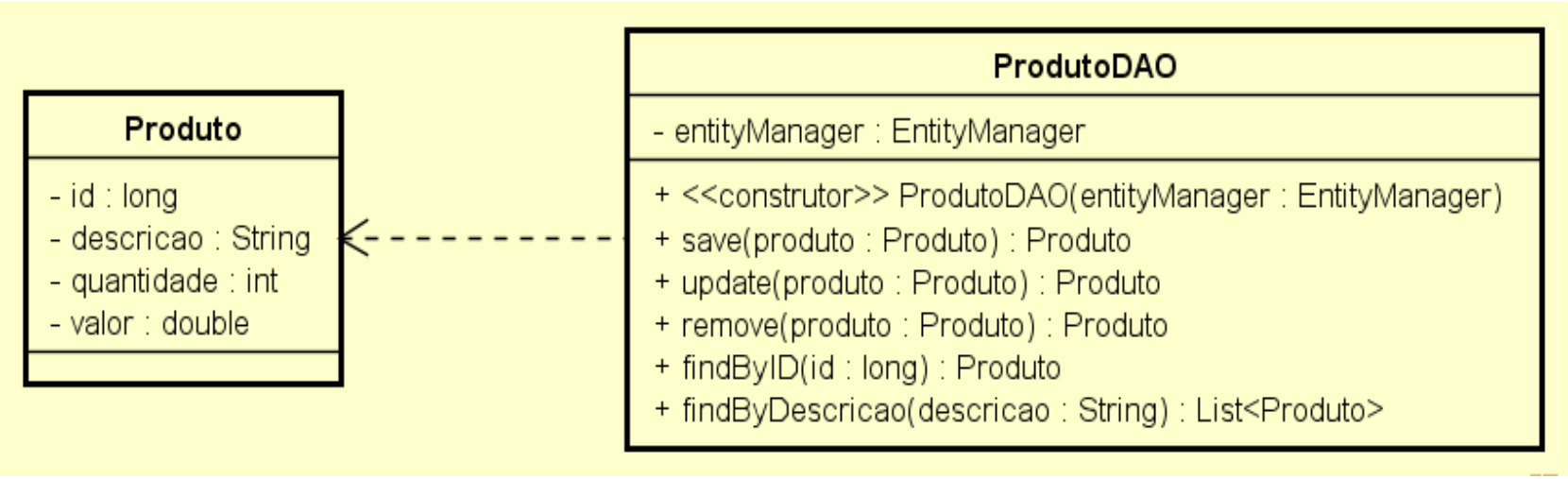
Mapeamento Objeto Relacional

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



Mapeamento Objeto Relacional

- 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 100, 150, 1000 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 que será capaz de abstrair um tipo qualquer de entidade da aplicação 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, cada uma representando um modelo, a aplicação passará a ter 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

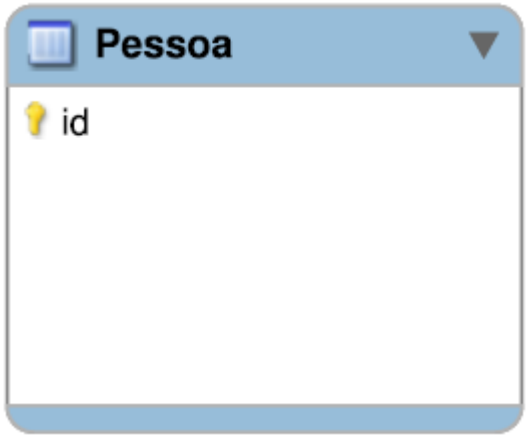
- Manipulando Entidades
  - Apresentação do DAO Genérico

Exemplo: Código disponível em:
https://github.com/joyceMiranda/codigosDeExemplo/tree/master/DRA_JPA_HIBERNATE

Mapeamento Objeto Relacional

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

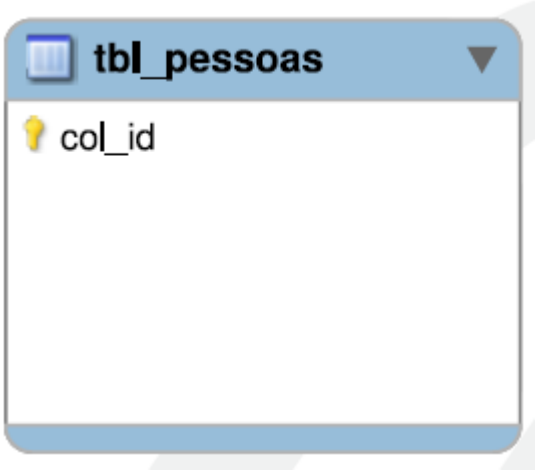
```
@Entity
class Pessoa {
  @Id
  @GeneratedValue
  private Long id;
}
```



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

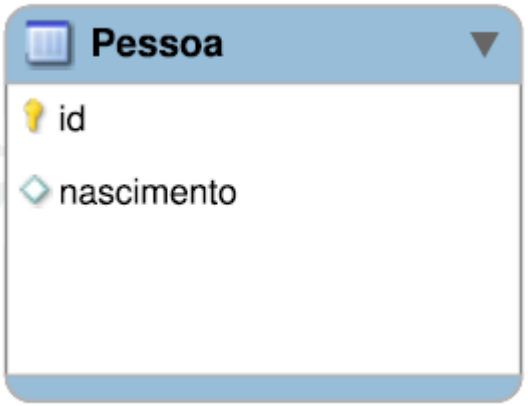
- Mapeamento
  - Objetos grandes (***Large Objects***)
    - **@LOB**
      - Imagem, música, texto
      - Aplicado em atributos dos tipos: String, byte[], Byte[], char[] ou Character[]

```
@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. Primerio -> 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 Periodo {
  MATUTINO,
  NOTURNO
}
```

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

Mapeamento Objeto Relacional

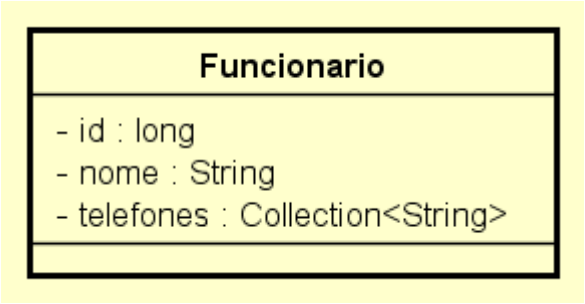
- Mapeamento
  - Tipos Enumerados
    - Solução
      - **@Enumarated:** faz com que elementos do tipo Enum sejam associados a uma String ao invés de um numero inteiro.

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

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

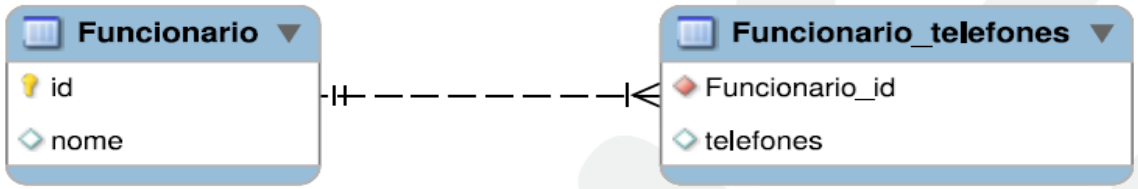
```
public enum Periodo {
  MATUTINO,
  VESPERTINO,
  NOTURNO
}
```

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 ;

}
```



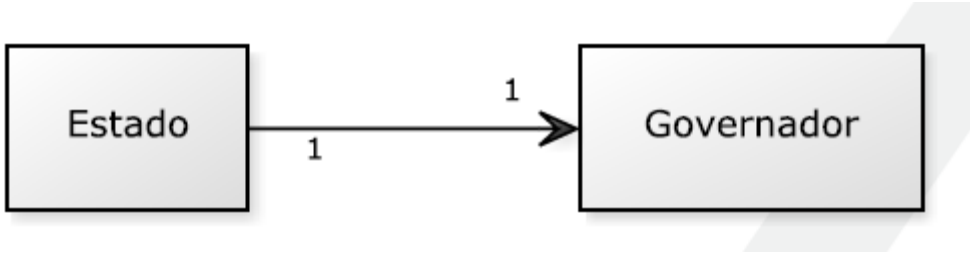
Mapeamento Objeto Relacional

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

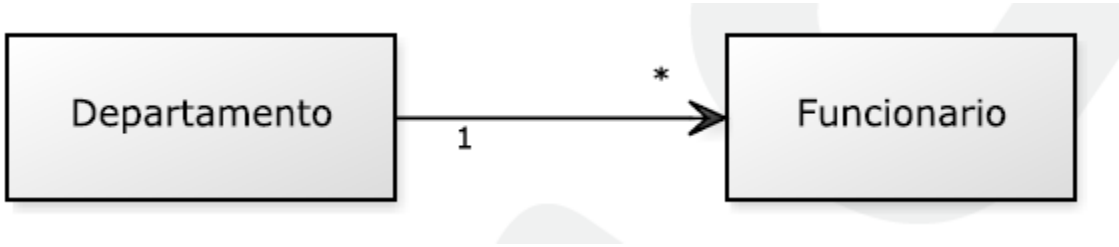
```
@Entity
public class Funcionario {

  @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)**

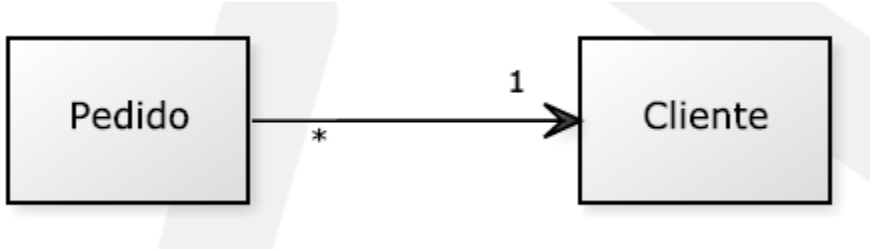


      - ***One To Many*** **(Um pra Muitos)**



Mapeamento Objeto Relacional

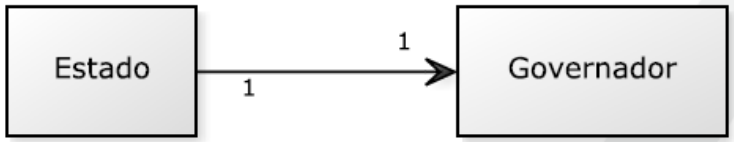
- Mapeamento
  - Relacionamentos
    - **Tipos**
      - ***Many To One*** **(Muitos pra Um)**



      - ***Many To Many*** **(Muitos pra Muitos)**

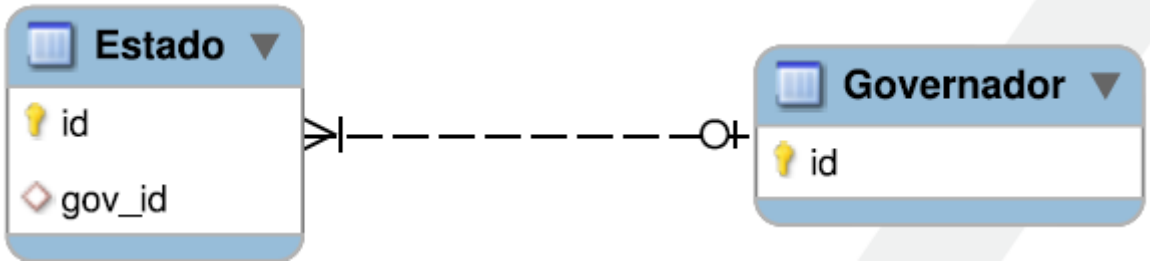
Mapeamento Objeto Relacional

- Mapeamento
  - Relacionamentos
    - ***One To One*** **(Um pra Um)**



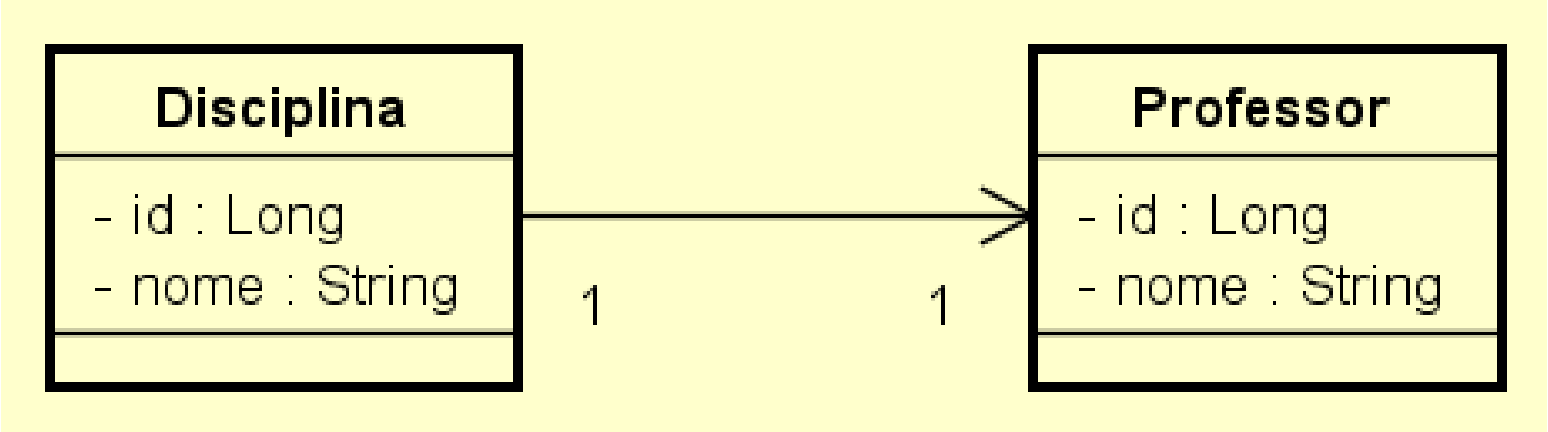

```
@Entity
class Estado {
  @Id
  @GeneratedValue
  private Long id;

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

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



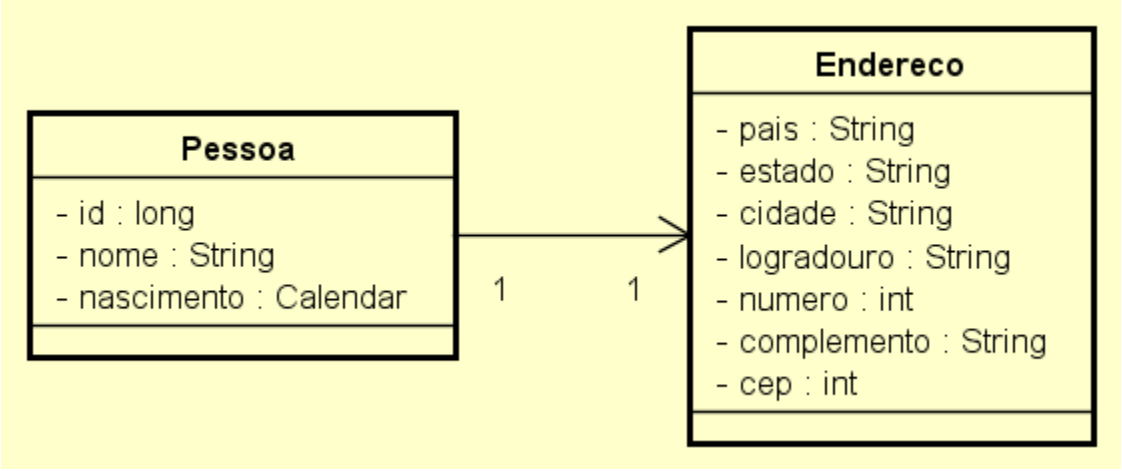
Mapeamento Objeto Relacional

- Pratique!
  - Mapeamento – Relacionamentos – ***One To One***



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.



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

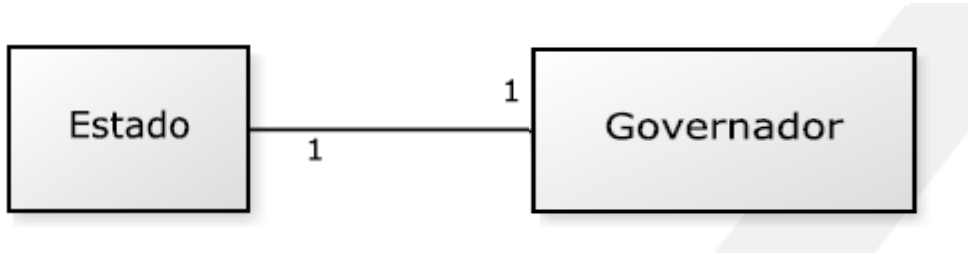
Substitui @Entity por @Embedable, que indica que é uma classe embutida.

```
@Embeddable
class Endereco {

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

}
```

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

Mapeamento Objeto Relacional

- Mapeamento
  - Relacionamentos
    - **Bidirecional: Sentido 1**

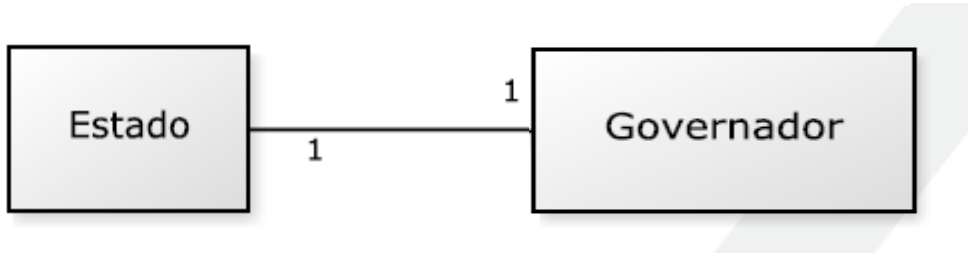



```
@Entity
class Estado {
  @Id
  @GeneratedValue
  private Long id;

  @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**

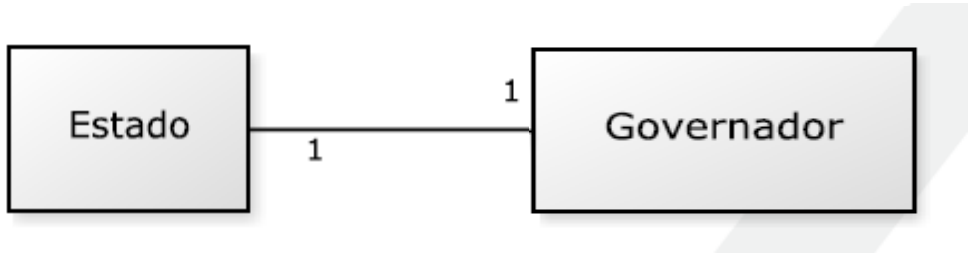



```
@Entity
class Governador {
  @Id
  @GeneratedValue
  private Long id;

  @OneToOne
  private Estado estado;

  // GETTERS E SETTERS
}
```

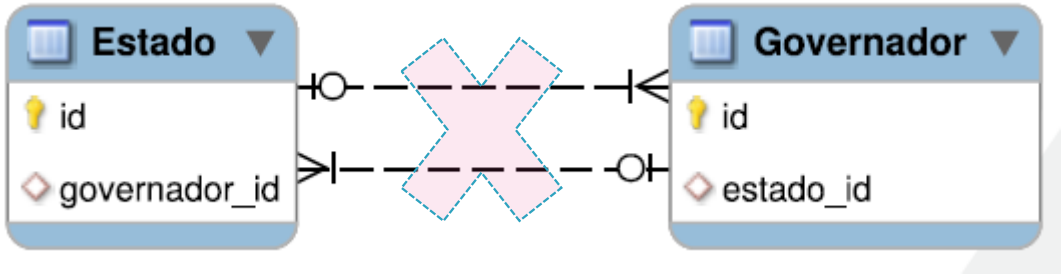
```
Governador g = manager.find(Governador.class, 1L);
Estado e = g.getEstado();
```

Mapeamento Objeto Relacional

- Mapeamento
  - Relacionamentos
    - **Bidirecional**

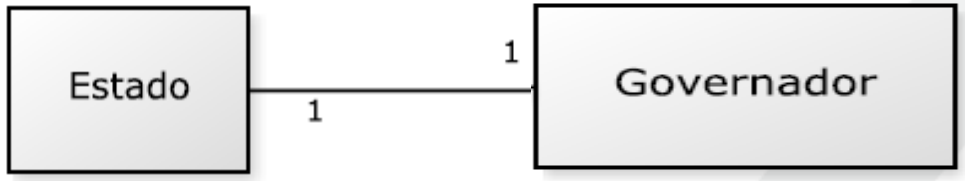


    - **Problema:** São criadas duas colunas de relacionamento, quando deveria existir apenas uma.



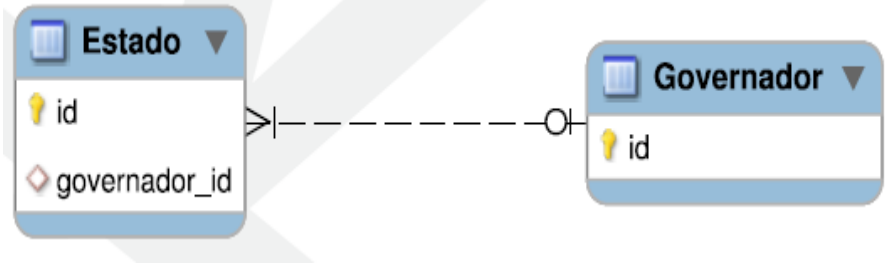
Mapeamento Objeto Relacional

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



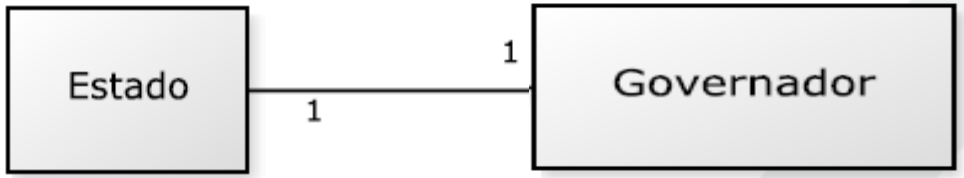

```
@Entity
class Governador {
  @Id
  @GeneratedValue
  private Long id;

  @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.

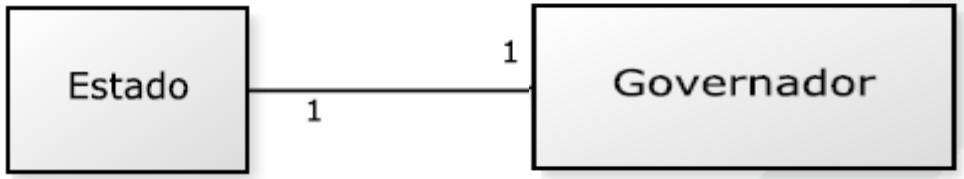
```
manager.getTransaction().begin();

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.

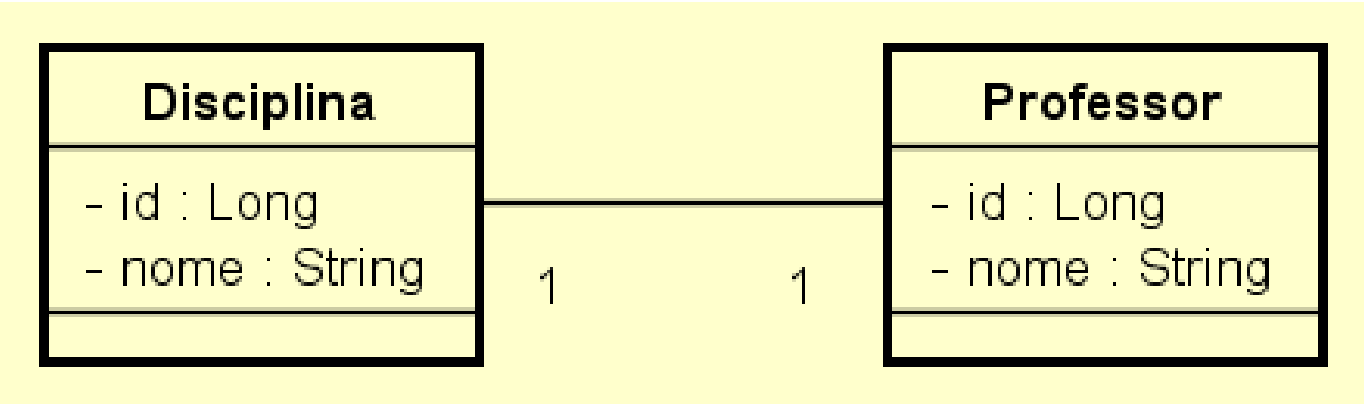



```
@Entity
class Estado {
  @Id
  @GeneratedValue
  private Long id;

  @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

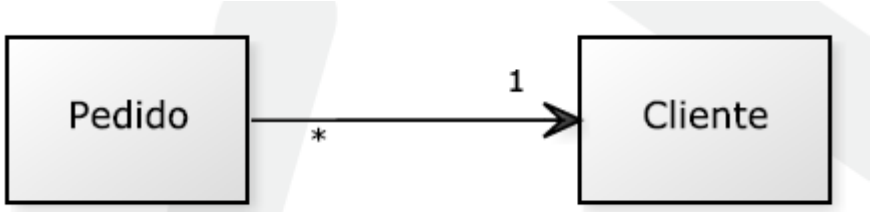
- Pratique!
  - Mapeamento – Relacionamentos – ***One To One – Bidirecional***



  - **Crie uma classe de persistência para inserir no BD objetos do tipo Professor e Disciplina.**

Mapeamento Objeto Relacional

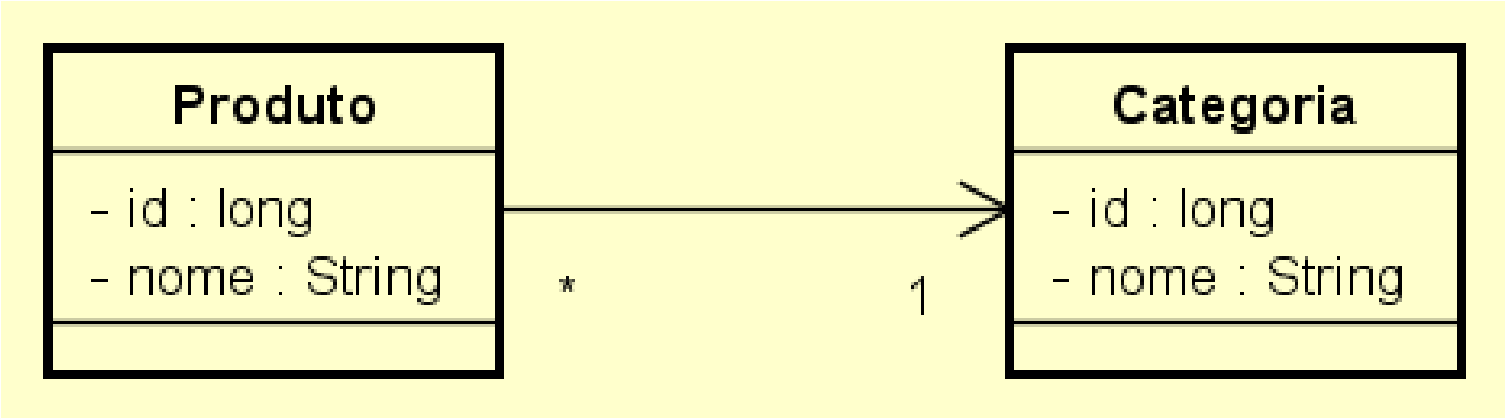
- Mapeamento
  - Relacionamentos
    - ***Many To One*** **(Muitos pra Um)**




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

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

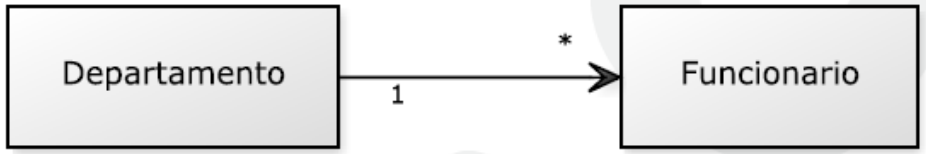
Mapeamento Objeto Relacional

- Pratique!
  - Mapeamento – Relacionamentos – ***Many To One***

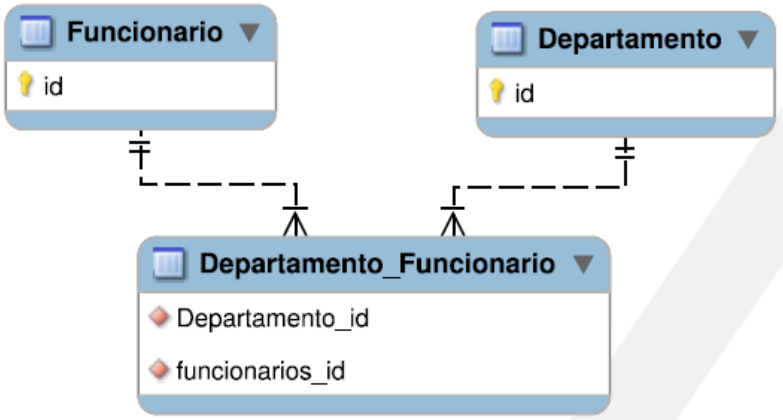


Mapeamento Objeto Relacional

- Mapeamento
  - Relacionamentos
    - ***One To Many* (Um pra Muitos)**



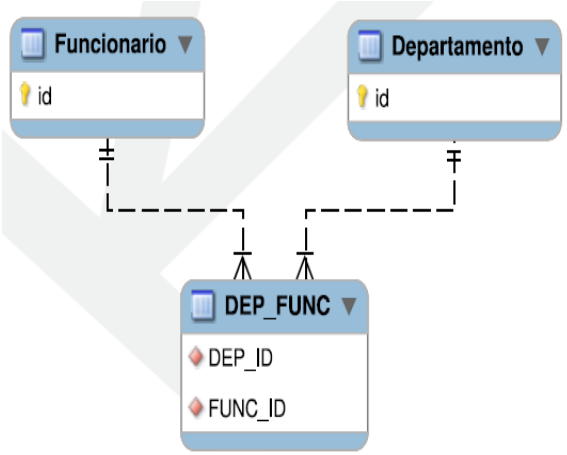

```
@Entity
class Departamento {
  @Id
  @GeneratedValue
  private Long id;

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



Mapeamento Objeto Relacional

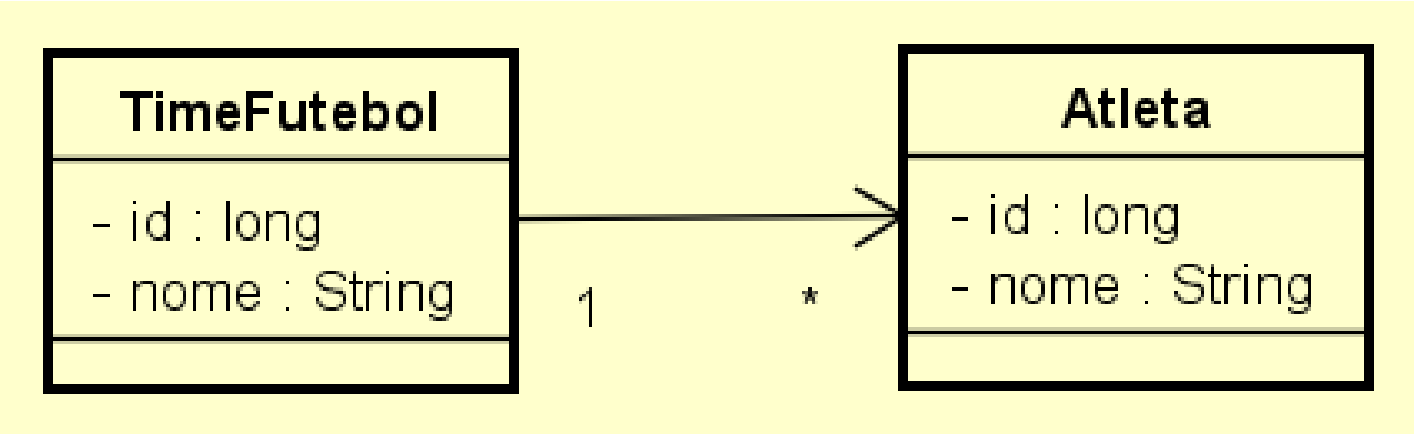
- Mapeamento
  - Relacionamentos
    - ***One To Many*** **(Um pra Muitos)**

```
@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

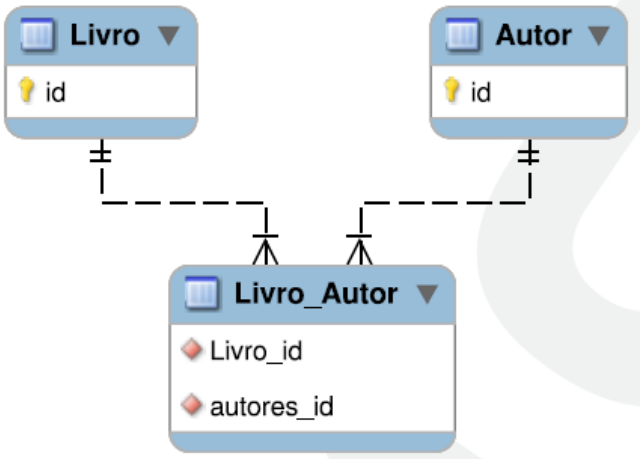
- Pratique!
  - Mapeamento – Relacionamentos – ***One To Many***



Mapeamento Objeto Relacional

- Mapeamento
  - Relacionamentos
    - ***Many To Many* (Muitos pra Muitos)**

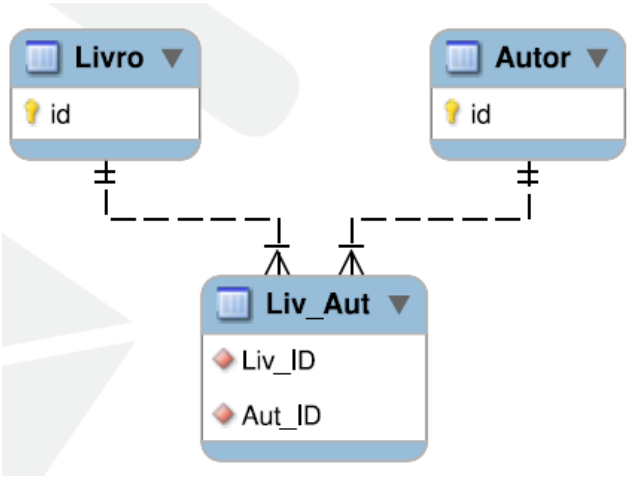

```
@Entity
class Livro {
  @Id
  @GeneratedValue
  private Long id;

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



Mapeamento Objeto Relacional

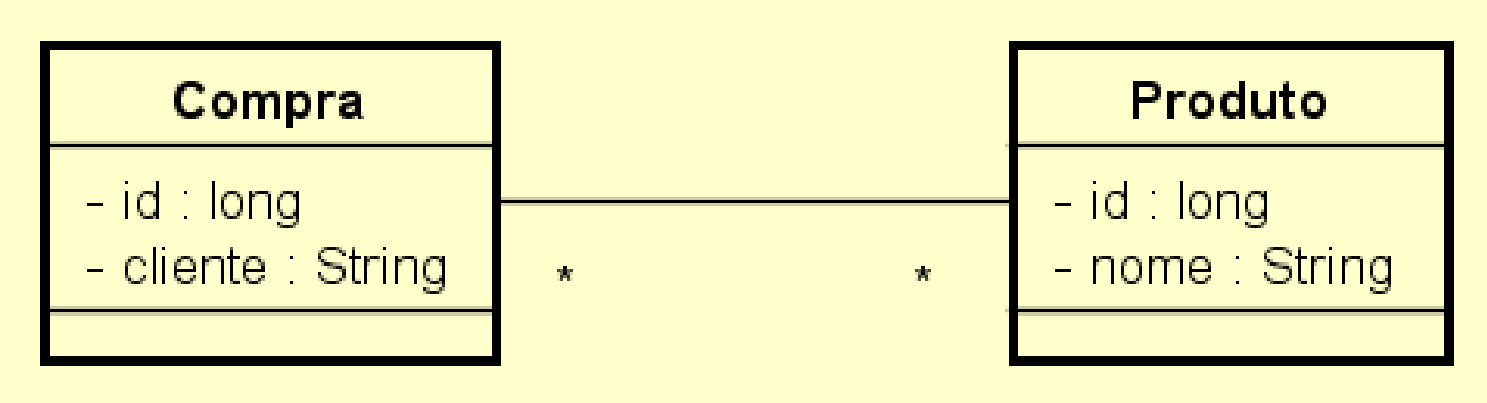
- Mapeamento
  - Relacionamentos
    - ***Many To Many*** **(Muitos pra Muitos)**

```
@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

- Pratique!
  - Mapeamento – Relacionamentos – ***Many To Many***



Mapeamento Objeto Relacional

- Mapeamento
  - Herança
    - JPA define três estratégias para o mapeamento de herança.

Single Table

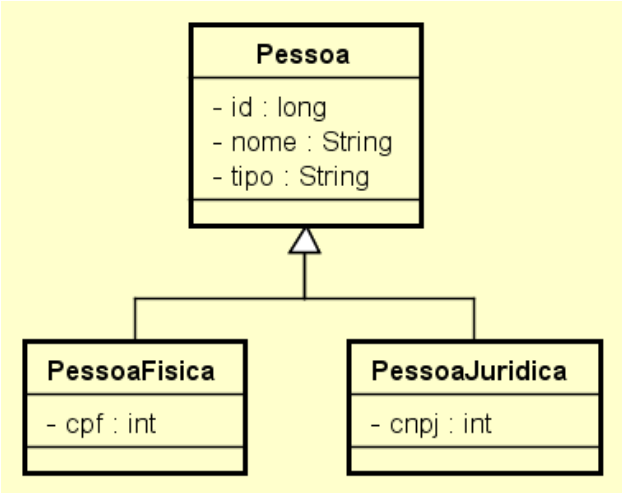
Joined

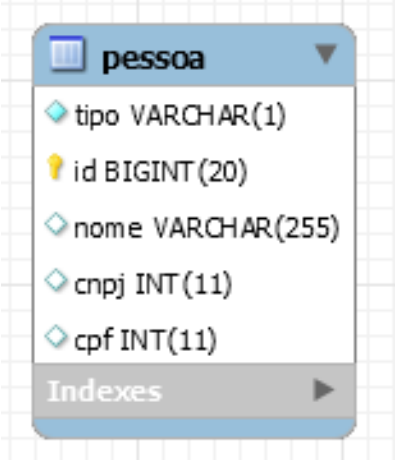
Table per Class

Mapeamento Objeto Relacional

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

Single Table





Single Table

```java
@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;

}
```

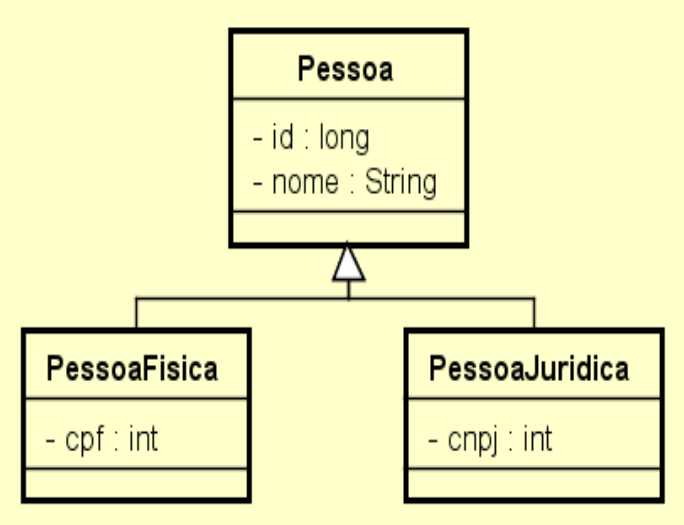
```java
@Entity
@DiscriminatorValue(value = "F")
public class PessoaFisica extends Pessoa {

    private int cpf;

}
```

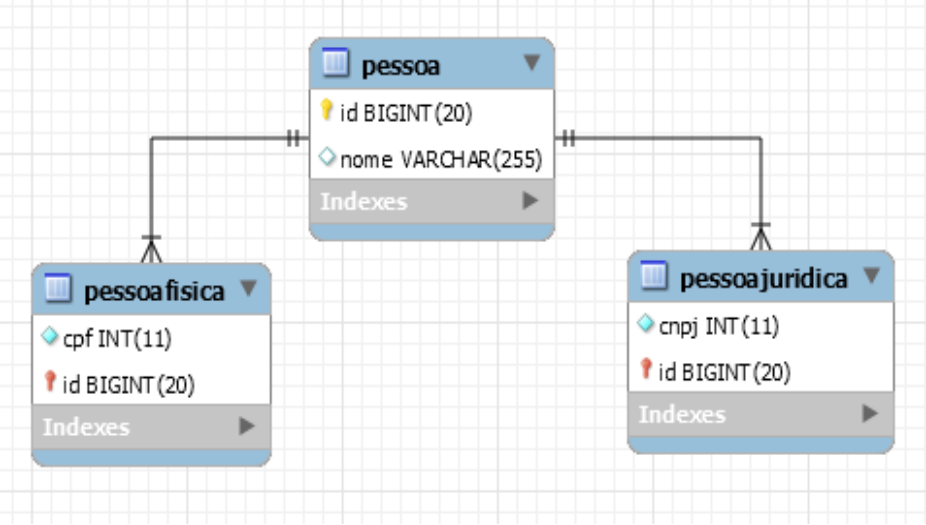
```java
@Entity
@DiscriminatorValue(value = "J")
public class PessoaJuridica extends Pessoa {

    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

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

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

}
```

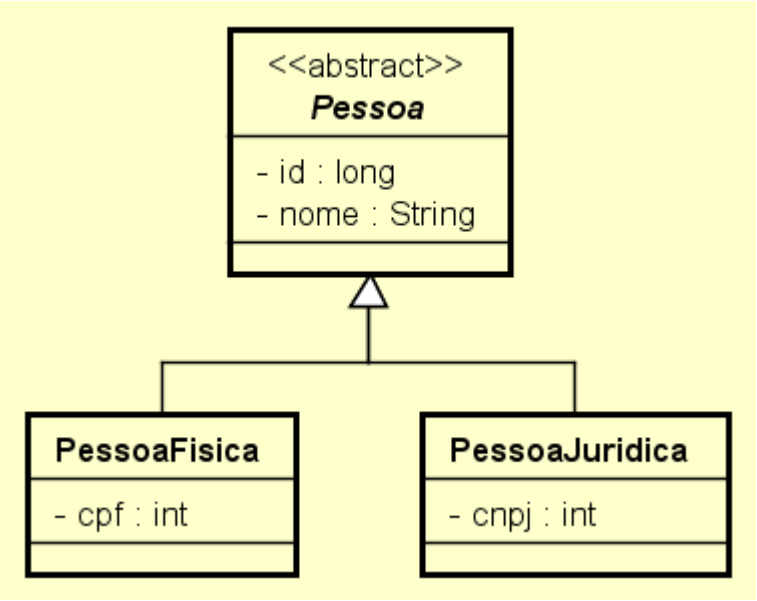
```java
@Entity
public class PessoaFisica extends Pessoa {

    private int cpf;

}
```

```java
@Entity
public class PessoaJuridica extends Pessoa {

    private int cnpj;

}
```

Mapeamento Objeto Relacional

Table per Class

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



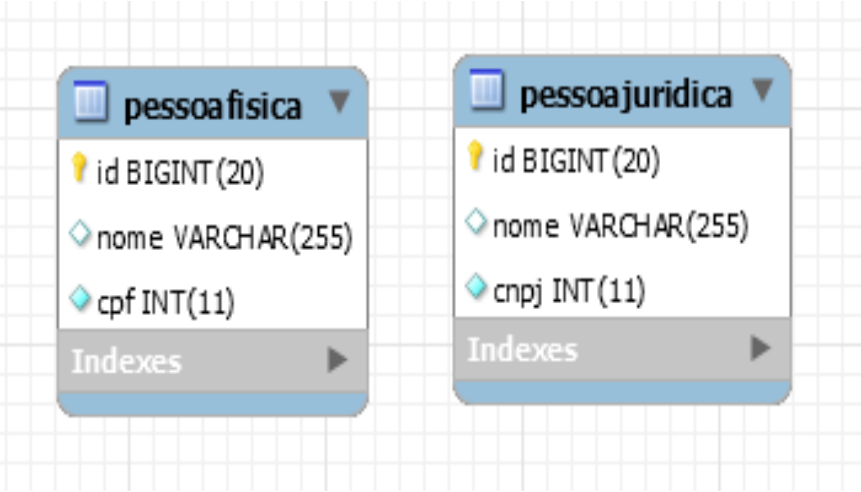


Table per Class

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

    @Id
    private Long id;
    private String nome;

}
```

```java
@Entity
public class PessoaFisica extends Pessoa {

    private int cpf;

}
```

```java
@Entity
public class PessoaJuridica extends Pessoa {

    private int cnpj;

}
```

Mapeamento Objeto Relacional

PROGRAMAÊ!

- Tarefa de Implementação
  - Considerando o modelo abaixo, utilizando o Hibernate:
    - Faça o mapeamento objeto-relacional;
    - Implemente a classe de gerenciamento de objetos;
    - Crie uma interface gráfica, para executar métodos de inserção (*insert*) dos objetos.