INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

PRW3 - Programação para a WEB III

JPA - parte final

Conteúdo 04

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

AVISO!! (será repetido todas as aulas)

- Mantenha a disposição de cabos nos computadores dos labs.
- Se retirar um cabo de rede (para usar notebook),
  não esqueça de repor no computador ao final da aula.
- Mantenha o filtro de linha e as tomadas, atrás da mesa,
  em seu estado original, especialmente para que não possa ser esbarrado por outros alunos.
- Não altere de jeito nenhum os cabos atrás dos computadores,
  especialmente mexendo nas “travas”.
- **Desliguem a(s) máquina(s) antes de sair.**
- Lembre-se! Os laboratórios são nossos!! Vamos cuidar!

Consultas

Uma pequena reorganizada no código do main ...

Apaguei o arquivo de dados ...

CadastroDeProduto.java

```
1   package br.edu.ifsp.carlao2005.testes;
2
3   import ...
10
11  public class CadastroDeProduto {
12
13
14      public static void main(String[] args) {}
22
23
24      ////////////////////////////////////////////////////////////
25
26
27      private static void cadastrarProdutos() {...}
56
57
58  }
```

O código a seguir tem algumas
partes comentadas,
pois só vamos usar elas
mais a frente na aula.

```java
private static void cadastrarProdutos() {

    // Criando a Categoria "CELULARES":
    Categoria celulares = new Categoria("CELULARES");
    /*  // Criando a Categoria "INFORMATICA":
    Categoria informatica = new Categoria("INFORMATICA"); */

    // Criando objetos da Classe Produto, associados a categoria CELULARES:
    Produto celular1 = new Produto("Celular UM", "Motorola",
            new BigDecimal("1000"), celulares);
    Produto celular2 = new Produto("Celular DOIS", "Samsung",
            new BigDecimal("2000"), celulares);

    /*  // Criando objetos da Classe Produto, associados a categoria INFORMATICA:
    Produto comp1 = new Produto("Deskjet 3776", "HP",
            new BigDecimal("700"), informatica);
    Produto comp2 = new Produto("Notebook i7", "DELL",
            new BigDecimal("9000"), informatica);           */

    // Criando o EntityManager:
    EntityManager em = JPAUtil.getEntityManager();

    // Criando os DAOs:
    ProdutoDao produtoDao = new ProdutoDao(em);
    CategoriaDao categoriaDao = new CategoriaDao(em);

    // Iniciando a transação:
    em.getTransaction().begin();
    // Persistindo os objetos:
    categoriaDao.cadastrar(celulares);
    //categoriaDao.cadastrar(informatica);
    produtoDao.cadastrar(celular1);
    produtoDao.cadastrar(celular2);
    //produtoDao.cadastrar(comp1);
    //produtoDao.cadastrar(comp2);
    // Finalizando (commit) a transação:
    em.getTransaction().commit();
    // Fechando o EntityManager:
    em.close();
}
```

```java
package br.edu.ifsp.carlao2005.testes;

import ...

public class CadastroDeProduto {

    public static void main(String[] args) {

        cadastrarProdutos();

    }

    ////////////////////////////////////////////////////////////////////

    private static void cadastrarProdutos() {  ...  }

}
```

Executando para criar o arquivo de dados...

SELECT * FROM CATEGORIAS;

ID	NOME
1	CELULARES

(1 row, 2 ms)

SELECT * FROM PRODUTOS;

ID	DATACADASTRO	DESCRICAO	NOME	PRECO	CATEGORIA_ID
1	2025-08-08	Motorola	Celular UM	1000.00	1
2	2025-08-08	Samsung	Celular DOIS	2000.00	1

(2 rows, 0 ms)

Busca pelo id

Na JPA o método `find()` é usado para recuperar uma entidade específica do banco de dados com base em sua chave primária. A JPA é uma especificação do Java que define um conjunto de interfaces e anotações para trabalhar com persistência de dados em aplicações Java.

A assinatura do método `find()` é a seguinte:

java Copy code

```java
public <T> T find(Class<T> entityClass, Object primaryKey);
```

Aqui está uma explicação dos parâmetros:

- `entityClass`: Representa a classe da entidade que você deseja recuperar. Por exemplo, se você tiver uma entidade `Product`, você passaria `Product.class` como o primeiro parâmetro.
- `primaryKey`: É o valor da chave primária da entidade que você deseja recuperar. A chave primária é um atributo único que identifica de forma exclusiva uma instância de uma entidade no banco de dados. Geralmente, a chave primária é um valor numérico ou uma combinação de valores que garantem a unicidade do registro.

O método `find()` consulta o banco de dados usando a chave primária fornecida e retorna a instância da entidade correspondente ou `null` se não for encontrada nenhuma correspondência. Aqui está um exemplo de como você pode usar o método `find()` para recuperar uma entidade do banco de dados:

Classe ProdutoDao

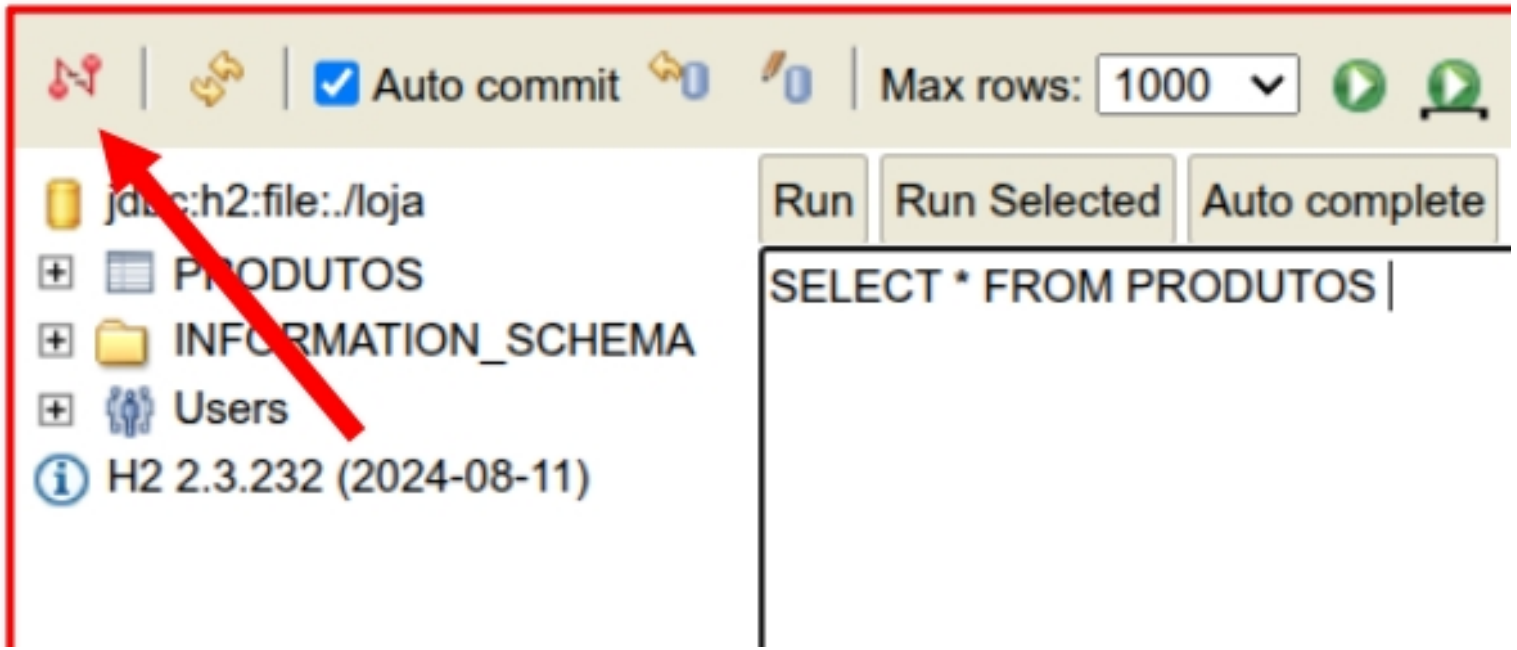
```
public class ProdutoDao {

    // EntityManager, que será usado por todos os métodos:
    private EntityManager em;

    // Construtor que já recebe o EntityManager criado:
    public ProdutoDao(EntityManager em) {
        this.em = em;
    }

    // Método para gravar um produto no BD:
    public void cadastrar(Produto produto) {
        this.em.persist(produto);
    }

    // Método para buscar um produto pelo seu id:
    public Produto buscarPorId(Long id) {

        return em.find(Produto.class, id);
    }
}
```

Lá no main() ...

```
public static void main(String[] args) {

    // cadastrarProdutos();


    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    // Buscando o produto com id = 1:
    Produto p = produtoDao.buscarPorId(1l);

    System.out.println(p.getNome());
    System.out.println(p.getDescricao());
    System.out.println(p.getDataCadastro());
    System.out.println(p.getPreco());
}
```

Lembrando...

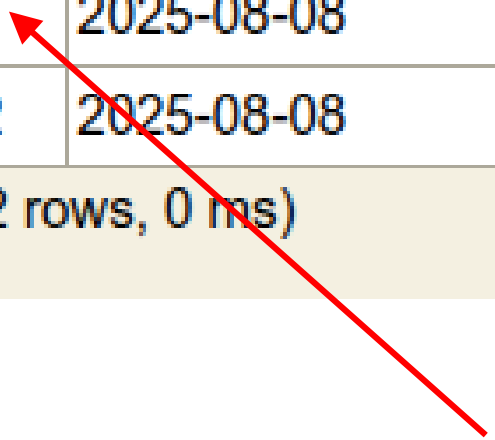
Antes de executar, não esqueça de fechar a conexão do console do H2 com o banco, senão vai receber erro de “banco em uso”.



SELECT * FROM PRODUTOS;

ID	DATACADASTRO	DESCRICAO	NOME	PRECO	CATEGORIA_ID
1	2025-08-08	Motorola	Celular UM	1000.00	1
2	2025-08-08	Samsung	Celular DOIS	2000.00	1

(2 rows, 0 ms)



```
Feb 20, 2025 3:01:50 PM org.hibernate.engine.transaction.jta.pla
INFO: HHH000489: No JTA platform available (set 'hibernate.transa
Feb 20, 2025 3:01:50 PM org.hibernate.resource.transaction.backe
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess
Hibernate: select p1_0.id,c1_0.id,c1_0.nome,p1_0.dataCadastro,p1_
Celular UM
Motorola
2025-02-20
1000.00

Process finished with exit code 0
```

Se não achar o registro,
retorna null

SELECT * FROM PRODUTOS;

ID	DATACADASTRO	DESCRICAO	NOME	PRECO	CATEGORIA_ID
1	2025-08-08	Motorola	Celular UM	1000.00	1
2	2025-08-08	Samsung	Celular DOIS	2000.00	1

(2 rows, 0 ms)

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    // Buscando o produto com id = 3:
    Produto p = produtoDao.buscarPorId(3l);

    if (p == null)
        System.out.println("\n\nProduto não encontrado!");
    else {
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }
}
```

SELECT * FROM PRODUTOS;

ID	DATACADASTRO	DESCRICAO	NOME	PRECO	CATEGORIA_ID
1	2025-08-08	Motorola	Celular UM	1000.00	1
2	2025-08-08	Samsung	Celular DOIS	2000.00	1

(2 rows, 0 ms)

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    // Buscando o produto com id = 3:
    Produto p = produtoDao.buscarPorId(3l);

    if (p == null)
        System.out.println("\n\nP
    else {
        System.out.println(p.getN
        System.out.println(p.getD
        System.out.println(p.getD
        System.out.println(p.getP
    }
}
```

```
2024-02-16T12:57:40.611-03:00  INFO 16132 --- [
Hibernate: select p1_0.id,c1_0.id,c1_0.nome,p1_0.



Produto não encontrado!
```

Lembrando do ciclo de vida das entidades
(aula passada).

Ler um objeto do banco, torna ele “managed”.

Podemos então alterar os atributos
diretamente no objeto,
que automaticamente as alterações
são replicadas no Banco de Dados.
(não esqueca da necessidade de uma transação).

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    // Buscando o produto com id = 1:
    Produto p = produtoDao.buscarPorId(1l);

    em.getTransaction().begin();
    p.setDescricao("xxxxxxxxxxxxxxxxxx");
    em.getTransaction().commit();
}
```

```java
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    // Buscando o produto com id = 1:
    Produto p = produtoDao.buscarPorId(1l);

    em.getTransaction().begin();
    p.setDescricao("xxxxxxxxxxxxxxxxxx");
    em.getTransaction().commit();
}
```

```
2024-02-16T13:01:40.055-03:00  INFO 16316 --- [           main] org.hiber
2024-02-16T13:01:40.762-03:00  INFO 16316 --- [           main] o.h.e.t.j
2024-02-16T13:01:40.770-03:00  INFO 16316 --- [           main] org.hiber
Hibernate: select p1_0.id,c1_0.id,c1_0.nome,p1_0.dataCadastro,p1_0.descri
Hibernate: update produtos set categoria_id=?,dataCadastro=?,descricao=?,
```

SELECT * FROM PRODUTOS;

ID	DATACADASTRO	DESCRICAO	NOME	PRECO	CATEGORIA_ID
1	2025-08-08	xxxxxxxxxxxxxxxxxx	Celular UM	1000.00	1
2	2025-08-08	Samsung	Celular DOIS	2000.00	1

(2 rows, 1 ms)

```
    // Buscando o produto com id = 1:
    Produto p = produtoDao.buscarPorId(1l);

    em.getTransaction().begin();
    p.setDescricao("xxxxxxxxxxxxxxxxxxx");
    em.getTransaction().commit();
}
```

```
2024-02-16T13:01:40.055-03:00  INFO 16316 --- [           main] org.hiber
2024-02-16T13:01:40.762-03:00  INFO 16316 --- [           main] o.h.e.t.j
2024-02-16T13:01:40.770-03:00  INFO 16316 --- [           main] org.hiber
Hibernate: select p1_0.id,c1_0.id,c1_0.nome,p1_0.dataCadastro,p1_0.descri
Hibernate: update produtos set categoria_id=?,dataCadastro=?,descricao=?,
```

Outras leituras.... SQL !

Outras leituras.... ~~SQL~~ !

Na verdade, **JPQL** !!

Dentro da Jakarta Persistence (anteriormente conhecida como Java Persistence API - JPA), a JPQL (Java Persistence Query Language) é uma linguagem de consulta orientada a objetos que permite que você realize consultas em entidades persistentes (objetos) em vez de escrever consultas SQL diretas.

A JPQL é independente de banco de dados, ou seja, suas consultas são feitas em relação às entidades do seu modelo de dados, não importando qual banco de dados subjacente está sendo utilizado. Isso torna o código mais portável, já que as consultas não precisam ser alteradas quando se muda de banco de dados.

Alguns conceitos importantes sobre a JPQL incluem:

1. **Consulta de Entidades**: A JPQL permite que você recupere entidades persistidas (objetos) diretamente, especificando uma consulta em termos das classes de entidade e seus atributos.
2. **Sintaxe Simples**: A sintaxe da JPQL é similar à sintaxe SQL, mas ao invés de operar com tabelas e colunas do banco de dados, você opera com as classes e atributos do seu modelo de dados.

Muito parecida com SQL...
mas não é SQL!

SQL trabalha com campos em tabelas.
JPQL trabalha com objetos e classes.

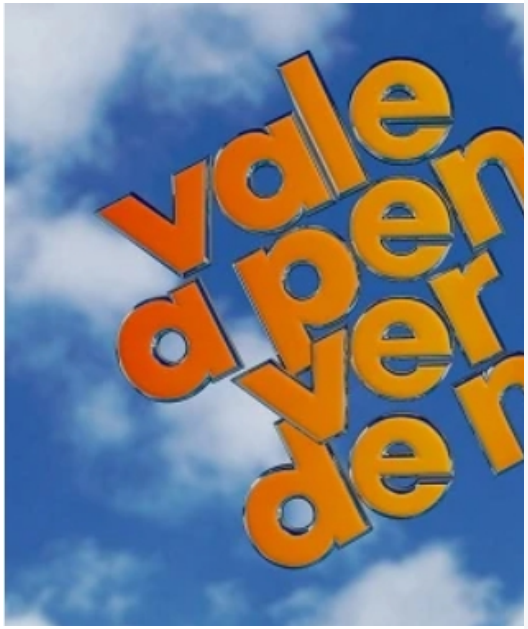
Lembrando... tipo List (java.util)

Em Java, `List` é uma interface definida no pacote `java.util` que representa uma coleção ordenada (ou seja, mantém a ordem de inserção) de elementos, onde elementos podem se repetir. É uma das principais interfaces do framework de coleções do Java e fornece métodos para adicionar, remover e acessar elementos da lista.

Aqui estão algumas características e principais métodos da interface `List`:

1. **Ordem de Inserção**: A lista mantém a ordem em que os elementos foram inseridos.
2. **Permite Duplicatas**: A lista permite armazenar elementos duplicados, ou seja, o mesmo elemento pode ocorrer mais de uma vez na lista.
3. **Índices**: Cada elemento da lista é associado a um índice inteiro, começando do índice 0 para o primeiro elemento.

Alguns dos principais métodos da interface `List` incluem:



- `add(E element)`: Adiciona o elemento especificado no final da lista.
- `add(int index, E element)`: Adiciona o elemento especificado na posição index da lista.
- `remove(int index)`: Remove o elemento da posição index da lista.
- `get(int index)`: Retorna o elemento da posição index da lista.
- `size()`: Retorna o número de elementos na lista.
- `indexOf(Object o)`: Retorna o índice da primeira ocorrência do elemento especificado na lista, ou -1 se o elemento não estiver na lista.
- `contains(Object o)`: Verifica se o elemento especificado está presente na lista.

A interface `List` é implementada por várias classes concretas no Java, incluindo `ArrayList`, `LinkedList`, `Vector`, entre outras. Cada classe de implementação possui suas próprias características e eficiências, permitindo que você escolha a melhor opção de acordo com os requisitos da sua aplicação.

createQuery()

O método `createQuery()` da Jakarta Persistence (anteriormente conhecida como Java Persistence API - JPA) é usado para criar uma consulta em JPQL (Java Persistence Query Language) a ser executada no banco de dados. Esse método é definido na interface `EntityManager`, que é responsável por interagir com as entidades do banco de dados.

A assinatura do método `createQuery()` é a seguinte:

```
public <T> TypedQuery<T> createQuery(String jpqlString, Class<T> resultClass);
```

Parâmetros:

- `jpqlString`: Uma string que contém a consulta em JPQL a ser criada. Essa string deve seguir a sintaxe válida da JPQL para formar a consulta desejada.
- `resultClass`: A classe da entidade (ou outra classe) que você espera que seja retornada como resultado da consulta. Isso é necessário porque a JPQL retorna um tipo de resultado `Object` por padrão, e o parâmetro `resultClass` é usado para informar ao JPA o tipo correto de resultado que você espera.

getResultList()

O método `getResultList()` da Jakarta Persistence (anteriormente conhecida como Java Persistence API - JPA) é usado para executar uma consulta e obter os resultados em forma de lista. Esse método é definido na interface `TypedQuery<T>`, que representa uma consulta com um tipo específico de resultado.

A assinatura do método `getResultList()` é a seguinte:

```
List<T> getResultList();
```

Retorno:

- `List<T>`: Uma lista contendo os resultados da consulta, onde `T` é o tipo de resultado especificado ao criar a consulta usando `createQuery()`.

O método `getResultList()` é comumente usado para executar consultas de seleção em JPQL e obter os resultados correspondentes. Quando a consulta é executada, o resultado é retornado como uma lista de objetos do tipo especificado na consulta.

Retornando uma lista
com todos os registros.

Classe ProdutoDao

```
// Método para buscar todos os produtos do BD.
public List<Produto> buscarTodos() {

    // Vamos usar JPQL.
    // Em SQL seria: select * from produtos
    // Na JPQL: o objeto p da classe (entidade) Produto:
    String jpql = "SELECT p FROM Produto p";
    return em.createQuery(jpql, Produto.class).getResultList();
}
```

main()

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    List<Produto> todos = produtoDao.buscarTodos();

    for(Produto p : todos) {
        System.out.println("-----------------------");
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }
}
```

main()

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    List<Produto> todos = produtoDao.buscarTodos();

    for(Produto p : todos) {
        System.out.println("-----------------------");
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }
}
```

```
-----------------------
Celular UM
xxxxxxxxxxxxxxxxxx
2025-02-20
1000.00
-----------------------
Celular DOIS
Samsung
2025-02-20
2000.00

Process finished with exit code 0
```

Obviamente, se não forem retornados elementos,
a lista fica vazia
(pode testar com **`isEmpty()`**).

Filtrando por algum parâmetro

Parâmetros na JPQL

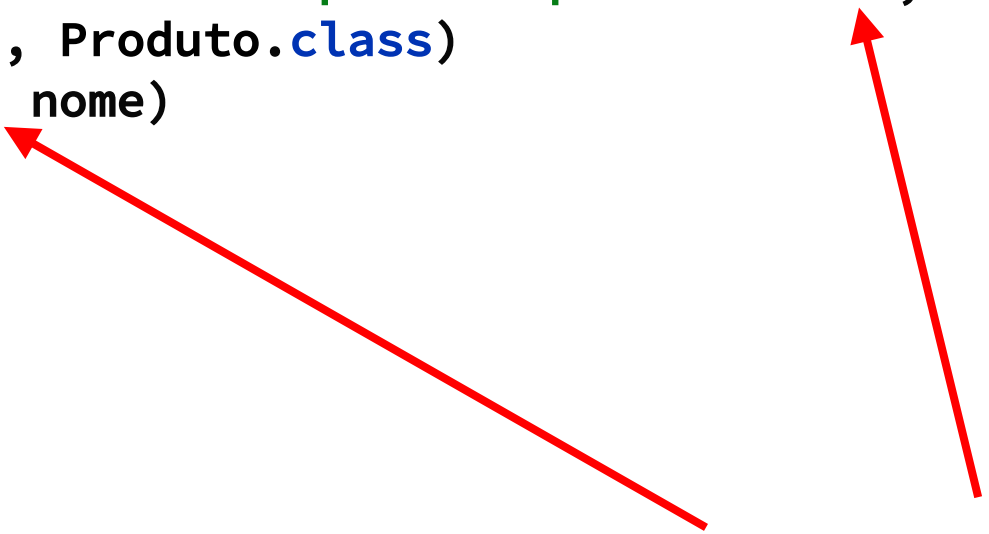
Os parâmetros na Jakarta Persistence Query Language (JPQL) são usados para criar consultas dinâmicas, onde você pode fornecer valores variáveis para critérios de busca, sem a necessidade de concatenar valores diretamente na string da consulta. Isso torna suas consultas mais seguras contra ataques de injeção de SQL e mais flexíveis, permitindo que você reutilize a mesma consulta com diferentes valores de parâmetros.

Para utilizar parâmetros na JPQL, você deve seguir os seguintes passos:

1. **Definir o Parâmetro na Consulta**: Na string da consulta JPQL, você deve definir um parâmetro usando o prefixo `:` seguido por um nome único para o parâmetro. Por exemplo, `:priceParam` ou `:nameParam`.
2. **Definir o Valor do Parâmetro**: Antes de executar a consulta, você deve definir o valor do parâmetro usando o método `setParameter()` do objeto `Query` ou `TypedQuery`. O método `setParameter()` recebe o nome do parâmetro e o valor que você deseja associar.

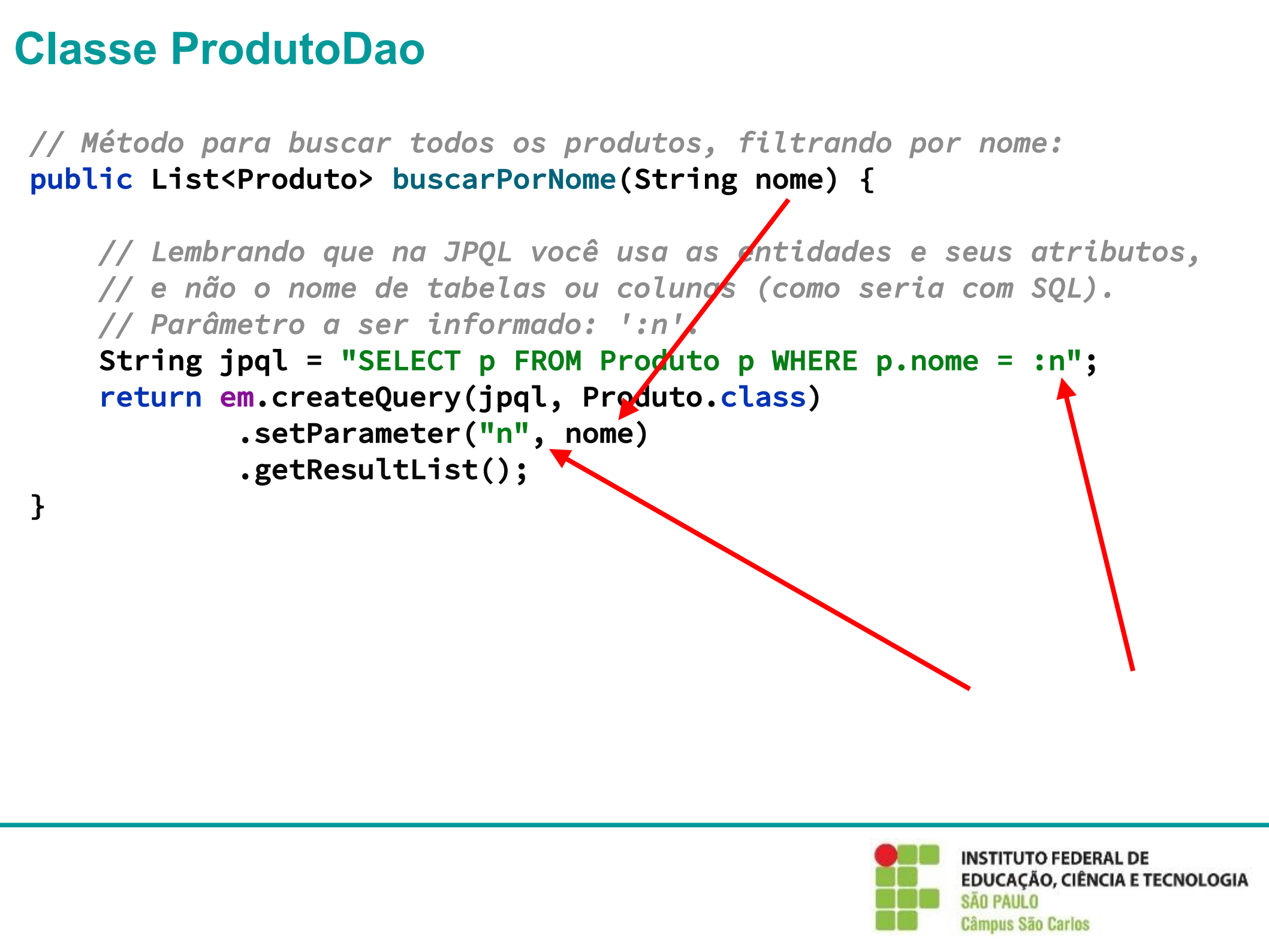
Classe ProdutoDao

```java
// Método para buscar todos os produtos, filtrando por nome:
public List<Produto> buscarPorNome(String nome) {

    // Lembrando que na JPQL você usa as entidades e seus atributos,
    // e não o nome de tabelas ou colunas (como seria com SQL).
    // Parâmetro a ser informado: ':n'.
    String jpql = "SELECT p FROM Produto p WHERE p.nome = :n";
    return em.createQuery(jpql, Produto.class)
            .setParameter("n", nome)
            .getResultList();
}
```



Classe ProdutoDao

```
// Método para buscar todos os produtos, filtrando por nome:
public List<Produto> buscarPorNome(String nome) {

    // Lembrando que na JPQL você usa as entidades e seus atributos,
    // e não o nome de tabelas ou colunas (como seria com SQL).
    // Parâmetro a ser informado: ':n'.
    String jpql = "SELECT p FROM Produto p WHERE p.nome = :n";
    return em.createQuery(jpql, Produto.class)
             .setParameter("n", nome)
             .getResultList();
}
```

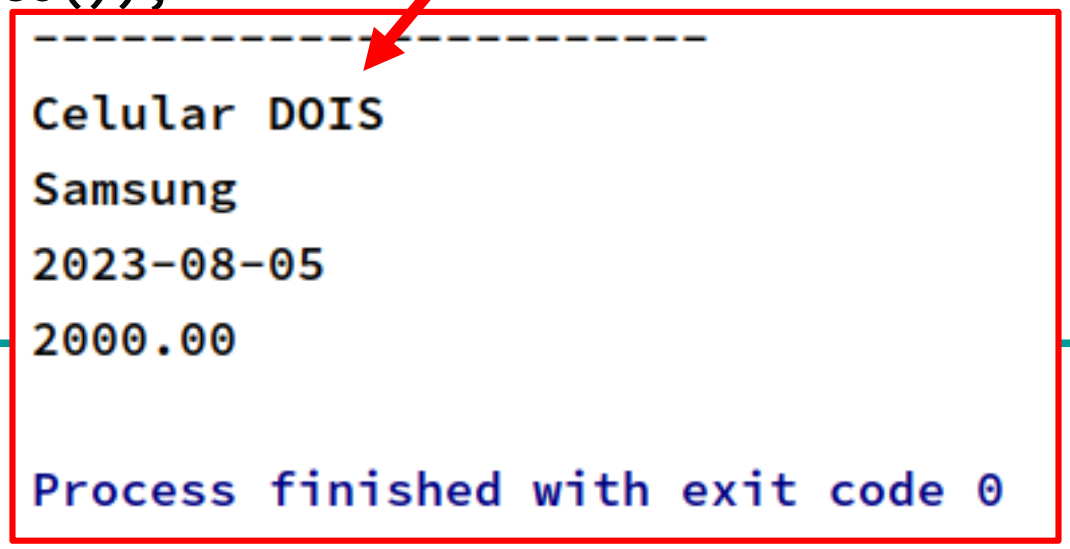


Testando no main()

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    List<Produto> todos = produtoDao.buscarPorNome("Celular DOIS");

    for(Produto p : todos) {
        System.out.println("-----------------------");
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }
}
```

Testando no main()

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    List<Produto> todos = produtoDao.buscarPorNome("Celular DOIS");

    for(Produto p : todos) {
        System.out.println("-----------------------");
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }
}
```



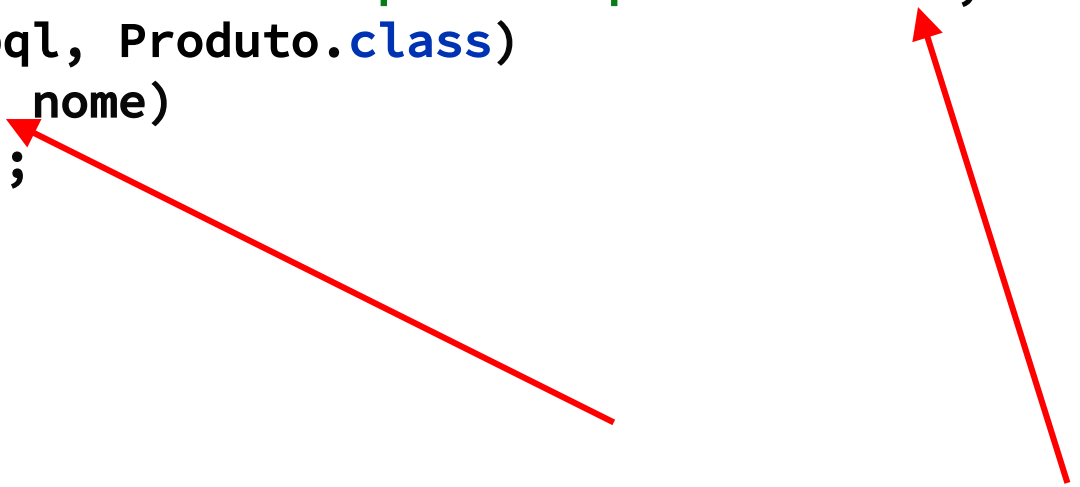
Passagem de parâmetros por números

Na Jakarta Persistence Query Language (JPQL), a passagem de parâmetros pode ser feita através de números, mas é importante notar que essa abordagem é menos comum e menos recomendada do que a passagem de parâmetros por nomes, que é mais segura e legível.

Para passar parâmetros através de números, você deve usar a notação de ponto de interrogação `?` seguida pelo índice numérico do parâmetro na consulta. Os índices começam a partir de 1 e aumentam sequencialmente para cada parâmetro adicionado à consulta.

Classe ProdutoDao

```java
// Método para buscar todos os produtos, filtrando por nome:
public List<Produto> buscarPorNome(String nome) {

    // Lembrando que na JPQL você usa as entidades e seus atributos,
    // e não o nome de tabelas ou colunas (como seria com SQL).
    // Parâmetro a ser informado: ':n'.
    String jpql = "SELECT p FROM Produto p WHERE p.nome = ?1";
    return em.createQuery(jpql, Produto.class)
             .setParameter(1, nome)
             .getResultList();
}
```



Testando no main()

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    List<Produto> todos = produtoDao.buscarPorNome("Celular DOIS");

    for(Produto p : todos) {
        System.out.println("-----------------------");
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }
}
```

Testando no main()

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    List<Produto> todos = produtoDao.buscarPorNome("Celular DOIS");

    for(Produto p : todos) {
        System.out.println("-----------------------");
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }
}
```

```
-----------------------
Celular DOIS
Samsung
2023-08-05
2000.00


Process finished with exit code 0
```

O exemplo anterior retorna uma lista,
com apenas um objeto dentro dela.

Para retornar apenas um único objeto, use

`getSingleResult()`

O método `getSingleResult` é um método da API JPA (Jakarta Persistence) que é usado para executar uma consulta JPQL (Java Persistence Query Language) e obter um único resultado como resultado da consulta. Ele é geralmente usado quando você espera que a consulta retorne apenas um único resultado ou quando você quer tratar a exceção `NoResultException` caso nenhum resultado seja encontrado.

Aqui está a descrição do método `getSingleResult`:

java Copy code

```
<T> T getSingleResult() throws NoResultException, NonUniqueResultException;
```

Explicando os parâmetros e retorno:

- `<T>`: O tipo do objeto que você espera obter como resultado da consulta. Por exemplo, se você estiver consultando uma entidade chamada `Product`, o tipo `<T>` será `Product`.
- `getSingleResult()`: Este é o método que executa a consulta JPQL e retorna o único resultado correspondente. Se a consulta não encontrar nenhum resultado, ele lançará uma exceção `NoResultException`. Se a consulta retornar mais de um resultado, ele lançará uma exceção `NonUniqueResultException`.
- `throws NoResultException`: É uma exceção que é lançada quando a consulta não encontra nenhum resultado. Você deve tratá-la em um bloco `try-catch` caso esteja usando `getSingleResult`.
- `throws NonUniqueResultException`: É uma exceção que é lançada quando a consulta retorna mais de um resultado. Se você espera apenas um único resultado, é melhor usar `getSingleResult`, pois esta exceção indica que há um problema com a consulta ou o modelo de dados, como múltiplos registros com o mesmo valor único.

Classe ProdutoDao

```
// Método para buscar um único produto, filtrando por nome:
public Produto buscarUnicoPorNome(String nome)
                                        throws NoResultException {

    String jpql = "SELECT p FROM Produto p WHERE p.nome = :n";
    return em.createQuery(jpql, Produto.class)
             .setParameter("n", nome)
             .getSingleResult();
}
```

No main() ...

```
Celular DOIS
Samsung
2023-08-05
2000.00
```

≠

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    try {
        // Buscando o produto com nome "Celular DOIS":
        Produto p = produtoDao.buscarUnicoPorNome("Celular Dois");

        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());

    } catch (NoResultException e) {
        System.out.println("\n\nProduto não encontrado!");
    }
}
```

No main() ...

```
Celular DOIS
Samsung
2023-08-05
2000.00
```

≠

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    try {
        // Buscando o produto com nome "Celular DOIS":
        Produto p = produtoDao.buscarUnicoPorNome("Celular Dois");

        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());

    } catch (NoResultException e) {
        System.out.println("\n\nProduto não encontrado!");
    }
}
```

```
2024-02-16T13:16:47.700-03:00  INFO 16961 --- [
Hibernate: select p1_0.id,p1_0.categoria_id,p1_0



Produto não encontrado!
```

No main() ...

```
Celular DOIS
Samsung
2023-08-05
2000.00
```

```java
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    try {
        // Buscando o produto com nome "Celular DOIS":
        Produto p = produtoDao.buscarUnicoPorNome("Celular DOIS");

        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());

    } catch (NoResultException e) {
        System.out.println("\n\nProduto não encontrado!");
    }
}
```

No main() ...

```
Celular DOIS
Samsung
2023-08-05
2000.00
```

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    try {
        // Buscando o produto com nome "Celular DOIS":
        Produto p = produtoDao.buscarUnicoPorNome("Celular DOIS");

        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());

    } catch (NoR
        System.
    }
}
```

```
Hibernate: select p1_0.id,p1_0.categoria_id,p
Hibernate: select c1_0.id,c1_0.nome from cate
Celular DOIS
Samsung
2024-02-16
2000.00
```

Usando o relacionamento,

filtrando pela categoria.

“inner join”...

Relembrando inner join... (matéria de BD1)

quando você faz uma consulta usando um relacionamento entre tabelas em um banco de dados, está geralmente envolvendo a cláusula "JOIN". O "JOIN" é um dos principais comandos SQL utilizados para combinar informações de duas ou mais tabelas relacionadas em um único resultado, permitindo que você recupere dados de diferentes tabelas com base em suas chaves de relacionamento.

Vamos explicar os fundamentos do "JOIN" usando um exemplo simples com duas tabelas relacionadas:

Suponha que temos duas tabelas: "Clientes" e "Pedidos". A tabela "Clientes" armazena informações sobre os clientes, como ID do cliente, nome e endereço, enquanto a tabela "Pedidos" armazena informações sobre os pedidos feitos pelos clientes, como ID do pedido, data e ID do cliente associado ao pedido.

Tabela "Clientes":

ID	Nome	Endereço
1	João	Rua A, 123
2	Maria	Rua B, 456
3	José	Rua C, 789

Tabela "Pedidos":

ID	Data	ID_Cliente
101	2023-08-01	2
102	2023-08-02	1
103	2023-08-03	3

Aqui, a coluna "ID_Cliente" na tabela "Pedidos" é uma chave estrangeira que se relaciona com a coluna "ID" na tabela "Clientes". Isso indica que cada pedido está associado a um cliente específico.

Agora, suponha que você queira recuperar os detalhes de pedidos juntamente com os detalhes do cliente para cada pedido. É aqui que entra o "JOIN". Existem diferentes tipos de JOINs, mas vamos focar no INNER JOIN, que é o mais comum.

A sintaxe do INNER JOIN é a seguinte:

```
SELECT
    Pedidos.ID,
    Pedidos.Data,
    Clientes.Nome,
    Clientes.Endereço
FROM
    Pedidos
INNER JOIN
    Clientes
ON
    Pedidos.ID_Cliente = Clientes.ID;
```

O que está acontecendo aqui?

1. **`SELECT`**: Estamos selecionando as colunas que desejamos retornar no resultado da consulta. Neste caso, queremos o ID e a data do pedido, juntamente com o nome e o endereço do cliente.
2. **`FROM`**: Estamos definindo as tabelas das quais queremos recuperar os dados, que são as tabelas "Pedidos" e "Clientes".
3. **`INNER JOIN`**: Esta é a cláusula que indica que queremos combinar as informações de ambas as tabelas. Ela especifica qual coluna é usada para fazer o relacionamento entre as tabelas.
4. **`ON`**: Aqui, definimos a condição de junção. Estamos dizendo que a coluna "ID_Cliente" da tabela "Pedidos" deve ser igual à coluna "ID" da tabela "Clientes" para que os registros sejam combinados corretamente.

```sql
SELECT
    Pedidos.ID,
    Pedidos.Data,
    Clientes.Nome,
    Clientes.Endereço
FROM
    Pedidos
INNER JOIN
    Clientes
ON
    Pedidos.ID_Cliente = Clientes.ID;
```

select p.ID, p.Data, p.Nome, p.Endereço
from Pedidos p join Clientes c
on p.ID_cliente=c.ID; 16:31

OU 16:32

select p.ID, p.Data, p.Nome, p.Endereço
from Pedidos p, Clientes c
where p.ID_cliente=c.ID; 16:32

Dessa forma, a consulta retornará o seguinte resultado:

ID	Data	Nome	Endereço
101	2023-08-01	Maria	Rua B, 456
102	2023-08-02	João	Rua A, 123
103	2023-08-03	José	Rua C, 789

Observe que os dados dos pedidos foram combinados corretamente com os detalhes do cliente usando a coluna de relacionamento "ID_Cliente" e "ID". O "JOIN" nos permite obter uma visão completa dos dados correlacionados a partir de várias tabelas relacionadas no banco de dados.

Obrigado Silvana!!

Apaguei o banco...

Recriei com mais dados...

(retirei os comentários do código
da função cadastrarProdutos())

```java
private static void cadastrarProdutos() {

    // Criando a Categoria "CELULARES":
    Categoria celulares = new Categoria("CELULARES");
    // Criando a Categoria "INFORMATICA":
    Categoria informatica = new Categoria("INFORMATICA");

    // Criando objetos da Classe Produto, associados a categoria CELULARES:
    Produto celular1 = new Produto("Celular UM", "Motorola",
            new BigDecimal("1000"), celulares);
    Produto celular2 = new Produto("Celular DOIS", "Samsung",
            new BigDecimal("2000"), celulares);

    // Criando objetos da Classe Produto, associados a categoria INFORMATICA:
    Produto comp1 = new Produto("Deskjet 3776", "HP",
            new BigDecimal("700"), informatica);
    Produto comp2 = new Produto("Notebook i7", "DELL",
            new BigDecimal("9000"), informatica);

    // Criando o EntityManager:
    EntityManager em = JPAUtil.getEntityManager();

    // Criando os DAOs:
    ProdutoDao produtoDao = new ProdutoDao(em);
    CategoriaDao categoriaDao = new CategoriaDao(em);

    // Iniciando a transação:
    em.getTransaction().begin();
    // Persistindo os objetos:
    categoriaDao.cadastrar(celulares);
    categoriaDao.cadastrar(informatica);
    produtoDao.cadastrar(celular1);
    produtoDao.cadastrar(celular2);
    produtoDao.cadastrar(comp1);
    produtoDao.cadastrar(comp2);
    // Finalizando (commit) a transação:
    em.getTransaction().commit();
    // Fechando o EntityManager:
    em.close();
}
```

Estado do banco, para testar:

SELECT * FROM CATEGORIAS;

ID	NOME
1	CELULARES
2	INFORMATICA

(2 rows, 1 ms)

SELECT * FROM PRODUTOS;

ID	DATACADASTRO	DESCRICAO	NOME	PRECO	CATEGORIA_ID
1	2025-08-08	Motorola	Celular UM	1000.00	1
2	2025-08-08	Samsung	Celular DOIS	2000.00	1
3	2025-08-08	HP	Deskjet 3776	700.00	2
4	2025-08-08	DELL	Notebook i7	9000.00	2

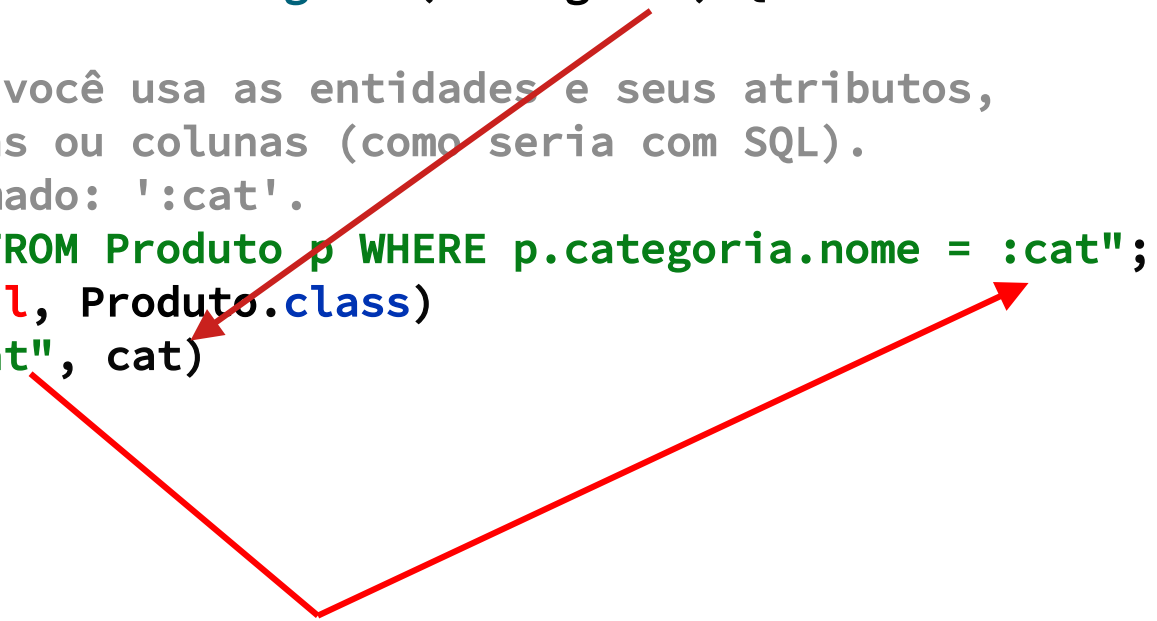
(4 rows, 1 ms)

Classe ProdutoDao

```
// Método para buscar todos os produtos, filtrando pela categoria,
// ou seja, pelo relacionamento:
public List<Produto> buscarPorNomeDaCategoria(String cat) {

    // Lembrando que na JPQL você usa as entidades e seus atributos,
    // e não o nome de tabelas ou colunas (como seria com SQL).
    // Parâmetro a ser informado: ':cat'.
    String jpql = "SELECT p FROM Produto p WHERE p.categoria.nome = :cat";
    return em.createQuery(jpql, Produto.class)
            .setParameter("cat", cat)
            .getResultList();
}
```

Classe ProdutoDao

```java
// Método para buscar todos os produtos, filtrando pela categoria,
// ou seja, pelo relacionamento:
public List<Produto> buscarPorNomeDaCategoria(String cat) {

    // Lembrando que na JPQL você usa as entidades e seus atributos,
    // e não o nome de tabelas ou colunas (como seria com SQL).
    // Parâmetro a ser informado: ':cat'.
    String jpql = "SELECT p FROM Produto p WHERE p.categoria.nome = :cat";
    return em.createQuery(jpql, Produto.class)
             .setParameter("cat", cat)
             .getResultList();
}
```



Classe ProdutoDao

```java
// Método para buscar todos os produtos, filtrando pela categoria,
// ou seja, pelo relacionamento:
public List<Produto> buscarPorNomeDaCategoria(String cat) {

    // Lembrando que na JPQL você usa as entidades e seus atributos,
    // e não o nome de tabelas ou colunas (como seria com SQL).
    // Parâmetro a ser informado: ':cat'.
    String jpql = "SELECT p FROM Produto p WHERE p.categoria.nome = :cat";
    return em.createQuery(jpql, Produto.class)
             .setParameter("cat", cat)
             .getResultList();
}
```

```java
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();
    // Precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;
```

Classe ProdutoDao

```java
// Método para buscar todos os produtos, filtrando pela categoria,
// ou seja, pelo relacionamento:
public List<Produto> buscarPorNomeDaCategoria(String cat) {

    // Lembrando que na JPQL você usa as entidades e seus atributos,
    // e não o nome de tabelas ou colunas (como seria com SQL).
    // Parâmetro a ser informado: ':cat'.
    String jpql = "SELECT p FROM Produto p WHERE p.categoria.nome = :cat";
    return em.createQuery(jpql, Produto.class)
             .setParameter("cat", cat)
             .getResultList();
}
```

```java
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();
    // Precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;
```

Classe ProdutoDao

```java
// Método para buscar todos os produtos, filtrando pela categoria,
// ou seja, pelo relacionamento:
public List<Produto> buscarPorNomeDaCategoria(String cat) {

    // Lembrando que na JPQL você usa as entidades e seus atributos,
    // e não o nome de tabelas ou colunas (como seria com SQL).
    // Parâmetro a ser informado: ':cat'.
    String jpql = "SELECT p FROM Produto p WHERE p.categoria.nome = :cat";
    return em.createQuery(jpql, Produto.class)
             .setParameter("cat", cat)
             .getResultList();
}
```

```java
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();
    // Precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;
```

```java
@Entity
@Table(name = "categorias")
public class Categoria {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
```

No main()

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    List<Produto> todos = produtoDao.buscarPorNomeDaCategoria("INFORMATICA");

    for(Produto p : todos) {
        System.out.println("-----------------------");
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }

}
```

No main()

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager
    ProdutoDao produtoDao = new ProdutoDao(em);

    List<Produto> todos = produtoDao.buscarPorN

    for(Produto p : todos) {
        System.out.println("------------------
        System.out.println(p.getNome());
        System.out.println(p.getDescricao());
        System.out.println(p.getDataCadastro());
        System.out.println(p.getPreco());
    }

}
```

```
Hibernate: select p1_0.id,p1_0.categ
Hibernate: select c1_0.id,c1_0.nome
----------------------
Deskjet 3776
HP
2025-02-20
700.00
----------------------
Notebook i7
DELL
2025-02-20
9000.00

Process finished with exit code 0
```

SELECT * FROM PRODUTOS;

ID	DATACADASTRO	DESCRICAO	NOME	PRECO	CATEGORIA_ID
1	2025-08-08	Motorola	Celular UM	1000.00	1
2	2025-08-08	Samsung	Celular DOIS	2000.00	1
3	2025-08-08	HP	Deskjet 3776	700.00	2
4	2025-08-08	DELL	Notebook i7	9000.00	2

(4 rows, 1 ms)

Como já vimos, o log mostra o comando sql gerado:

```
ago. 05, 2023 5:06:45 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@49e2b3c5] for (n
Hibernate: select p1_0.id,p1_0.categoria_id,p1_0.dataCadastro,p1_0.descricao,p1_0.nome,p1_0.preco from produtos p1_0 join categorias c1_0 on c1_0.id=p1_0.categoria_id where c1_0.nome=?
Hibernate: select c1_0.id,c1_0.nome from categorias c1_0 where c1_0.id=?
-----------------------
Deskjet 3776
HP
2023-08-05
700.00
-----------------------
Notebook i7
DELL
2023-08-05
9000.00

Process finished with exit code 0
```

Como já vimos, o log mostra o comando sql gerado:

```
ago. 05, 2023 5:06:45 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@49e2b3c5] for (n
Hibernate: select p1_0.id,p1_0.categoria_id,p1_0.dataCadastro,p1_0.descricao,p1_0.nome,p1_0.preco from produtos p1_0 join categorias c1_0 on c1_0.id=p1_0.categoria_id where c1_0.nome=?
Hibernate: select c1_0.id,c1_0.nome from categorias c1_0 where c1_0.id=?
-----------------------
Deskjet 3776
HP
2023-08-05
700.00
-----------------------
Notebook i7
DELL
2023-08-05
9000.00

Process finished with exit code 0
```

Linha muito grande!!!!
De difícil leitura.

Podemos configurar o Hibernate pra formatar melhor isso....

Lá no persistence.xml...

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

Executando novamente...

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

```
2024-02-16T13:48:37.263-03:00  INFO 17813 --- [           main] org.hibernate.orm.
Hibernate:
    select
        p1_0.id,
        p1_0.categoria_id,
        p1_0.dataCadastro,
        p1_0.descricao,
        p1_0.nome,
        p1_0.preco
    from
        produtos p1_0
    join
        categorias c1_0
            on c1_0.id=p1_0.categoria_id
    where
        c1_0.nome=?
Hibernate:
    select
        c1_0.id,
        c1_0.nome
    from
        categorias c1_0
    where
        c1_0.id=?
----------------------
Deskjet 3776
HP
2024-02-16
700.00
----------------------
```

Aproveitando...
pra desligar totalmente as mensagens do Hibernate:

```
// Para desligar totalmente as mensagens do Hibernate:
java.util.logging.Logger.getLogger("org.hibernate").setLevel(Level.OFF);
```

CadastroDeProduto

```
/usr/lib/jvm/jdk-21-oracle-x64/bin/java ..
Hibernate:
    select
        p1_0.id,
        p1_0.categoria_id,
        p1_0.dataCadastro,
        p1_0.descricao,
        p1_0.nome,
        p1_0.preco
    from
        produtos p1_0
    join
        categorias c1_0
            on c1_0.id=p1_0.categoria_id
    where
        c1_0.nome=?
Hibernate:
    select
        c1_0.id,
        c1_0.nome
    from
        categorias c1_0
    where
        c1_0.id=?
-----------------------
Deskjet 3776
HP
2024-08-05
700.00
-----------------------
Notebook i7
DELL
2024-08-05
9000.00
```

Retornando apenas **um atributo** de um objeto.

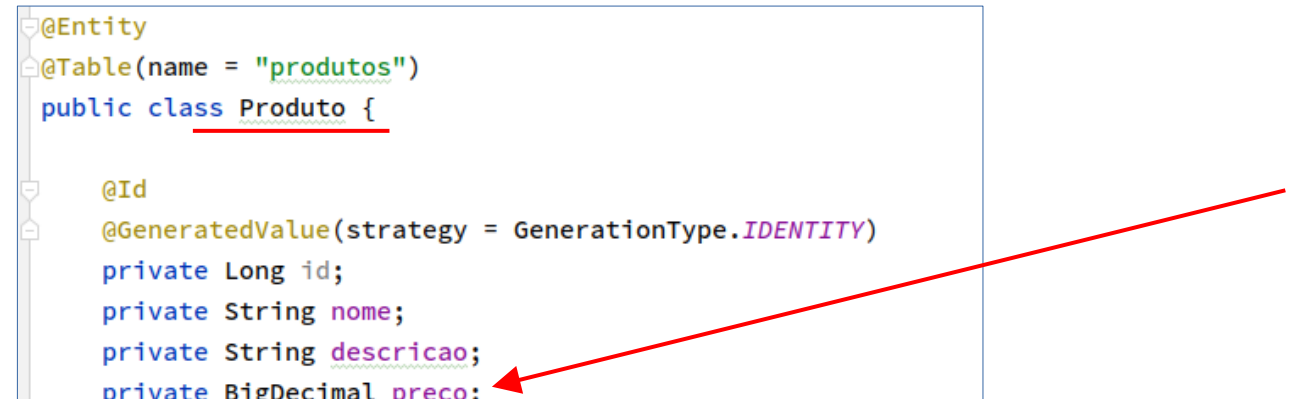
Classe ProdutoDao

```
// Método para buscar o preço de um produto, filtrando por nome:
public BigDecimal buscarPrecoDoProdutoComNome(String nome) {

    String jpql = "SELECT p.preco FROM Produto p WHERE p.nome = :n";
    return em.createQuery(jpql, BigDecimal.class)
             .setParameter("n", nome)
             .getSingleResult();
}
```

Classe ProdutoDao

```java
// Método para buscar o preço de um produto, filtrando por nome:
public BigDecimal buscarPrecoDoProdutoComNome(String nome) {

    String jpql = "SELECT p.preco FROM Produto p WHERE p.nome = :n";
    return em.createQuery(jpql, BigDecimal.class)
             .setParameter("n", nome)
             .getSingleResult();
}
```

```java
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();
    // Precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;
```



Classe ProdutoDao

```java
// Método para buscar o preço de um produto, filtrando por nome:
public BigDecimal buscarPrecoDoProdutoComNome(String nome) {

    String jpql = "SELECT p.preco FROM Produto p WHERE p.nome = :n";
    return em.createQuery(jpql, BigDecimal.class)
             .setParameter("n", nome)
             .getSingleResult();
}
```

```java
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();
    // Precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;
```

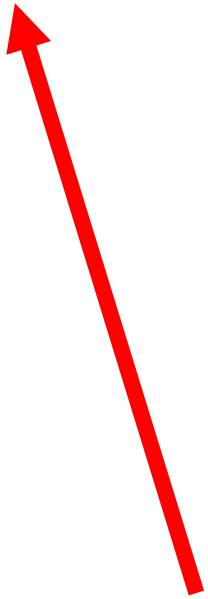
Classe ProdutoDao

```java
// Método para buscar o preço de um produto, filtrando por nome:
public BigDecimal buscarPrecoDoProdutoComNome(String nome) {

    String jpql = "SELECT p.preco FROM Produto p WHERE p.nome = :n";
    return em.createQuery(jpql, BigDecimal.class)
            .setParameter("n", nome)
            .getSingleResult();
}
```

```java
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();
    // Precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;
```

Classe ProdutoDao

```
// Método para buscar o preço de um produto, filtrando por nome:
public BigDecimal buscarPrecoDoProdutoComNome(String nome) {

    String jpql = "SELECT p.preco FROM Produto p WHERE p.nome = :n";
    return em.createQuery(jpql, BigDecimal.class)
             .setParameter("n", nome)
             .getSingleResult();
}
```



No main()...

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    BigDecimal precoDeskjet = produtoDao.buscarPrecoDoProdutoComNome("Deskjet 3776");

    System.out.println("\nPreço da Deskjet: " + precoDeskjet);

}
```

No main()...

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    BigDecimal precoDeskjet = produtoDao.buscarPrecoDoProdutoComNome("Deskjet 3776");

    System.out.println("\nPreço da Deskjet: " + precoDeskjet);

}
```

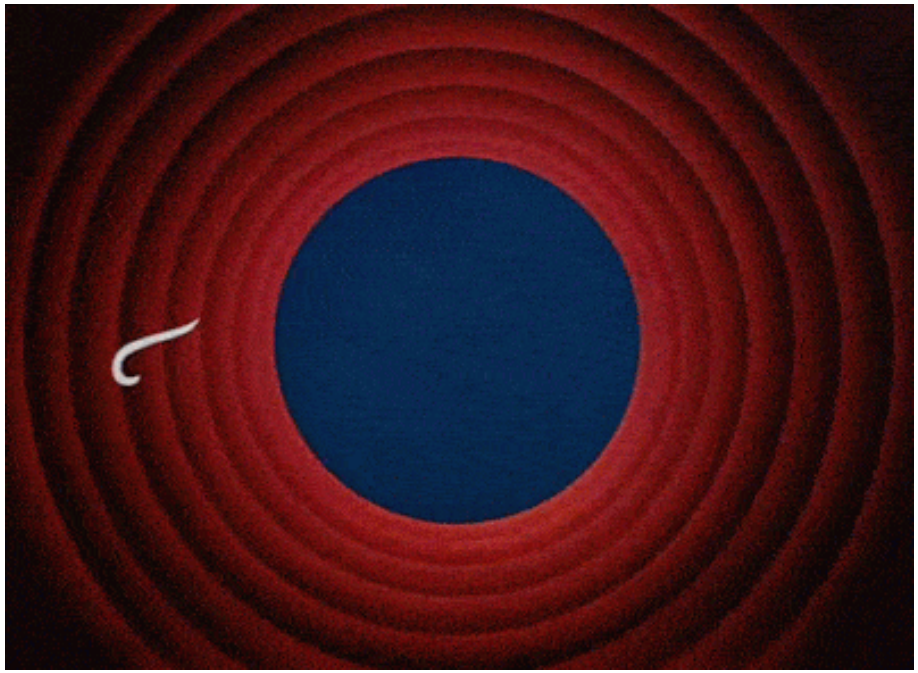
```
2024-02-16T13:52:18.071-03:00  INFO 18010 ---
2024-02-16T13:52:18.081-03:00  INFO 18010 ---
Hibernate:
    select
        p1_0.preco
    from
        produtos p1_0
    where
        p1_0.nome=?

Preço da Deskjet: 700.00
```

No main()...

```
public static void main(String[] args) {

    //cadastrarProdutos();

    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao produtoDao = new ProdutoDao(em);

    BigDecimal precoDeskjet = produtoDao.buscarPrecoDoProdutoComNome("Deskjet 3776");

    System.out.println("\nPreço da Deskjet: " + precoDeskjet);

}
```

Não fizemos neste exemplo, mas obviamente deveriamos ter tratado o caso da consulta não encontrar nenhum elemento, como vimos anteriormente.

```
2024-02-16T13:52:18.071-03:00  INFO 18010 ---
2024-02-16T13:52:18.081-03:00  INFO 18010 ---
Hibernate:
    select
        p1_0.preco
    from
        produtos p1_0
    where
        p1_0.nome=?

Preço da Deskjet: 700.00
```



Teriamos muita coisa ainda pra falar de JPA...

Mas esse não é o objetivo principal da disciplina.

Mas já dá pra fazer muita coisa boa...
e principalmente, fundamentar novos estudos.

Lembrando que muitos conceitos apresentados
aqui serão utilizados no Spring Data.

Estado final do projeto disponível no Github.

Exercícios


Social Network
Public Cloud
Laptop
Mobile
Tablet
PC
APPs
entertainment
Telemedicine
Infrastructure
Docs
Training
Storage
Backup
Enterprise
Cloud
Private
internal
Cloud
depositphotos

Exercícios

- Reproduza e experimente com os códigos vistos nesta aula.

  ou...

- Já comece a trabalhar na Avaliação #1 !! 

Avaliações

- **Avaliação 1**
  - Prática - Conteúdo: JPA.
  - Vale **15%** da média.
- **Avaliação 2**
  - Escrita - Conteúdo: parte teórica (api rest, reqs. http, arquit. spring).
  - Vale **10%** da média.
- **Avaliação 3**
  - Prática - Conteúdo: API Rest, CRUD, com Spring.
  - A nota final vale **30%** da média da disciplina.
- **Avaliação 4**
  - Prática - Conteúdo: dotar a api criada na **av.3** de autenticação (com Spring Security).
  - Vale **45%** da média.

Avaliação #1

- Faça um programa em Java, usando JPA para persistência de dados, que apresente o seguinte menu de opções:

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM
```

Classe Aluno (obrigatório)

```
public class Aluno {

    private Long id;
    private String nome;
    private String ra;
    private String email;
    private BigDecimal nota1;
    private BigDecimal nota2;
    private BigDecimal nota3;

    ...
```

Relembrando a leitura via teclado...

```
import java.util.Scanner;


    Scanner leitorTeclado = new Scanner(System.in);

    float  numF = leitorTeclado.nextFloat();
    int    numI = leitorTeclado.nextInt();
    byte   numB = leitorTeclado.nextByte();
    long   numL = leitorTeclado.nextLong();
    boolean  vb = leitorTeclado.nextBoolean();
    double numD = leitorTeclado.nextDouble();
    String  str = leitorTeclado.nextLine();


    leitorTeclado.close()
```

Exemplo de execução

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 1

CADASTRO DE ALUNO:
Digite o nome: Asdrubal
Digite o RA: RA111
Digite o email: asd@gmail.com
Digite a nota 1: 7
Digite a nota 2: 8
Digite a nota 3: 8
```

```
mysql> select * from alunos;
+----+---------------+----------+-------+-------+-------+-------+
| id | email         | nome     | nota1 | nota2 | nota3 | ra    |
+----+---------------+----------+-------+-------+-------+-------+
|  1 | asd@gmail.com | Asdrubal |  7.00 |  8.00 |  8.00 | RA111 |
+----+---------------+----------+-------+-------+-------+-------+
```

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 1

CADASTRO DE ALUNO:
Digite o nome: Lupita
Digite o RA: RA222
Digite o email: lu@gmail.com
Digite a nota 1: 3
Digite a nota 2: 4
Digite a nota 3: 3
```

```
mysql> select * from alunos;
+----+---------------+----------+-------+-------+-------+-------+
| id | email         | nome     | nota1 | nota2 | nota3 | ra    |
+----+---------------+----------+-------+-------+-------+-------+
|  1 | asd@gmail.com | Asdrubal |  7.00 |  8.00 |  8.00 | RA111 |
|  2 | lu@gmail.com  | Lupita   |  3.00 |  4.00 |  3.00 | RA222 |
+----+---------------+----------+-------+-------+-------+-------+
```

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 2

EXCLUIR ALUNO:
Digite o nome: Asdrubal

Aluno removido com sucesso!
```

```
mysql> select * from alunos;
+----+--------------+--------+-------+-------+-------+-------+
| id | email        | nome   | nota1 | nota2 | nota3 | ra    |
+----+--------------+--------+-------+-------+-------+-------+
|  2 | lu@gmail.com | Lupita |  3.00 |  4.00 |  3.00 | RA222 |
+----+--------------+--------+-------+-------+-------+-------+
```

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 2

EXCLUIR ALUNO:
Digite o nome: Bianca

Aluno não encontrado!
```

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 3

ALTERAR ALUNO:
Digite o nome: Lupita
Dados do aluno:
Nome: Lupita
Email: lu@gmail.com
RA: RA222
Notas: 3.00 - 4.00 - 3.00

NOVOS DADOS:
Digite o nome: Lupita Pereira
Digite o RA: RA222X
Digite o email: lupita@gmail.com
Digite a nota 1: 4
Digite a nota 2: 5
Digite a nota 3: 4

Aluno Alterado com sucesso!
```

```
mysql> select * from alunos;
+----+--------------+--------+-------+-------+-------+-------+
| id | email        | nome   | nota1 | nota2 | nota3 | ra    |
+----+--------------+--------+-------+-------+-------+-------+
|  2 | lu@gmail.com | Lupita |  3.00 |  4.00 |  3.00 | RA222 |
+----+--------------+--------+-------+-------+-------+-------+
```

```
      select * from alunos;
+----+-----------------+---------------+-------+-------+-------+--------+
| id | email           | nome          | nota1 | nota2 | nota3 | ra     |
+----+-----------------+---------------+-------+-------+-------+--------+
|  2 | lupita@gmail.com | Lupita Pereira |  4.00 |  5.00 |  4.00 | RA222X |
+----+-----------------+---------------+-------+-------+-------+--------+
```

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 3

ALTERAR ALUNO:
Digite o nome: Bianca

Aluno não encontrado!
```

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 4

CONSULTAR ALUNO:
Digite o nome: Lupita Pereira
Dados do aluno:
Nome: Lupita Pereira
Email: lupita@gmail.com
RA: RA222X
Notas: 4 - 5 - 4
```

```
select * from alunos;
+----+-----------------+---------------+-------+-------+-------+--------+
| id | email           | nome          | nota1 | nota2 | nota3 | ra     |
+----+-----------------+---------------+-------+-------+-------+--------+
|  2 | lupita@gmail.com | Lupita Pereira |  4.00 |  5.00 |  4.00 | RA222X |
+----+-----------------+---------------+-------+-------+-------+--------+
```

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 4

CONSULTAR ALUNO:
Digite o nome: Bianca

Aluno não encontrado!
```

```
mysql> select * from alunos;
+----+------------------+----------------+-------+-------+-------+---------------+
| id | email            | nome           | nota1 | nota2 | nota3 | ra            |
+----+------------------+----------------+-------+-------+-------+---------------+
|  2 | lupita@gmail.com | Lupita Pereira |  4.00 |  5.00 |  4.00 | RA222X        |
|  3 | asd@gmail.com    | Asdrubal       |  7.00 |  8.00 |  9.00 | RA666         |
|  4 | RAZZZZZZZ        | Zoroastro      |  2.00 |  3.00 |  2.00 | zora@gmail.com |
+----+------------------+----------------+-------+-------+-------+---------------+
```

```
** CADASTRO DE ALUNOS **

1 - Cadastrar aluno
2 - Excluir aluno
3 - Alterar aluno
4 - Buscar aluno pelo nome
5 - Listar alunos (com status aprovação)
6 - FIM

Digite a opção desejada: 5
```

```
Exibindo todos os alunos:

Nome: Lupita Pereira
Email: lupita@gmail.com
RA: RA222X
Notas: 4.00 - 5.00 - 4.00
Media: 4.33
Situação: Recuperação

Nome: Asdrubal
Email: asd@gmail.com
RA: RA666
Notas: 7 - 8 - 9
Media: 8
Situação: Aprovado

Nome: Zoroastro
Email: RAZZZZZZZ
RA: zora@gmail.com
Notas: 2 - 3 - 2
Media: 2
Situação: Reprovado
```

- abaixo de 4 : reprovado
- 4 ou acima, abaixo de 6 : recuperação
- 6 ou acima : aprovado

Entrega

- Prazo:
  - Até as 23:55 do dia 20/agosto (próxima aula 4a feira).
    - Não será feito controle de presença neste dia.
    - **Porém a presença na aula está vinculada a uma entrega válida do trabalho.**
- Forma de entrega:
  - Envio de email para **`discprw3ifspcarlao@gmail.com`**
    - Link do github, **obrigatoriamente!**
  - Se feito em dupla, colocar os nomes dos integrantes no email.
  - Colocar também como comentário no início do **main()**.
- Vale nota e presença nas 4 aulas deste dia (mínimo nota 8.0)

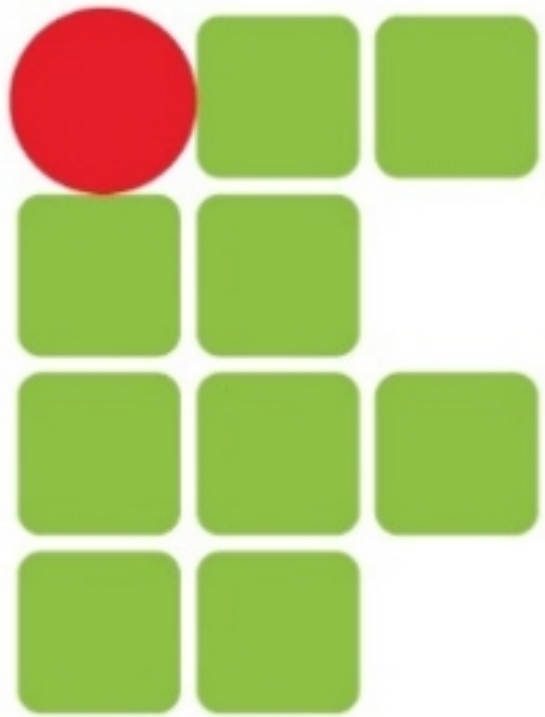
ATENÇÃO

Os projetos devem **“rodar”** !!
Projetos com erros que impeçam a execução,
não terão seus códigos avaliados,
e receberão nota **ZERO**.

AVISO!! (será repetido todas as aulas)

- Mantenha a disposição de cabos nos computadores dos labs.
- Se retirar um cabo de rede (para usar notebook),
  não esqueça de repor no computador ao final da aula.
- Mantenha o filtro de linha e as tomadas, atrás da mesa,
  em seu estado original, especialmente para que não possa ser esbarrado por outros alunos.
- Não altere de jeito nenhum os cabos atrás dos computadores,
  especialmente mexendo nas “travas”.
- **Desliguem a(s) máquinas antes de sair.**
- Lembre-se! Os laboratórios são nossos!! Vamos cuidar!


obrigado
pela atenção!
RENAN PAVINI



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos