

# **Disciplina: Desenvolvimento de Software**

## **Aula 10: Sistema Web Completo**

## Apresentação

Construir um sistema completo para Web, dentro de uma arquitetura MVC e com a utilização dos padrões corretos, envolve um bom planejamento e o uso de ferramentas para ganho de produtividade, além de conhecimento acerca de frameworks e tecnologias úteis para as diversas necessidades corporativas.

Uma boa estratégia para o desenvolvimento de sistemas desse tipo, na plataforma Java, é a utilização de JPA na camada Model, EJB na camada Control e JSF na camada View.

Promove-se, assim, uma organização natural, tirando proveito de várias tecnologias Java, como o uso de JTA para o controle transacional. O NetBeans permite automatizar boa parte da implementação, trazendo grande ganho de produtividade.

---

## Objetivos

- Usar o mapeamento objeto-relacional com JPA;
- Demonstrar componentes EJB como Facade;
- Desenvolver um sistema MVC completo com EJB, JPA e JSF.

# Configurando o aplicativo

---

Com o uso dos recursos do NetBeans, será muito fácil criar um sistema Java Web no padrão arquitetural MVC, utilizando o Front Control, pois boa parte das tarefas será automatizada pelo ambiente.

Inicialmente, vamos criar um aplicativo corporativo, o qual chamaremos de **ExemploCompleto** (lembrando que deveremos escolher **Java EE..Aplicação Enterprise** como tipo de projeto).

Desejamos utilizar o framework JSF nesse projeto, o que faz com que seja necessário fugir um pouco da criação padronizada de aplicativos corporativos oferecida pelo NetBeans.

Nesse novo projeto, selecionaremos apenas a criação do módulo EJB, ou seja, devemos **desmarcar** a opção “Criar Módulo de Aplicação Web”.

Continuaremos a utilizar o servidor GlassFish e o Java EE 7.

Agora temos o projeto principal e o projeto EJB criados, faltando apenas o módulo Web.

Criaremos um novo projeto **Java Web..Aplicação Web**, e seguiremos os seguintes passos:

1. Definir o nome como **ExemploCompletoW** e clicar em **Próximo**.

2. Associar à aplicação corporativa **ExemploCompleto**, com uso de GlassFish e Java EE 7, e clicar em **Próximo**.

Selecionar o framework **Java Server Faces** e clicar em **Finalizar**.

Agora já temos todos os projetos de que precisaremos, mas, como a parte Web foi criada de forma externa àquela adotada pelo processo padrão de criação, será necessário efetuar uma pequena adequação.

Da forma como está, o projeto Web não consegue utilizar as classes de EJB e JPA que criaremos no projeto EJB, pois não foi feita a associação entre as classes.

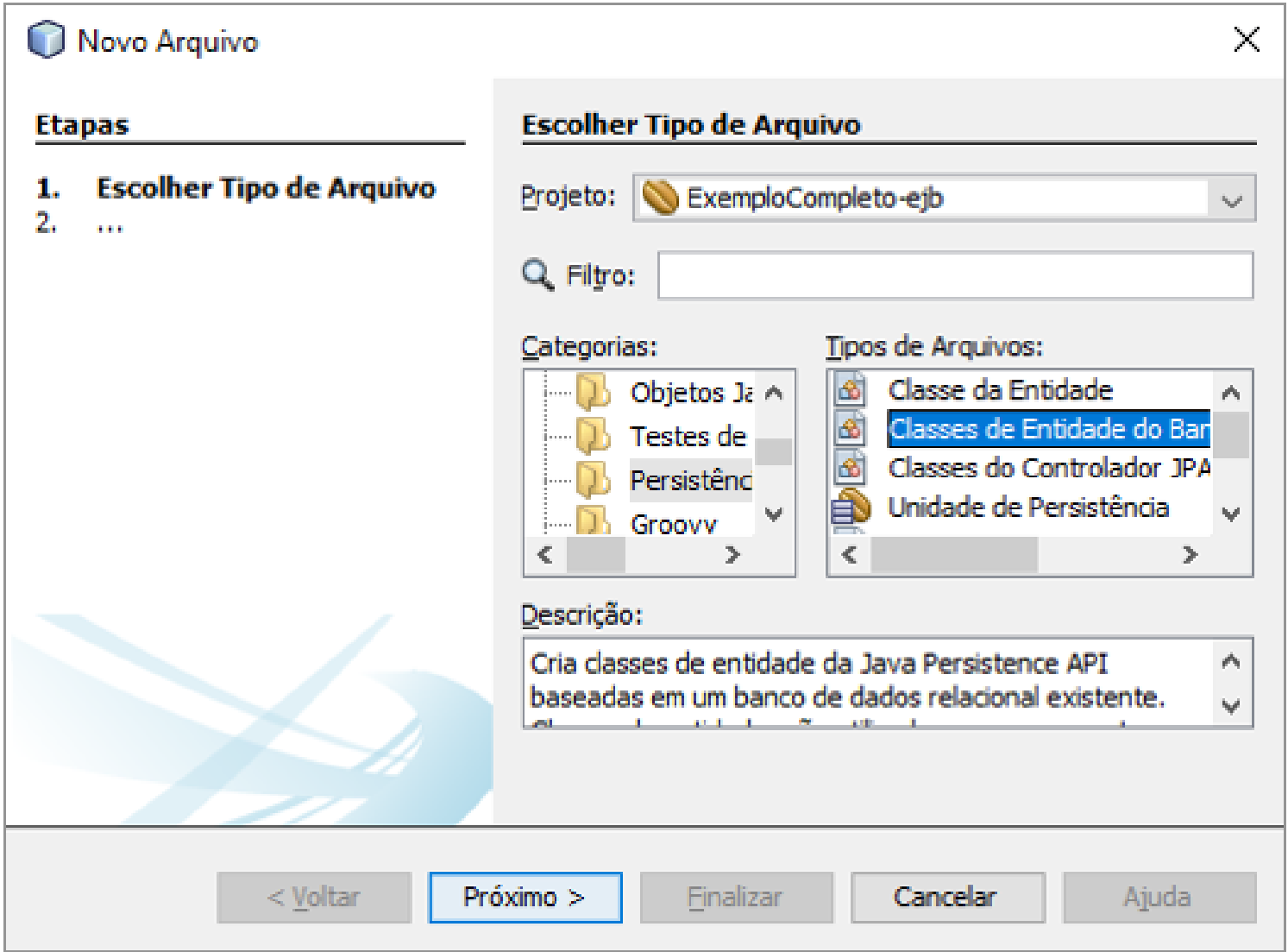
Devemos então clicar com o botão direito na divisão **Bibliotecas** do projeto **ExemploCompletoW** (Web), escolher Adicionar Projeto e, na janela que se abrirá, associar ao projeto **ExemploCompleto-ejb**, como podemos observar na figura seguinte.

Agora nossos projetos estão completamente configurados e podemos iniciar a implementação de nosso sistema.

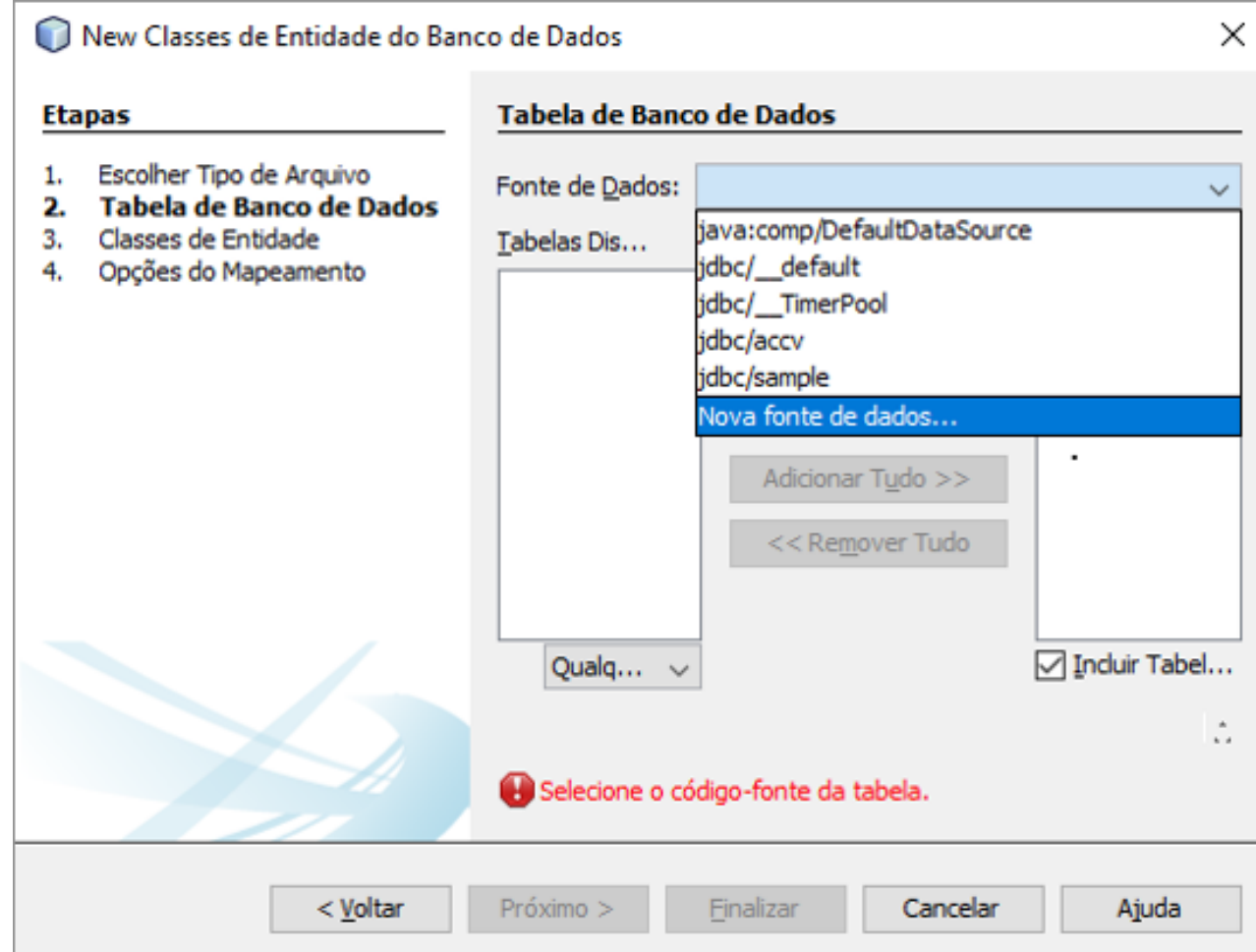
# Criando a camada Model

Após a criação do aplicativo corporativo e do projeto Web associado, vamos iniciar a implementação da camada **Model**, selecionando o projeto secundário **ExemploCompleto-ejb** e adicionando ao mesmo as entidades de forma automatizada.

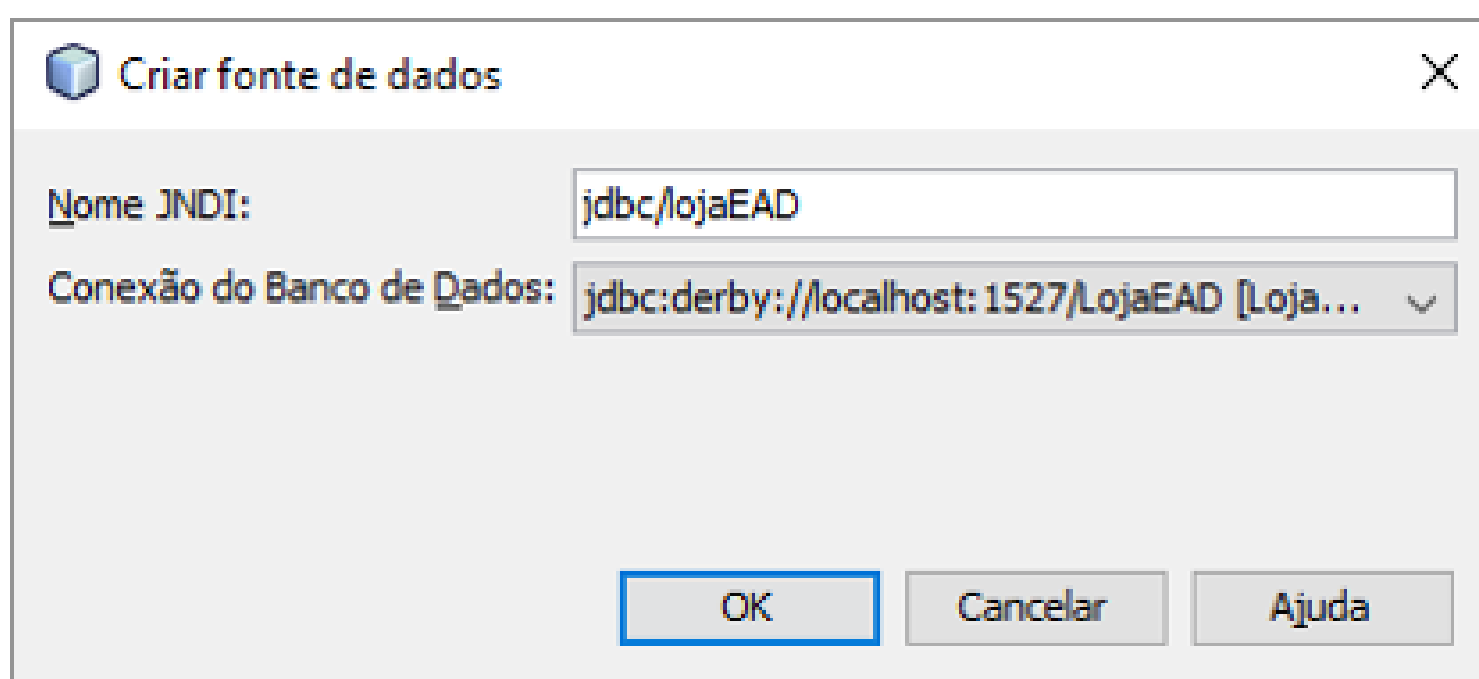
Para adicionar as entidades, devemos selecionar o menu **Arquivo..Novo Arquivo** e executar os seguintes passos:



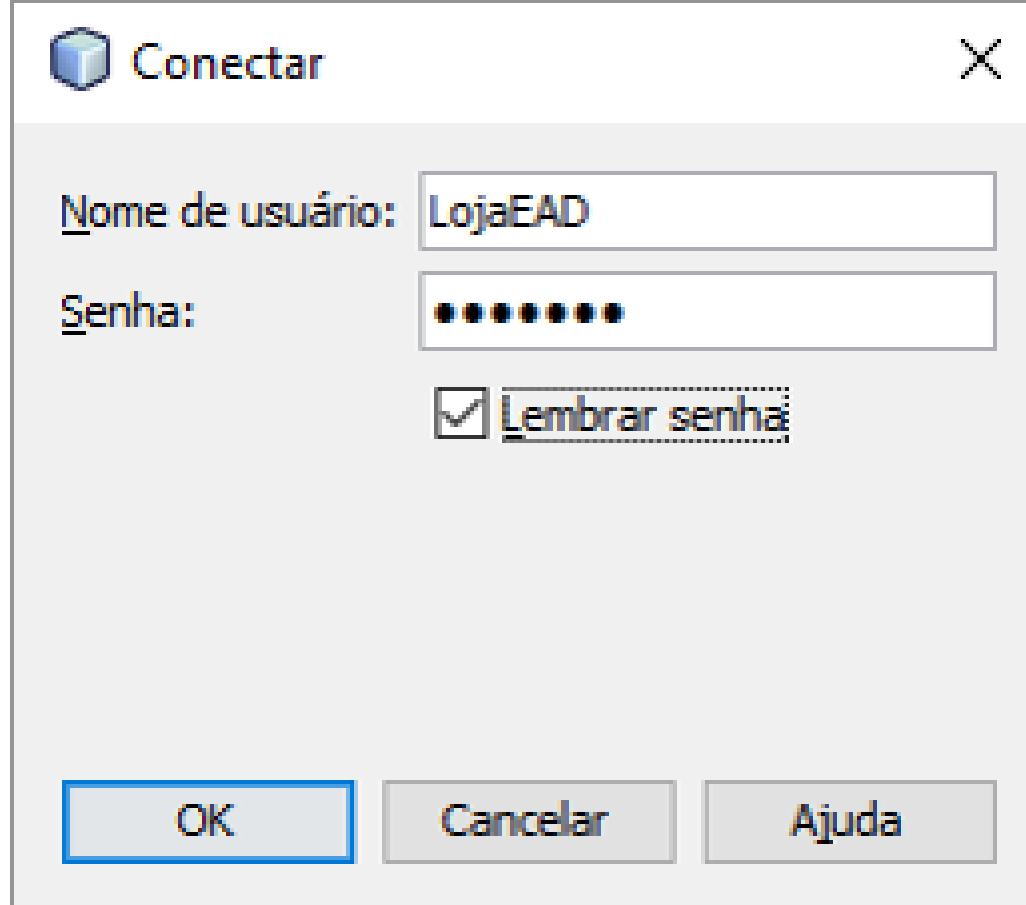
1. Selecionar o tipo de componente **Persistência..Classes de Entidade do Banco de Dados**, e clicar em **Próximo**.



Adicionar um **Nova Fonte de Dados**.



Configurar a nova fonte com o nome JNDI **jdbc/lojaEAD**, apontando para a conexão com nosso banco de exemplo.

A dialog box titled "Conectar" with a close button (X) in the top right corner. It contains two text input fields: "Nome de usuário:" with the value "LojaEAD" and "Senha:" with masked characters (dots). Below the password field is a checkbox labeled "Lembrar senha" which is checked. At the bottom are three buttons: "OK", "Cancelar", and "Ajuda".

Conectar

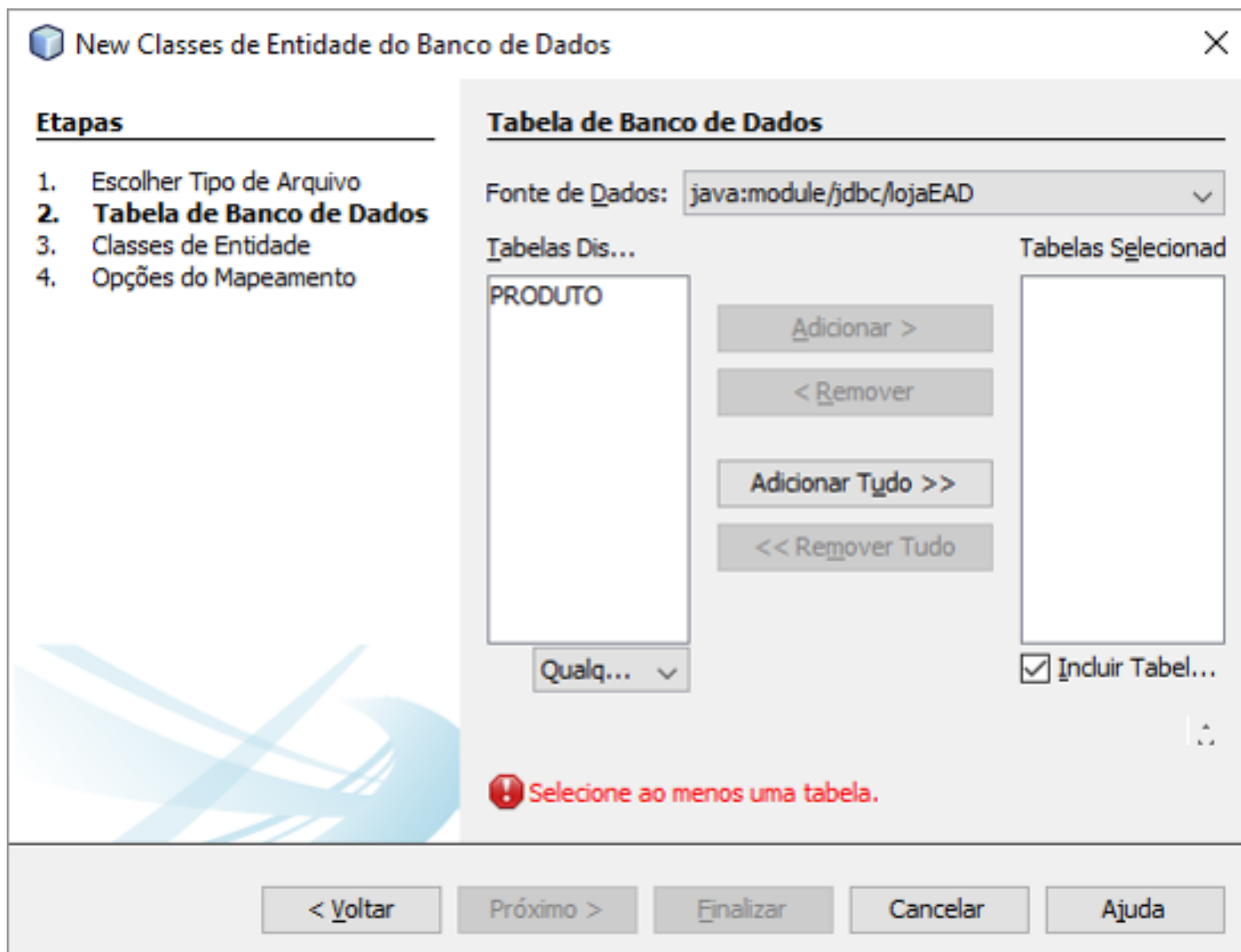
Nome de usuário: LojaEAD

Senha: ●●●●●●●●

☒ Lembrar senha

OK Cancelar Ajuda

Digitar a senha do banco e escolher **Lembrar Senha**.

A dialog box titled "New Classes de Entidade do Banco de Dados" with a close button (X) in the top right corner. It has a left sidebar with "Etapas" (Steps) numbered 1 to 4, where step 2, "Tabela de Banco de Dados", is selected. The main area is titled "Tabela de Banco de Dados" and contains a "Fonte de Dados:" dropdown set to "java:module/jdbc/lojaEAD". Below this are two list boxes: "Tabelas Dis..." (containing "PRODUTO") and "Tabelas Selecionadas" (empty). Between them are buttons: "Adicionar >", "< Remover", "Adicionar Tudo >>", and "<< Remover Tudo". At the bottom left of the main area is a "Qualq..." dropdown. At the bottom right is a checkbox "Incluir Tabel..." which is checked. A red error message "Selecione ao menos uma tabela." is displayed at the bottom of the main area. The footer contains five buttons: "< Voltar", "Próximo >", "Finalizar", "Cancelar", and "Ajuda".

New Classes de Entidade do Banco de Dados

**Etapas**

- Escolher Tipo de Arquivo
- Tabela de Banco de Dados**
- Classes de Entidade
- Opções do Mapeamento

**Tabela de Banco de Dados**

Fonte de Dados: java:module/jdbc/lojaEAD

Tabelas Dis... Tabelas Selecionadas

PRODUTO

Adicionar >

< Remover

Adicionar Tudo >>

<< Remover Tudo

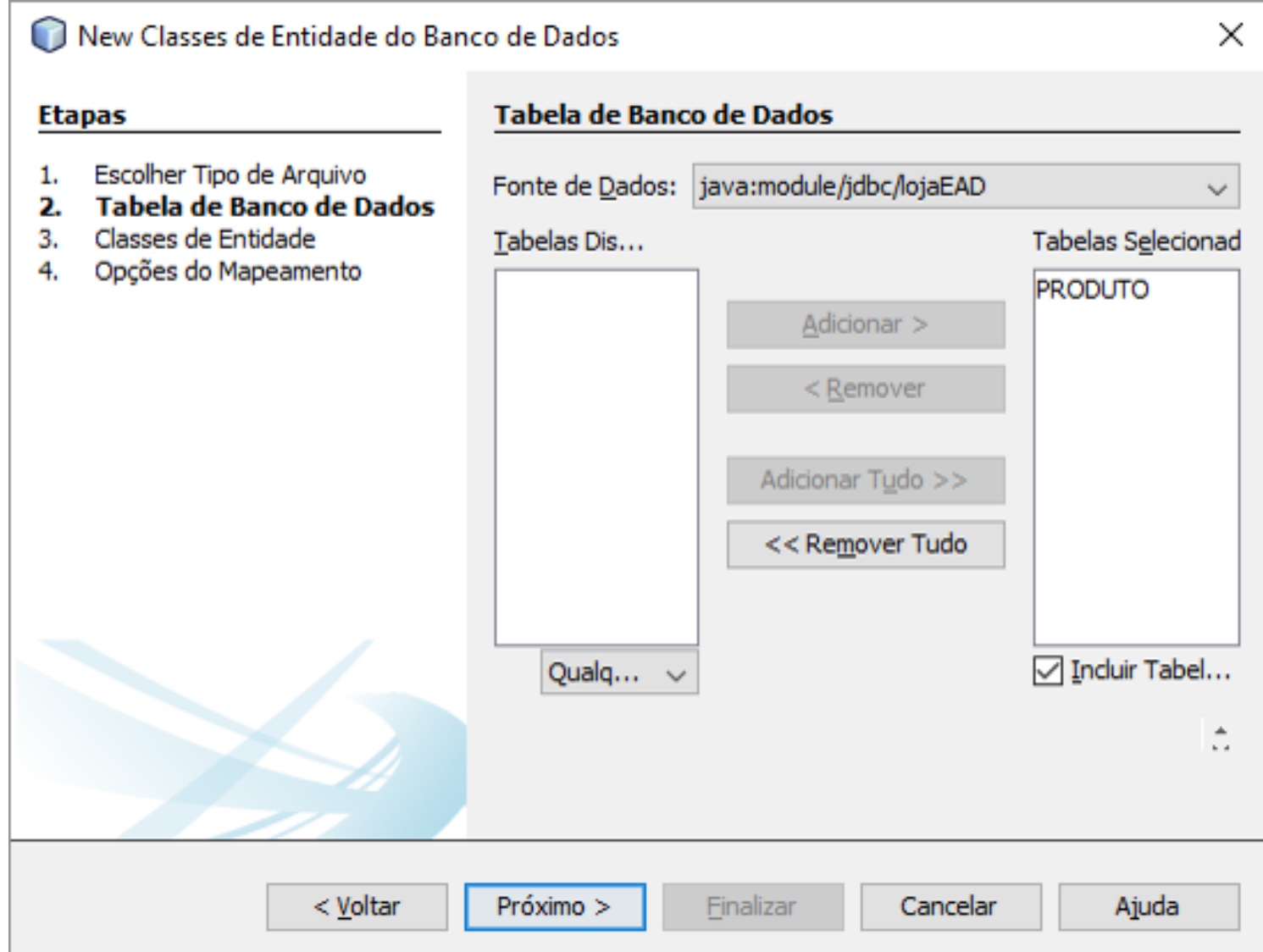
Qualq... ▾

☒ Incluir Tabel...

Selecione ao menos uma tabela.

< Voltar Próximo > Finalizar Cancelar Ajuda

Verificar se as tabelas do banco foram recuperadas e clicar em **Adicionar Tudo**.



Verificar se as tabelas foram adicionadas em **Tabelas Selecionadas** e clicar em **Próximo**.

**Classes de Entidade**

Especifique os nomes e a localização das classes de entidade.

Nomes de Classes:

| Tabela de Banco d... | Nome da Classe | Tipo de Geração |
|----------------------|----------------|-----------------|
| PRODUTO              | Produto        | Novo            |
|                      |                |                 |

...

Projeto: ExemploCompleto-ejb

Localização: Pacotes de Códigos-fonte

Pacote: model

☐ Gerar Anotações de Consulta Nomeada para Campos Persistentes  
☐ Gerar Anotações JAXB  
☐ Gerar Superclasses Mapeadas em vez de Entidades  
☒ Criar Unidade de Persistência

Definir o nome do pacote com **model**, deixar marcada apenas a opção de criação da **Unidade de Persistência** e clicar em **Finalizar**.

A entidade **Produto** será gerada no pacote **model** com o mesmo código que foi anteriormente considerado quando estudamos o JPA.

Nos passos dois até quatro, o que estamos fazendo é definir um novo pool de conexões registrado via **JNDI** no **GlassFish**, e a unidade de persistência utilizará esse pool no acesso a dados, sendo a única diferença do método utilizado por nós na criação de entidades para ambiente desktop.

Isso pode ser observado nos **Arquivos de Configuração** do projeto, a começar pela definição do pool através de **glassfish-resources.xml**, presente no diretório **META-INF**.

Esse arquivo será responsável pela definição do pool, cujo nome adotado será **derby\_net\_LojaEAD\_LojaEADPool** e estará relacionado ao nome JNDI definido anteriormente na geração de entidades.

```
pool-name="derby_net_LojaEAD_LojaEADPool"/>
```

Como estamos em um ambiente corporativo agora, o uso do pool de conexões é um grande diferencial, e o arquivo **persistence.xml** irá configurar tal utilização para o JPA a partir do nome JNDI utilizado, conforme podemos observar na listagem seguinte.

```
xmlns="//xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="//www.w3.org/2001/XMLSchema-instance"
```

```
//xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="ExemploCompleto-ejbPU"
```

```
<jta-data-source>java:module/jdbc/lojaEAD
</jta-data-source>
```

Precisamos observar que as transações agora serão realizadas via JTA (Java Transaction API), ou seja, controladas pelo



```
<persistence-unit name="ExemploCompleto-ejbPU"  
    transaction-type="JTA">
```

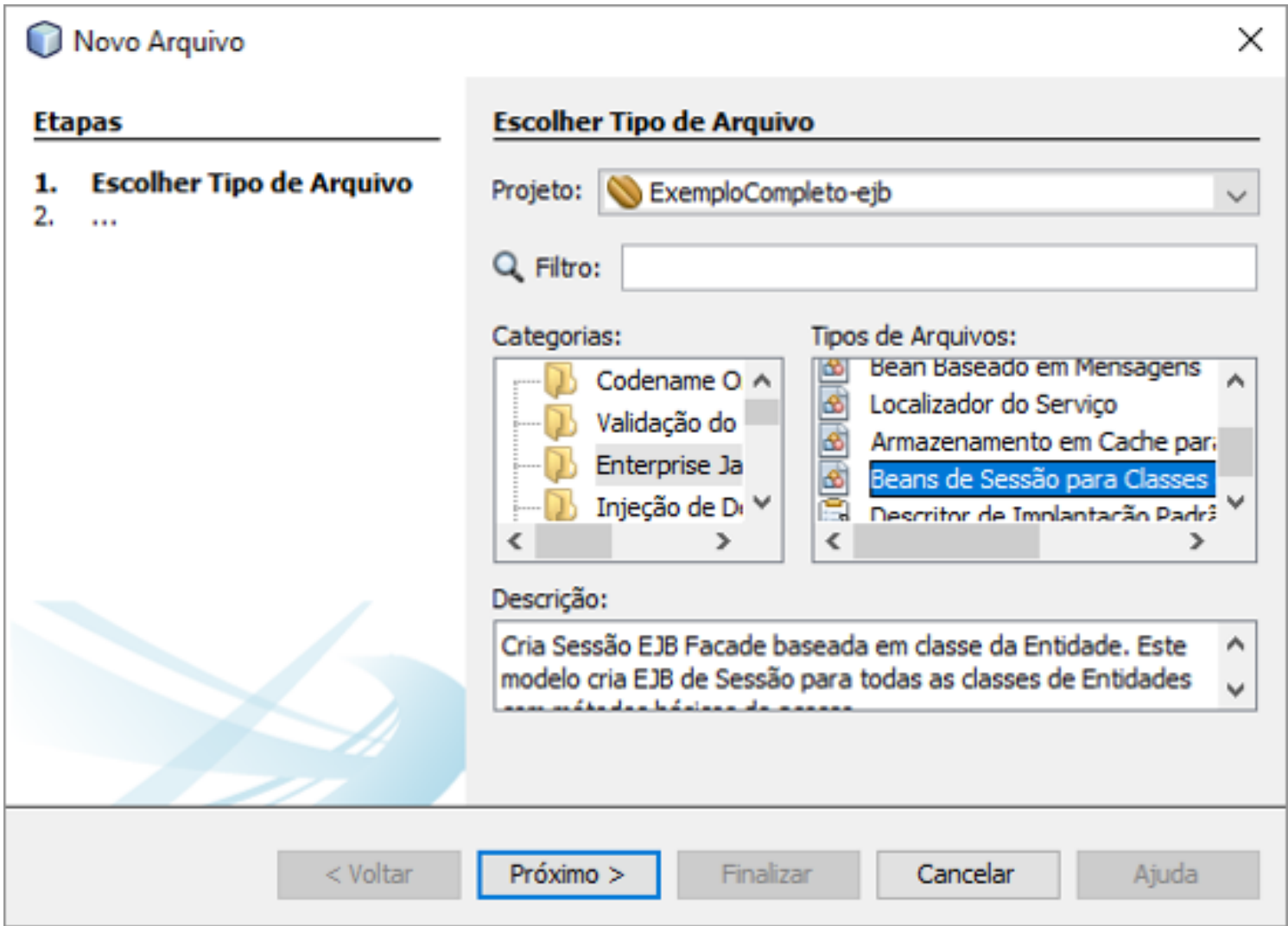
A conexão com o pool é configurada em seguida, sendo também determinada a informação que todas as classes de entidade serão utilizadas, mesmo que não estejam listadas no arquivo persistence.xml.

```
<jta-data-source>java:module/jdbc/lojaEAD  
<exclude-unlisted-classes>>false</exclude-unlisted-classes>
```

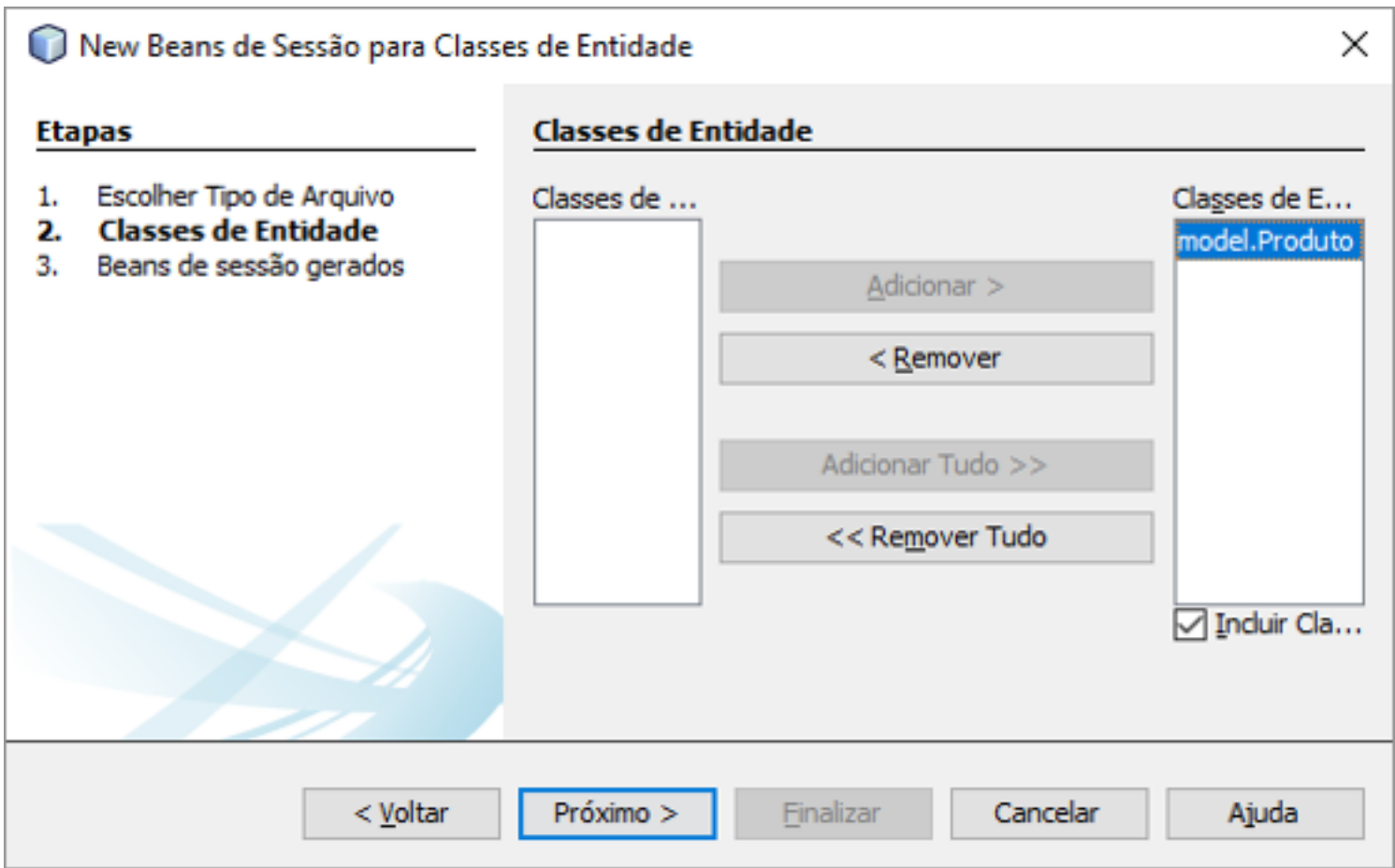
# Criando a camada Control

Resolvida a camada Model, temos que implementar a camada **Control** do MVC, a qual também será definida no projeto **ExemploCompleto-ejb**.

Uma forma muito simples de gerar a camada de controle, nesse caso, será a geração dos Session Beans que funcionarão como Facade de forma automatizada, o que é feito adicionando um **Novo Arquivo** ao projeto e seguindo os seguintes passos:



1. Selecionar o tipo de componente **Enterprise JavaBeans..Beans de Sessão para Classes de Entidade**, e clicar em **Próximo>**.



2. Selecionar as classes de entidade que serão utilizadas e clicar em **Próximo**.

New Beans de Sessão para Classes de Entidade

1. Escolher Tipo de Arquivo

2. Classes de Entidade

3. Beans de sessão gerados

Beans de sessão gerados

Especificar a localização das novas classes de beans de sessão

Projeto: ExemploCompleto-ejb

Local: Pacotes de Códigos-fonte

Pacote: control

Arquivos Criados: <ClassName>Facade, <ClassName>FacadeLo

Criar interfaces:

☒ Local

☐ Remoto

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

3. Definir o nome do pacote com **control**, deixar marcada apenas a opção de interface **Local** e clicar em **Finalizar**.

Ao final desse processo, serão gerados três arquivos: **ProdutoFacade**, **ProdutoFacadeLocal** e **AbstractFacade**.

Todos os processos de bancos de dados são bastante repetitivos e, por isso, o NetBeans gera uma classe chamada **AbstractFacade**, concentrando toda a parte comum da programação para banco com JPA. Podemos observar parte do código de AbstractFacade a seguir.

```
public abstract class AbstractFacade<T> {

    private Class<T> entityClass;
    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }
    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }
    public void edit(T entity) {
        getEntityManager().merge(entity);
    }
}
```

Trata-se de uma classe genérica, aceitando um tipo **T**, e diversos métodos são criados com o uso desse tipo.

```
public void edit(T entity) {
    getEntityManager().merge(entity);
}
```

## Comentário

Notou que não há transação aqui? É porque a transação será gerenciada pelo container JEE com uso do JTA.

O método **getEntityManager** é abstrato, devendo ser implementado pelos seus descendentes, como **ProdutoFacade**, o que podemos observar no código seguinte.

```
@Stateless
public class ProdutoFacade extends AbstractFacade<Produto>
    implements ProdutoFacadeLocal {

    @PersistenceContext(unitName = "ExemploCompleto-ejbPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public ProdutoFacade() {
        super(Produto.class);
    }
}
```

A classe **ProdutoFacade** é um componente do tipo **Stateless** Session Bean, herdando de **AbstractFacade** com uso da entidade Produto, ou seja, onde existe T em AbstractFacade será utilizada a classe **Produto**.

Esse processo poderia ser feito para qualquer entidade, e o código comum não precisa ser replicado, facilitando muito a manutenção.

```
public void edit(Produto entity) {
    getEntityManager().merge(entity);
}
```

Também temos a relação direta do Session Bean com o **EntityManager** através da anotação **@PersistenceContext**, necessitando apenas do nome da **unidade de persistência**.

```
@PersistenceContext(unitName = "ExemploCompleto-ejbPU")
private EntityManager em;
```

Finalmente, temos a interface local, com a assinatura dos métodos na classe de entidade alvo, em vez do T existente nos métodos de AbstractFacade.


```
@Local
public interface ProdutoFacadeLocal {
    void create(Produto produto);
    void edit(Produto produto);
    void remove(Produto produto);
    Produto find(Object id);
    List<Produto> findAll();
    List<Produto> findRange(int[] range);
    int count();
}
```

Agora já temos os métodos necessários para efetuar consultas, incluir, alterar e excluir os dados, com uso das camadas Control e Model.

Não há dependência de ambiente, pois nem sequer temos uma interface visual.

# Criando a camada View

É neste ponto que devemos nos preocupar com a última camada do MVC, a camada **View**, que será implementada no projeto **ExemploCompletoW**.

 Fonte: Shutterstock.

## Atenção

Devido à arquitetura do framework JSF, o **FacesServlet** funcionará como **Front Control** de nossa arquitetura.

Nosso primeiro passo será a criação dos **Managed Beans**, os quais deverão encapsular dados recebidos e chamadas efetuadas, de forma a manter a independência das demais camadas em relação à View.

Como estratégia de implementação, criaremos um bean denominado **ProdutoMB**, o qual ficará responsável por deter os atributos equivalentes de Produto, além de acrescentar um atributo booleano indicando a ocorrência de **inclusão** ou **alteração**, operações nas quais esse bean será utilizado.

```
package view;

import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import model.Produto;

@Named(value = "produtoMB")
@SessionScoped
public class ProdutoMB extends Produto{
    private boolean inclusao = false;
    public ProdutoMB() {
    }

    public boolean isInclusao() {
        return inclusao;
    }
    public void setInclusao(boolean inclusao) {
        this.inclusao = inclusao;
    }
}
```

Esse bean foi criado com o escopo de **sessão** para facilitar a efetivação das operações de inclusão e alteração, como veremos posteriormente.

Quanto ao atributo booleano, ele é denominado **inclusao**, indicando a ocorrência de inclusão quando true e alteração para false.

Outro bean que criaremos visa ao encapsulamento das chamadas ao EJB e o controle do fluxo de navegação, e será denominado **ProdutoControlMB**.

Como podemos observar a seguir, o código desse segundo bean é bem mais extenso:





```
package view;
```

```
import control.ProdutoFacadeLocal;
import java.util.List;
import javax.ejb.EJB;
import javax.inject.Named;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import model.Produto;
```

```
@Named(value = "produtoControlMB")
```

```
@ApplicationScoped
```

```
public class ProdutoControlMB{
```

```
    @EJB
```

```
    ProdutoFacadeLocal facade;
```

```
    @Inject
```

```
    ProdutoMB produtoMB;
```

```
    public ProdutoControlMB() {
    }
```

```
    // Métodos de acesso direto
    public List<Produto> obterTodos(){
        return facade.findAll();
    }
```

```
    // Métodos de navegação
    public String listar(){
        return "ListaProduto?faces-redirect=true";
    }
```

```
    public String incluir(){
        produtoMB.setCodigo(0);
        produtoMB.setNome("");
        produtoMB.setQuantidade(0);
        produtoMB.setInclusao(true);
        return "DadosProduto?faces-redirect=true";
    }
```

```
    public String alterar(Integer idProduto){
        Produto produto = facade.find(idProduto);
        produtoMB.setCodigo(produto.getCodigo());
        produtoMB.setNome(produto.getNome());
        produtoMB.setQuantidade(produto.getQuantidade());
        produtoMB.setInclusao(false);
        return "DadosProduto?faces-redirect=true";
    }
```

```
    public String persistir(){
        return (produtoMB.isInclusao())?incluirX():alterarX();
    }
```

```
    public String incluirX(){
        Produto produto = new Produto(produtoMB.getCodigo());
        produto.setNome(produtoMB.getNome());
        produto.setQuantidade(produtoMB.getQuantidade());
        facade.create(produto);
        return listar();
    }
```

```

}

public String alterarX(){
    Produto produto = facade.find(produtoMB.getCodigo());
    produto.setNome(produtoMB.getNome());
    produto.setQuantidade(produtoMB.getQuantidade());
    facade.edit(produto);
    return listar();
}

public String excluirX(Integer idProduto){
    facade.remove(facade.find(idProduto));
    return listar();
}

}

```

O método **listar** visa apenas ao direcionamento do fluxo de navegação para a página de listagem, sendo chamado também por todas as efetivações de operações, caracterizadas por métodos que trazem a letra X ao final do nome.

```

public String listar(){
    return "ListaProduto?faces-redirect=true";
}

```

Quase todos os métodos fazem chamadas ao EJB, o que é viabilizado pelo uso da anotação correta, conforme podemos observar a seguir.

```

@EJB
ProdutoFacadeLocal facade;

```

Efetuando a referência ao EJB, podemos utilizar diretamente o **facade** como, por exemplo, no método excluir.

```
public String excluirX(Integer idProduto){
    facade.remove(facade.find(idProduto));
    return listar();
}
```

Temos apenas um método de acesso direto — o método **obterTodos**, retornando a lista completa de produtos — que não interfere no fluxo de navegação, e que, por conseguinte, não retorna nenhuma URL.

```
public List<Produto> obterTodos(){
    return facade.findAll();
}
```

Para efetuar as operações de inclusão e alteração, é necessário obter os dados digitados pelo usuário, fato que justifica o uso de um bean com essas informações no escopo da sessão do usuário.

Com o uso de **@Inject** é muito fácil recuperar esse bean a partir de ProdutoControlMB.

```
@Inject
ProdutoMB produtoMB;
```

Após recuperarmos o bean, é só utilizá-lo na operação desejada, como na efetivação da alteração, por exemplo.

Na alteração, é importante encontrar a entidade existente com **find**, alterar os dados desejados e gravar no banco com **edit**.

```
public String alterarX(){
    Produto produto = facade.find(produtoMB.getCodigo());
    produto.setNome(produtoMB.getNome());
    produto.setQuantidade(produtoMB.getQuantidade());
    facade.edit(produto);
    return listar();
}
```

A inclusão transcorre de forma similar, porém não existe a entidade no banco ainda, devendo ser criado um novo Produto, e a gravação é feita com **create**.

**É muito importante observarmos a chamada para listar, ao final desses métodos, de forma a controlar a navegação, direcionando para a tela de listagem de produtos.**

Outro método importante é o **persistir**, que utilizará o atributo **inclusao** de produtoMB para decidir se efetuará a chamada para **incluirX** ou **alterarX**.

```

<code>
    public String persistir(){
        return (produtoMB.isInclusao())?incluirX():alterarX();
    }
</code>

```

Quanto aos métodos **incluir** e **alterar**, eles iniciam os processos referentes, configurando produtoMB com os dados corretos para cada situação, como podemos observar a seguir, no início da alteração.

```

<code>
    public String alterar(Integer idProduto){
        Produto produto = facade.find(idProduto);
        produtoMB.setCodigo(produto.getCodigo());
        produtoMB.setNome(produto.getNome());
        produtoMB.setQuantidade(produto.getQuantidade());
        produtoMB.setInclusao(false);
        return "DadosProduto?faces-redirect=true";
    }
</code>

```

Esses dois métodos direcionam o fluxo para a página DadosProduto.

A partir do código desses dois beans já podemos chegar à conclusão de que precisaremos de duas páginas JSF: **ListaProduto** e **DadosProduto**.

A página **ListaProduto** ficará responsável pela exibição da listagem de produtos, além das chamadas para **incluir**, **alterar** e **excluirX**.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
  <title>Facelet Title</title>
</h:head>
<h:body>
  <h:form>
    <h:commandLink action="#{produtoControlMB.incluir()}"
      value="Novo Produto"/>
    <hr/>
    <h:dataTable value="#{produtoControlMB.obterTodos()}"
      var="p" border="1">
      <h:column>#{p.codigo}</h:column>
      <h:column>#{p.nome}</h:column>
      <h:column>#{p.quantidade}</h:column>
      <h:column><h:commandButton
        action="#{produtoControlMB.excluirX(p.codigo)}"
        value="Excluir"/>
        <h:commandButton
          action="#{produtoControlMB.alterar(p.codigo)}"
          value="Alterar"/>
      </h:column>
    </h:dataTable>
  </h:form>
</h:body>
</html>
```

Nessa página JSF, podemos observar um **DataTable** sendo alimentado pelo método **obterTodos**, com o respectivo preenchimento das linhas contendo os dados de cada produto da lista.

Note como os **CommandButtons** trabalham com as **ações** de forma dinâmica.

```
      action="#{produtoControlMB.alterar(p.codigo)}"
      value="Alterar"/>
```

Tanto para **alterar** como para **excluirX**, nós utilizamos, como parâmetro, o atributo **código** do produto corrente na lista.

Também acionamos o método **incluir** através de um **CommandLink**.

```
<h:commandLink action="#{produtoControlMB.incluir()}"
    value="Novo Produto"/>
```

De forma geral, o código é bastante simples e direto. O mesmo ocorre com a página **DadosProduto**, conforme podemos observar a seguir.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            Código:<br/>
            <h:inputText value="#{produtoMB.codigo}"
                readonly="#{!produtoMB.inclusao}"/><br/>
            Nome:<br/>
            <h:inputText value="#{produtoMB.nome}"/><br/>
            Quantidade:<br/>
            <h:inputText value="#{produtoMB.quantidade}"/>
            <p><h:commandButton
                action="#{produtoControlMB.persistir()}"
                value="Gravar"/>
            </p>
        </h:form>
    </h:body>
</html>
```

Basicamente, temos um formulário simples, onde os valores dos campos são preenchidos a partir dos dados do bean **produtoMB**, devendo ser feita a observação de que o campo de código só pode ser modificado na operação de inclusão.

```
Código:<br/>
<h:inputText value="#{produtoMB.codigo}"
  readonly="#{!produtoMB.inclusao}"/><br/>
```

Ao final do formulário, encontramos uma chamada ao método persistir, o qual decidirá entre a inclusão ou alteração do produto a partir do atributo **inclusao** do bean.

```
action="#{produtoControlMB.persistir()}"
value="Gravar"/>
```

O último passo é a inclusão de uma chamada inicial a partir do index.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    <h:form>
      <h:commandLink value="Listar Produtos"
        action="#{produtoControlMB.listar()}" />
    </h:form>
  </h:body>
</html>
```

Finalmente, executamos o projeto principal (ExemploCompleto), identificado pelo ícone de um **triângulo**. Podemos observar as telas do sistema a seguir.



## Atenção

Podemos aumentar a independência da página de listagem criando um bean com escopo de sessão para guardar a lista de produtos, e preenchendo esse bean na chamada para listar.

Para efetuar essa mudança, primeiro criamos o bean, o qual deve conter apenas um atributo do tipo List<Produto>, como podemos observar a seguir.

```
package view;

import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import java.io.Serializable;
import java.util.List;
import model.Produto;

@Named(value = "produtoContainerMB")
@SessionScoped
public class ProdutoContainerMB implements Serializable {
    private List<Produto> lista;

    public ProdutoContainerMB() {
    }

    public List<Produto> getLista() {
        return lista;
    }
    public void setLista(List<Produto> lista) {
        this.lista = lista;
    }
}
```

Esse bean deve ser referenciado no código de ProdutoControlMB, e o método listar será modificado para preencher o atributo lista do mesmo.

```
@Inject
ProdutoContainerMB produtoContainerMB;

// Métodos de navegação
public String listar(){
    produtoContainerMB.setLista(facade.findAll());
    return "ListaProduto?faces-redirect=true";
}
```

Finalmente, o DataTable deverá carregar os dados a partir do atributo lista.

```
<h:dataTable value="#{produtoContainerMB.lista}" var="p"
border="1">
```

Essa mudança não traz nenhum impacto para a funcionalidade do sistema, mas permite maior isolamento de ProdutoControlMB, o qual passa a ser acessado apenas nos métodos de navegação, evitando chamadas diretas ao bean de controle.

**Apesar de simples, nosso pequeno projeto contempla listagem, alteração, exclusão e inclusão de produtos, implementado em uma arquitetura MVC, utilizando um Servlet como Front Control por meio do JSF.**

## Atividade

---

1. Para efetuar o mapeamento objeto-relacional, na camada Model, devemos utilizar o JPA, o qual é baseado em anotações, e a chave primária deve ser anotada como:

- a) @Column
- b) @Inject
- c) @Entity
- d) @Table
- e) @Id

2. Para acessar um Managed Bean a partir de outro, qual anotação deve ser utilizada?

- a) @Named
- b) @Inject
- c) @SessionScoped
- d) @Deprecated
- e) @Override

3. Considerando a entidade Produto, utilizada nos exemplos, e o Session EJB, denominado ProdutoFacade, com os métodos para inclusão, alteração, exclusão e consulta às entidades desse tipo, implemente um ManagedBean com o método **limpar**, o qual deverá buscar todos os produtos com quantidade zerada e removê-los, depois navegue para .

## Referências

---

CASSATI, J. P. **Programação servidor em sistemas web**. Rio de Janeiro: Estácio, 2016.

DEITEL, P.; DEITEL, H. **Ajax, rich internet applications e desenvolvimento web para programadores**. São Paulo: Pearson Education, 2009.

\_\_\_\_\_. **Java, como programar**. 8. ed. São Paulo: Pearson, 2010.

MONSON-HAEFEL, R.; BURKE, B. **Enterprise Java Beans 3.0**. 5. ed. São Paulo: Pearson, 2007.

## Próxima aula

---

## Explore mais

---

- [Sintaxe JPQL](https://thoughts-on-java.org/jpql/); <<https://thoughts-on-java.org/jpql/>>
- [Tutorial de JPA](https://www.tutorialspoint.com/jpa/); <[https://www.tutorialspoint.com/jpa](https://www.tutorialspoint.com/jpa/)>
- [Noções gerais de Enterprise Java Beans](https://docs.oracle.com/javaee/6/tutorial/doc/gijjsz.html); <<https://docs.oracle.com/javaee/6/tutorial/doc/gijjsz.html>>
- [Tutorial de JSF com NetBeans](https://netbeans.org/kb/docs/web/jsf20-intro_pt_BR.html). <[https://netbeans.org/kb/docs/web/jsf20-intro\\_pt\\_BR.html](https://netbeans.org/kb/docs/web/jsf20-intro_pt_BR.html)>