→ Introdução ao Java Server Faces

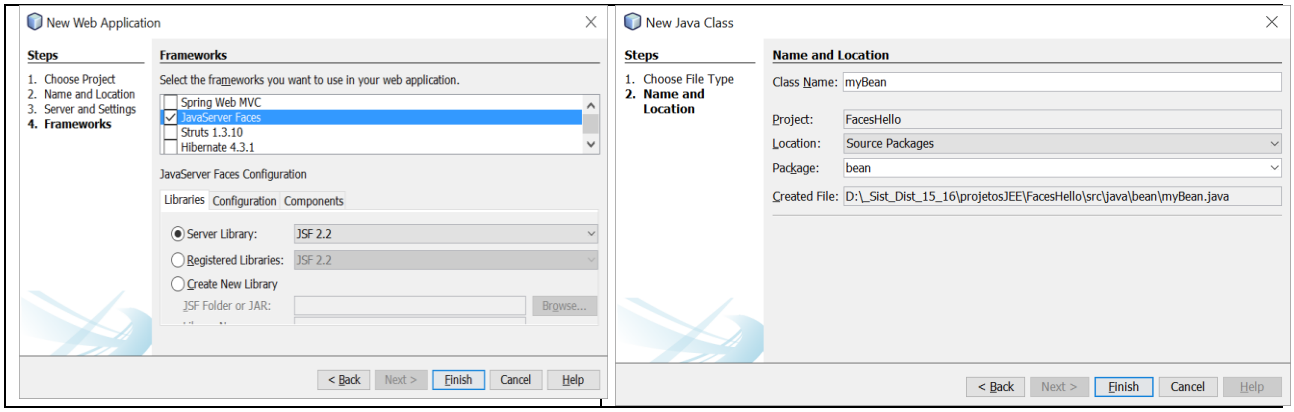
A – Hello World

1 - Criar uma Web Application (FacesHello) com interface em JSF.

a) *New project / Java Web / Web application / Next/ atribuir o nome ao projecto/ Next / seleccionar versão do JavaEE e o servidor Glasfish /* ***Next /***

seleccionar a framework JavaServer faces */ Finish].*

- Executar a aplicação e observar que a página **index.xhtml** é aberta no Browser definido por omissão.



b) De seguida vamos criar um "Named bean" muito simples e aceder-lhe por JSF (Um Named bean, ou CDI Bean, é uma classe de controlo que substitui o antigo "Managed Bean" para JSF).

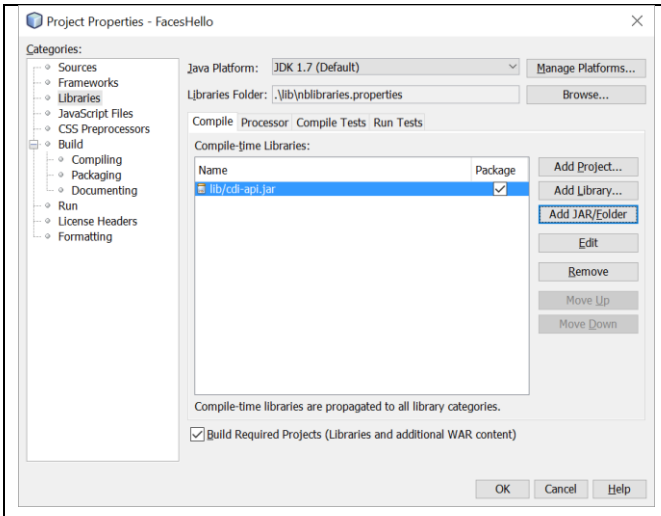
- Em Source Packages, click com o botão direito do rato, seleccione new/ java class e crie a classe mybean no package bean. Acima do cabeçalho da classe coloque as seguintes anotações:

```
@Named (value = "myBean")
@RequestScoped
```

Ao fazer fix imports a anotação RequestScoped deve corresponder ao package **javax.enterprise.context.RequestScoped.**

c) Se o package, não estiver disponível, adicione ao projecto a biblioteca cdi-api.jar. Para isso selecione o seu projecto / click com o botão direito do rato / Properties/ libraries / add JAR/Folder e seleccione o ficheiro cdi-api.jar que deverá estar na diretoria lib da sua instalação do GlassFish (ver figura que se segue à esquerda).

Na classe myBean declare uma String name, o getter e setter respectivo e um construtor sem parâmetros. A sua classe deverá ficar como na figura da direita:

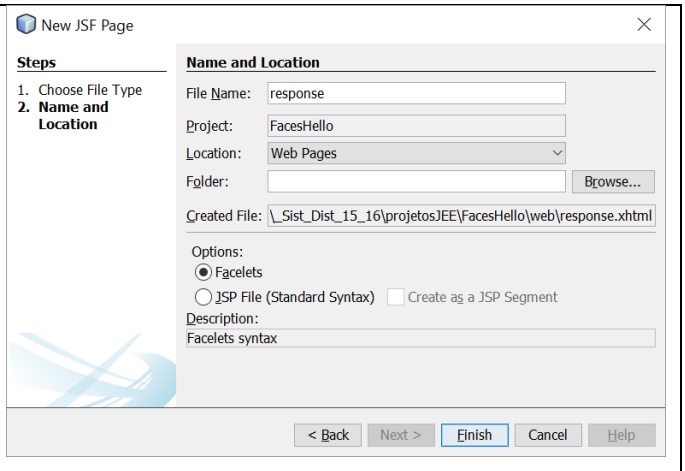



```
package bean;
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;
@Named (value = "myBean")
@RequestScoped
public class myBean {
   private String name;

   public myBean() {
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
}
```

d)Vamos agora alterar o ficheiro index.xhtml: No body insira o código da figura abaixo à esquerda. Nesta página lemos o nome do utilizador e com o botão "submit" invocamos a página response.xhtml que vamos criar de seguida. No package Web Pages seleccione: new / JSF page, e atribua-lhe o nome response.

```
<h:form>
 <h2>What's your your name?</h2>
 <h:inputText id="username"
       title="My name is: "
       value="#{myBean.name}"
       required="true"
       requiredMessage="Error: A name is
required."
      maxlength="25" />
 <p></p>
 <h:commandButton id="submit" value="Submit"
action="response" />
<h:commandButton id="reset" value="Reset"
   type="reset" />
 </h:form>
```



- No body da página response.xhtml insira o código da imagem da abaixo à esquerda:

<h:form> Hello, #{myBean.name}! <h:commandButton id="back" value="Back" action="index" /> </h:form>	**What's your your name?** SD Faces Submit Reset	**Hello, SD Faces!** Back

- Depois de executar o projecto explore os seus componentes. Adicione novos campos à classe e altere as páginas criadas de forma a mostrar o seu valor.

Para Explorar:
[Estude como construir interfaces em java Server Faces explorando o tutorial abaixo, ou outros que encontre na web.
https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm *]*

B – Aplicação Web com acesso a base de dados e interface em JSF.

2 -Vamos repetir o exercício 1 da folha 9 (1 – Gerar JPA "entities", criar um EJB para aceder à Base de Dados …) mas agora queremos construir a interface com o utilizador em JSF.

a) Criar uma web application com a framework JSF (chame ao projecto JPAentitiesFaces):
[New project / Java Web / Web application / Next/ atribuir o nome ao projecto/ Next / seleccionar versão do JavaEE e o servidor Glassfish / Finish].

b) Se no primeiro exercício desta ficha (10) teve de adicionar o jar **cdi-api.jar**, repita o processo descrito na alínea c) do exercício 1.

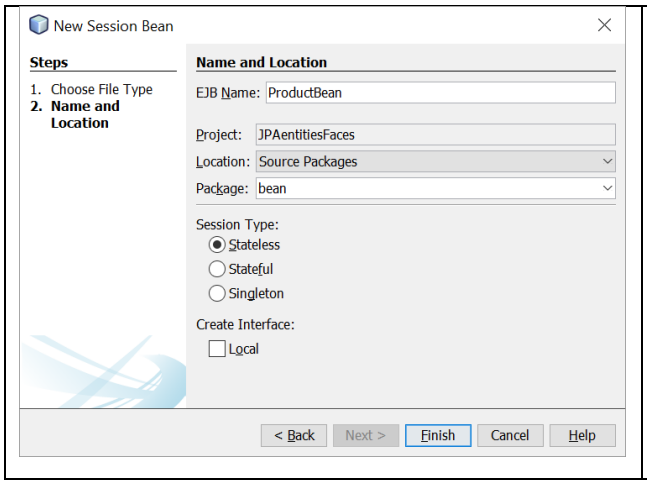
c) Selecione agora a tabela PRODUCT. Tal como no exercício anterior vamos criar entities para a tabela PRODUCT e tabelas associadas com esta da base de dados Sample.
[Rigth-click no projeto e selecionar: New / Other … / Persistence / **Entity classes from database** /Next…
- Selecionar no campo Data Source o valor "jdbc/sample". Selecionar a tabela PRODUCT e fazer Add. Após Next, introduzir **"entities"** para o nome do package. Next e Finish]

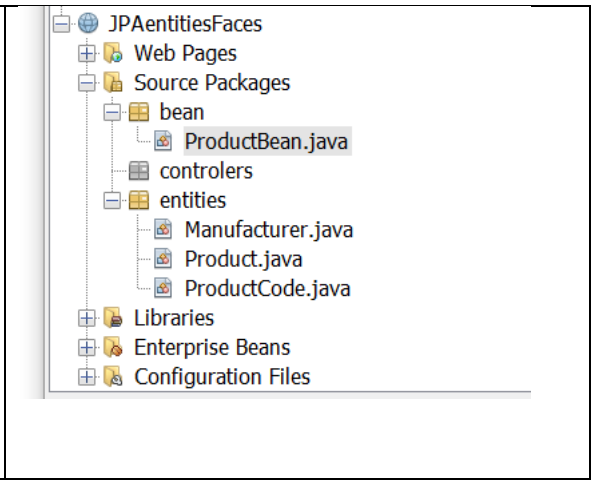
Para cada tabela seleccionada da base de dados temos uma entitiy (isto é, uma classe que mapeia uma tabela).

d) Vamos agora criar um EJB que aceda à entity Product e nos permita manipular os seus dados.

- Criar um stateless Session Bean de nome ProductBean:

[selecionar o projeto / new session bean/ stateless. Coloque o Bean num package de nome "beans" e faça finish] .





- No EJB ProductBean vamos:

1 - injetar uma instância de EntityManager;. após o cabeçalho da classe inserir:

```
@PersistenceContext
   EntityManager em;
```

2 - construir um método para consultar todos os produtos da tabela Product:

```
public List<Product> getProducts () {
    return (List<Product>) em.createNamedQuery("Product.findAll").getResultList();
}
```

Repare que estamos a usar um dos queries criados na entitty Product (findAll).

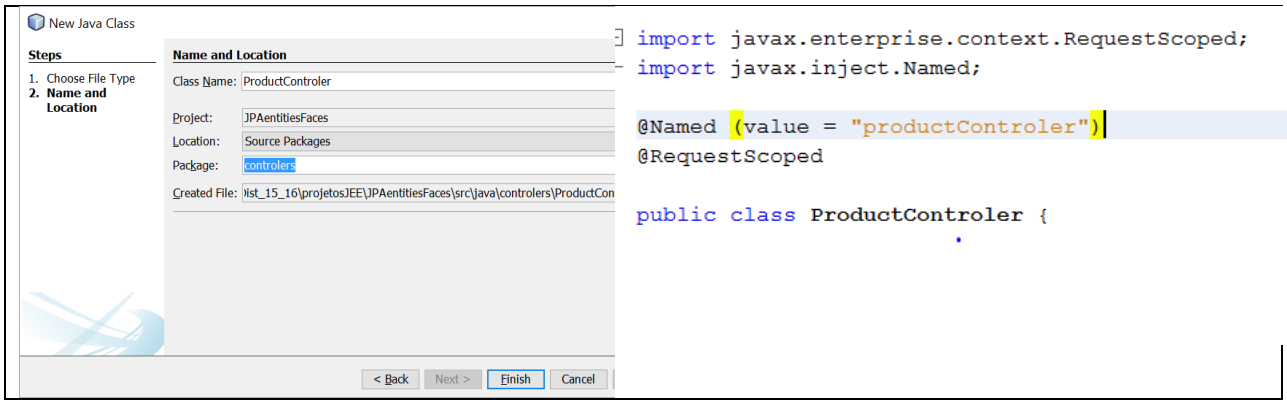
3 – construir um método para inserir um novo produto:

```
public Product addProduct(Product   prd) {
        em.persist(prd);
        return prd;
  }
```

Nota: fazer import entities.;*

e) Criar um bean de controlo que acede aos métodos do EJB e interage com páginas JSF.

- Em Source Packages, click com o botão direito do rato, seleccione new/ java class e crie a classe ProductControler no package "**controlers"**.



- Acima do cabeçalho da classe coloque as seguintes anotações:
@Named (value = " productControler ")
@RequestScoped

- Ao fazer fix imports a anotação RequestScoped deve corresponder ao package **javax.enterprise.context.RequestScoped.**

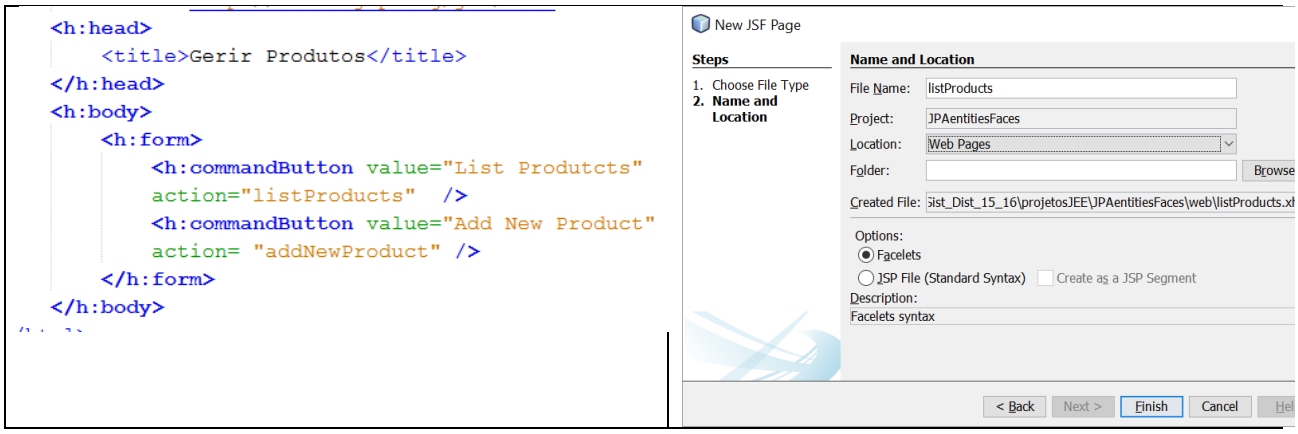
- Na classe ProductControler vamos injectar o EJB ProductBean que criamos na alínea anterior e criar um método para invocar getProducts do EJB. O interior da classe ficará:

```
@EJB
ProductBean prod;
List<Product> productList = new ArrayList<>();
 public List<Product>  getProductList() {
       productList = prod.getProducts();
       return productList;
 }
```

f) Criar as páginas JSF

- Começamos por modificar a página index.xhtml que está na pasta Web Pages (ver figura abaixo à esquerda).

Criamos um botão para cada opção, listar produtos e inserir produto. Falta criar as páginas listProducts.xhtml e addNewProduct.xhtml. Para isso: botão direito do rato em Web Pages / new / JSF page, dar o nome listProducts / Finish.




```
<h:head>
    <title>Gerir Produtos</title>
</h:head>
<h:body>
    <h:form>
        <h:commandButton value="List Produtcts"
        action="listProducts"  />
        <h:commandButton value="Add New Product"
        action= "addNewProduct" />
    </h:form>
</h:body>
```

Construa a página listProducts.xhml de modo a ficar com o conteúdo abaixo. As 5 primeiras linhas já estão no ficheiro que criou. A biblioteca da linha 6 pode inserir manualmente ou de forma automática com o editor quando este não identificar o símbolo *facet* .

```
<?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"
     xmlns:f="http://xmlns.jcp.org/jsf/core">
   <h:head>
      <title> Products List</title>
   </h:head>
   <h:body>
      <h:dataTable value="#{productControler.productList}" var="produto"  border="1">
      <h:column>
           <f:facet name="header">
              <h:outputText value="Product ID"/>
           </f:facet>
         <h:outputText value="#{produto.productId}"/>
      </h:column>
          </h:dataTable>
     <h:form>
     <h:commandButton id="back" value="Back" action="index" />
     </h:form>
   </h:body>
</html>
```

- Repare que no final o botão Back permite voltar à página inicial.

Exercício - Complete a pagina listProducts de forma a mostrar todos os dados da tabela Product.

h) Inserir novo produto (Classe ProductControler)

Para inserirmos um novo produto regressamos à classe ProductControler. Precisamos de construir um objeto do tipo Product e adicioná-lo à base de dados. Um objeto do tipo Product tem dois campos que são objectos. O manufacturerId que é um Manufacturer e o prodCode que é um ProductCode:

- Adicionar à classe ProductControler a declaração dos objectos e a sua instanciação, para podermos atribuir-lhe valores a ler na página web:

```
Product novoProduto = new Product();
Manufacturer mn = new Manufacturer();
ProductCode pc = new ProductCode();
```

- Insira os respectivos getters e setters.
- Estude e adicionar à classe o método para adicionar o novo produto:

```
public String addNewProduct() {
        novoProduto.setProductCode(pc);
        novoProduto.setManufacturerId(mn);

        prod.addProduct(novoProduto);

        productList = prod.getProducts();
       return "listProducts.xhtml";
   }
```

Atribuímos ao novoProduto os valores do productCode e do ManufacturerId (*linhas 1 e 2)*
Usamos o método addProduct do EJB ProductBean para inserir o produto na base de dados *(linha 3)*
Usámos o método getProducts do EJB para consultar a nova lista de produtos *(linha 4)*.
Finalmente o método devolve uma String com o nome da página que lista os produtos, para vermos a nova lista já com o produto inserido.

i) Inserir novo produto (página addNewProduct.xhtml)

Estude a página addNewProduct.xhtml listada abaixo, adicione-a ao seu projecto e execute o projecto. A página lê os valores do novo produto, apresenta um botão "Save" que irá invocar o método addNewProduct(). Observe as linhas a bold.

```
<h:body>
    <h:form>
     <h:panelGrid columns="2" >
     <h:outputLabel value="Product ID: "/>
     <h:inputText value="#{productControler.novoProduto.productId}"/>
     <h:outputLabel value="Manufacturer ID: "/>
     <h:inputText value="#{productControler.mn.manufacturerId}"/>
     <h:outputLabel value="Product code: "/>
     <h:inputText value="#{productControler.pc.prodCode}"/>
     <h:outputLabel value="Purchase Cost: "/>
     <h:inputText value="#{productControler.novoProduto.purchaseCost}"/>
     <h:outputLabel value="Quantity on hand: "/>
     <h:inputText value="#{productControler.novoProduto.quantityOnHand}"/>
     <h:outputLabel value="Markup: "/>
     <h:inputText value="#{productControler.novoProduto.markup}"/>
     <h:outputLabel value="Available: "/>
     <h:inputText value="#{productControler.novoProduto.available}"/>
     <h:outputLabel value="Description: "/>
     <h:inputText value="#{productControler.novoProduto.description}"/>
     </h:panelGrid>
     <h:commandButton value="Save"
                action="#{productControler.addNewProduct()}"/>
   </h:form>
 </h:body>
```

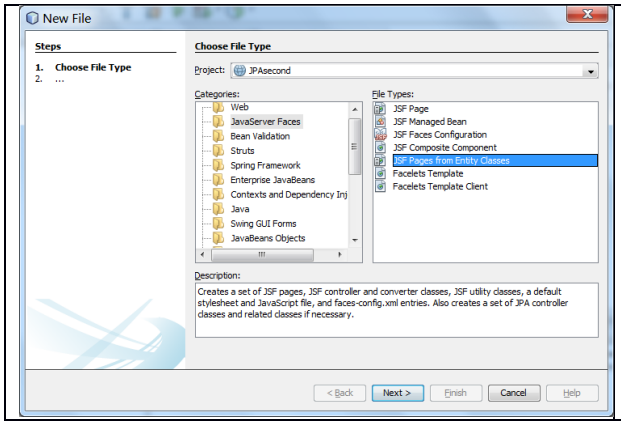
C - Gerar uma aplicação JSF completa a partir de Entity classes

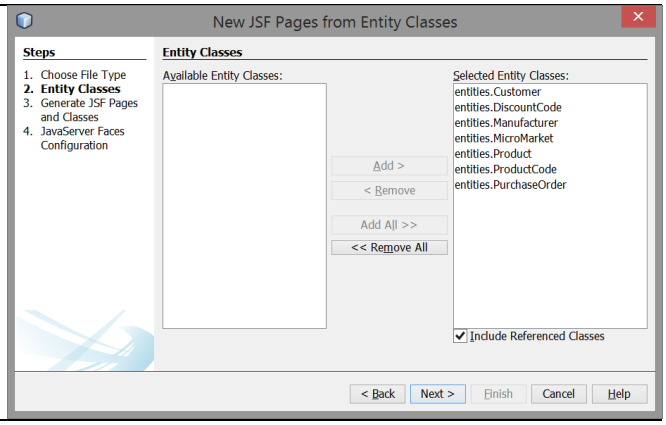
O processo pode ser ainda mais automatizado. Vamos ver com criar uma aplicação completa a partir de uma base de dados já existente.

a) Repita o processo dos exercícios anteriores: - criar uma "web application com a framework JSF"; criar entities a partir de uma base de dados já existente, seleccionando agora todas as tabelas da base de dados. Se tiver outra base de dados criada pode usá-la.

b) Adicione ao seu projecto o jar **cdi-api.jar** 1.

c) No projeto, selecionar File / new File / Java Server Faces / JSF from Entity Classes e selecionar todas as entities.

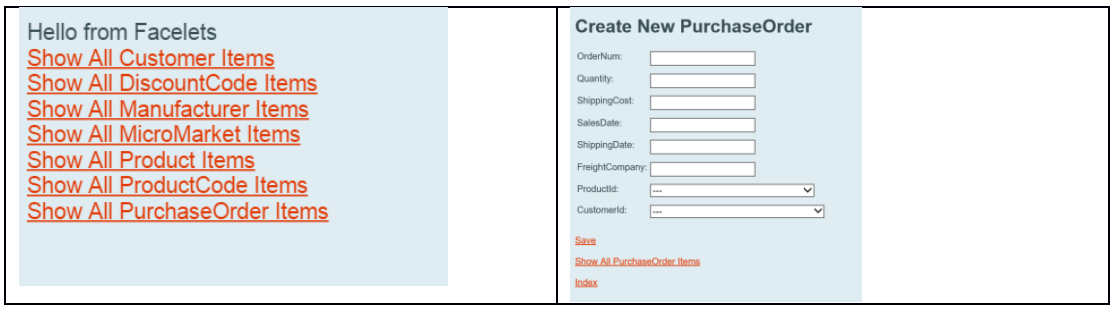




- Especificar um package para os EJBs que acedem à base de dados (em Session Package insira beans) e especifique outro package para as classes de controlo (em JSF classes Package insira controlers). Deixar em branco a localização das páginas JSF, por omissão serão criadas na pasta Web Pages.

- Após terminar, executar a aplicação. Foi criada uma aplicação web completa que permite efetuar as operações CRUD na(s) tabela(s) da base de dados considerada.

- Na janela projeto observe e explore as classes que foram criadas. O Wizard do NetBeans cria, na pasta Web Pages, uma pasta para cada entity. Para cada entity criou também um Session Bean (facade) e uma classe de controlo.

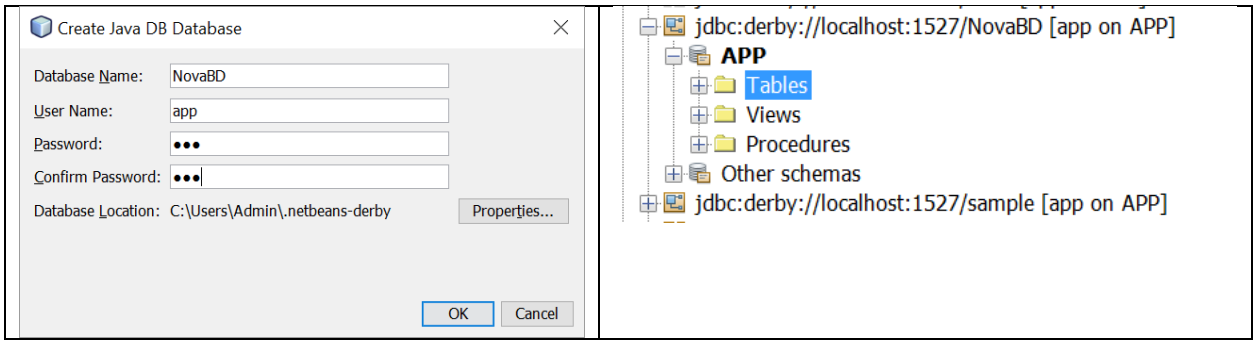


D – Criar uma bases de dados e gerar uma aplicação JSF completa a partir de entity classes

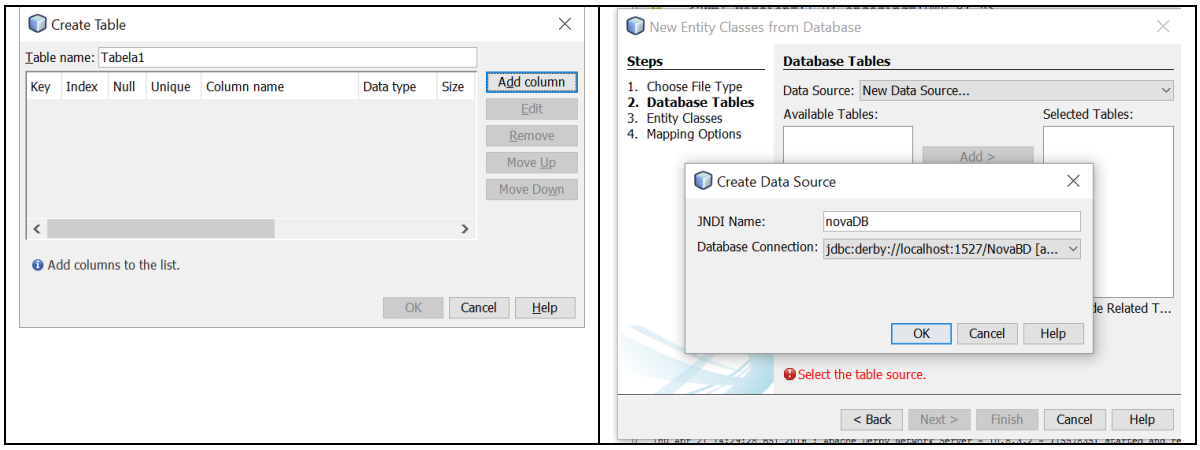
- Crie uma base de dados simples em javaDb ou outro sgdb à sua escolha.

Criar uma base de Dados em Java DB: Em services, seleccionar Java DB / click no Botão direito do rato / create Database. Dar o nome à base de dados e atribuir um user name e password.

É criado um driver que pode observar mais abaixo. Se selecionar o driver associado à base de dados e seleccionar connect pode criar tabelas na base de dados. Em Tables seleccionar create table. Continuar o processo criando as colunas e depois outras tabelas …



Numa web application, seleccionar Entity classes from Database, ao seleccionar a base de dados, criar um novo Data Dource e associá-lo ao driver criado.



Depois de adicionar a biblioteca cdi-api.jar, gerar as Java Server Faces como anteriormente.