Folha 10 - 1

# → 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.

New Web Applicatio	'n		×	🜍 New Java Class		>	×
Steps	Frameworks			Steps	Name and	Location	
1. Choose Project	Select the frameworks you	want to use in your web application.		1. Choose File Type	Class <u>N</u> ame:	myBean	
3. Server and Settings	Spring Web MVC JavaServer Faces		^	Location	Project:	FacesHello	
4. Traineworks	Struts 1.3.10 Hibernate 4.3.1		~		Location:	Source Packages	$\sim$
	JavaServer Faces Configural	tion			Pac <u>k</u> age:	bean	~
	Libraries Configuration C	omponents			Created File:	D:\_Sist_Dist_15_16\projetosJEE\FacesHello\src\java\bean\myBean.java	
	• Server Library:	JSF 2.2	~				
	O Registered Libraries:	JSF 2.2	~				
	○ <u>C</u> reate New Library						
	JSF Folder or JAR:		Browse				
		< Back Next > Finish	Cancel <u>H</u> elp			< Back Next > Finish Cancel Help	

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).

Folha 10 - 2Na 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:

Project Properties - FacesHello	×	package bean;
Project Properties - FacesHello   Categories:   • Sources   • Frameworks   • Ubraries   • Daradiscript Files   • Compiling   • Acading   • Run   • Concumenting   • Run   • Ubraries Headers   • Formatting    Compile-time libraries are propagated to all libraries and addition  Compile-time libraries and  Compile-time libraries  Compil	Manage Platforms  Package Add Project Add JAr/Eolder Edit Bernove Move Up Move Up Move Down any categories. ald WAR content)  Kategories	<pre>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; } } }</pre>

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></h:form>	New JSF Page			×
<h2>What's your your name?</h2>	Steps	Name and	Location	
<h:inputtext <="" id="username" td=""><td>1. Choose File Type</td><td>File <u>N</u>ame:</td><td>response</td><td></td></h:inputtext>	1. Choose File Type	File <u>N</u> ame:	response	
title="My name is: "	2. Name and Location	Project:	FacesHello	
value="#{myBean.name}"		Location:	Web Pages 🗸	
required="true"		Folder:		Browse
requiredMessage="Error: A name is		Created File:	L_Sist_Dist_15_16\projetosJEE\FacesHello\web\re	esponse.xhtml
required."		Options:		
maxlength="25" />		Facelets		
		Description:	(Standard Syntax) Create as a JSP Segment	
<h:commandbutton <="" id="submit" td="" value="Submit"><td></td><td>Facelets synt</td><td>tax</td><td></td></h:commandbutton>		Facelets synt	tax	
action="response" />				
<h:commandbutton <="" id="reset" td="" value="Reset"><td></td><td></td><td></td><td></td></h:commandbutton>				
type="reset" />			< Back Next > Finish Cancel	Help

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

Paula Prata, Departamento de Informática da UBI

## Sistemas Distribuídos - 2016/2017 Curso: Engª Informática

Folha 10 - 3

<h:form> <h2>Hello, #{myBean.name}!</h2></h:form>	What's your your name?	Hello, SD Faces!
<h:commandbutton action="index" id="back" value="Back"></h:commandbutton>	SD Faces × Submit Reset	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. <u>https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm</u>]

### 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 / seleccionar a framework **JavaServer faces**/ 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]

#### Sistemas Distribuídos - 2016/2017 Curso: Engª Informática

Folha 10 - 4

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].

New Session Bean		×	De De JPAentitiesFaces
Steps	Name and Location		🕀 🕀 Web Pages
Steps 1. Choose File Type 2. Name and Location	Name and Location         EJB Name:       ProductBean         Project:       JPAentitiesFaces         Location:       Source Packages         Package:       bean         Session Type: <ul> <li>Stateful</li> <li>Singleton</li> <li>Create Interface:</li> <li>Local</li> </ul>		Source Packages  Source Packages  Source Packages  Source Packages  Source Packages  Source Packages  Source ProductBean.java  Source ProductBean.java  Source Product.java  Source Product.java  Source ProductCode.java  Source Packages  Source
			🗄 🗟 Configuration Files
	< Back Next > Einish Cancel Help		

#### - 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.*;
```

Paula Prata, Departamento de Informática da UBI

Folha 10 - 5

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**".

New Java Class     Steps     . Choose File Type     Name and     Location	Name and Location Class Name: ProductControler	<pre>import javax.enterprise.context.RequestScoped; import javax.inject.Named;</pre>
Location	Project: JPAentitiesFaces	<pre>@Named (value = "productControler")</pre>
	Location: Source Packages	Request Scoped
	Package: controlers	enequesebeeped
	Created File: Nst_15_16\projetosJEE\JPAentitiesFaces\src\java\controlers\ProductCon	<pre>public class ProductControler {     .</pre>
	< Back Next > Einish Cancel	

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.

Paula Prata, Departamento de Informática da UBI

Sistemas Distribuídos - 2016/2017 Curso: Engª Informática

		Folha 10 - 6
<h:head> <title>Gerir Produtos</title></h:head>	New JSF Page Steps	Name and Location
<pre> <h:body></h:body></pre>	1. Choose File Type 2. Name and Location	File Name:       listProducts         Project:       JPAentitiesFaces         Location:       Web Pages         Folder:       Browse         Created File:       ist_Dist_15_16\projetosJEE\JPAentitiesFaces\web\listProducts.xt         Options:       Image: Create as a JSP Segment         Description:       Facelets         Vacelets syntax       Create as a JSP Segment         Escription:       Facelets syntax

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.

*Folha 10 - 7* 

## 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 objeto do tipo Manufacturer e o prodCode que é um objeto do tipo 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.

Folha 10 - 8

#### 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:form>			
<h:panelgrid columns="2"></h:panelgrid>			
<h:outputlabel value="Product ID: "></h:outputlabel>			
<h:inputtext value="#{productControler.novoProduto.productId}"></h:inputtext>			
<h:outputlabel value="Manufacturer ID: "></h:outputlabel>			
<h:inputtext value="#{productControler.mn.manufacturerId}"></h:inputtext>			
<h:outputlabel value="Product code: "></h:outputlabel>			
<h:inputtext value="#{&lt;b&gt;productControler.pc.prodCode&lt;/b&gt;}"></h:inputtext>			
<h:outputlabel value="Purchase Cost: "></h:outputlabel>			
<h:inputtext value="#{productControler.novoProduto.purchaseCost}"></h:inputtext>			
<h:outputlabel value="Quantity on hand: "></h:outputlabel>			
$<\!\!h:inputText\ value="\#{productControler.novoProduto.quantityOnHand}"/\!\!>$			
<h:outputlabel value="Markup: "></h:outputlabel>			
<h:inputtext value="#{productControler.novoProduto.markup}"></h:inputtext>			
<h:outputlabel value="Available: "></h:outputlabel>			
<h:inputtext value="#{productControler.novoProduto.available}"></h:inputtext>			
<h:outputlabel value="Description: "></h:outputlabel>			
<h:inputtext value="#{productControler.novoProduto.description}"></h:inputtext>			
<h:commandbutton <="" td="" value="Save"></h:commandbutton>			
action="#{productControler.addNewProduct()}"/>			

</h:form>

</h:body>

- Ao testar a aplicação, tenha em conta que terá de inserir dados válidos. Veja o conteúdo das tabelas em services / databases / ...

# 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.

Sistemas Distribuídos - 2016/2017 Curso: Engª Informática

Folha 10 - 9

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, se necessário.

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.

# Sistemas Distribuídos - 2016/2017 Curso: Engª Informática

		Folha	10 -	10
Hello from Facelets	Create New PurchaseOrder			
Show All Customer Items	OrderNum:			
Show All DiscountCode Items	Quantity:			
Show All Manufacturer Items	ShippingCost:			
Show All MicroMarket Items	SalesDate:			
Show All Product Items	ShippingDate:			
Show All ProductCode Items	Productid:			
Show All PurchaseOrder Items	Customerid:			
	Save			
	Show All PurchaseOrder Items			
	Index			

# 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.

# Sistemas Distribuídos - 2016/2017 Curso: Engª Informática

				Folha	a 10 - 11
🗊 Create Table		×	New Entity Classes	from Database	×
Table name: Tabela1			Steps	Database Tables	
Key Index Null Unique Column name	Data type Size	Add column Edit Remove Move Up Move Down	1. Choose File Type 2. Database Tables 3. Entity Classes 4. Mapping Options Create D JNDI Name: Database Co	Data Source: New Data Source Available Tables: Data Source novaDB nnection: jdbc:derby://localhost:1527/Nov	Selected Tables:
• Add columns to the list.	ОК С	ancel <u>H</u> elp		OK Cancel Select the table source. < Back Next > Fin	le Related T

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