Java Enterprise Edition - Java EE

Java Server Faces

com Exemplos Básicos usando o NetBeans e GlassFish

Sumário

1. Criando uma nova base de dados para o exemplo

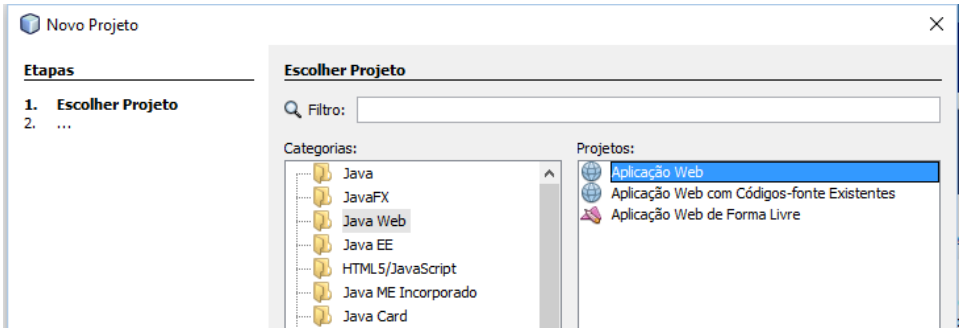
Crie um novo usuário com o login: **javaee**, senha: **javaee** e crie uma base de dados com o mesmo nome do usuário (**javaee**), com todas as permissões.

Na base **javaee**, crie uma tabela **pessoa** com os seguintes campos:

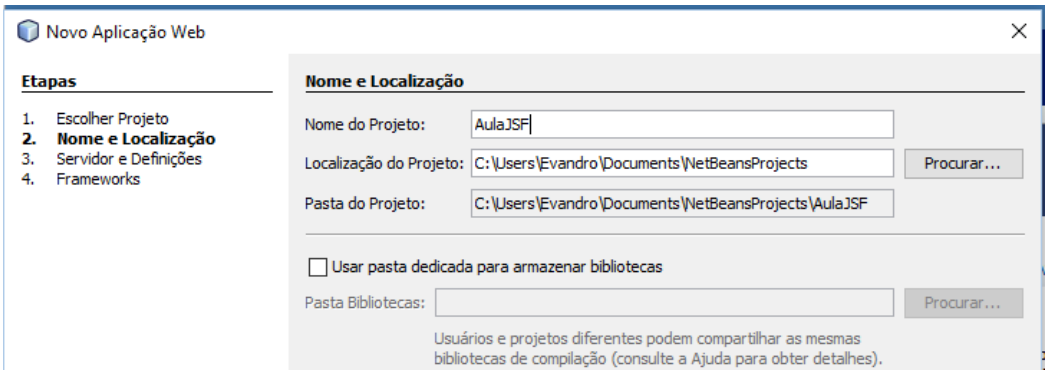
- cpf: char(15), chave primária
- nome: varchar(50)
- email: varchar(100)

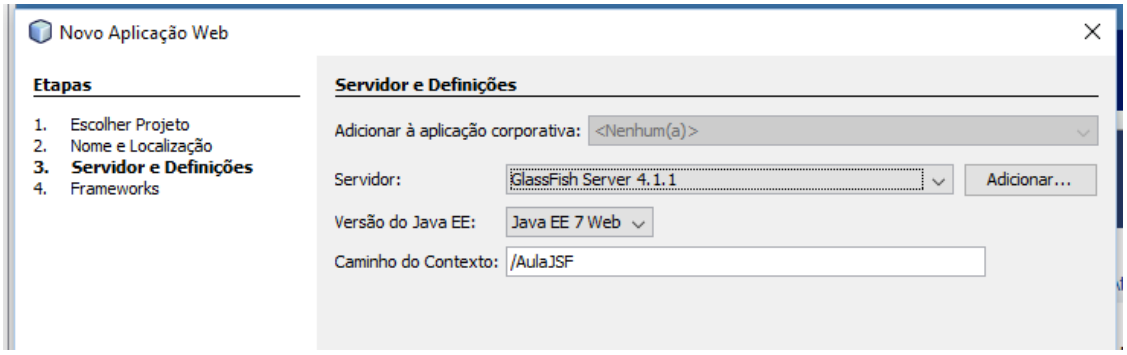
2. Criando uma aplicação com Java Server Faces

Vamos criar um novo projeto => Java Web => Aplicação Web

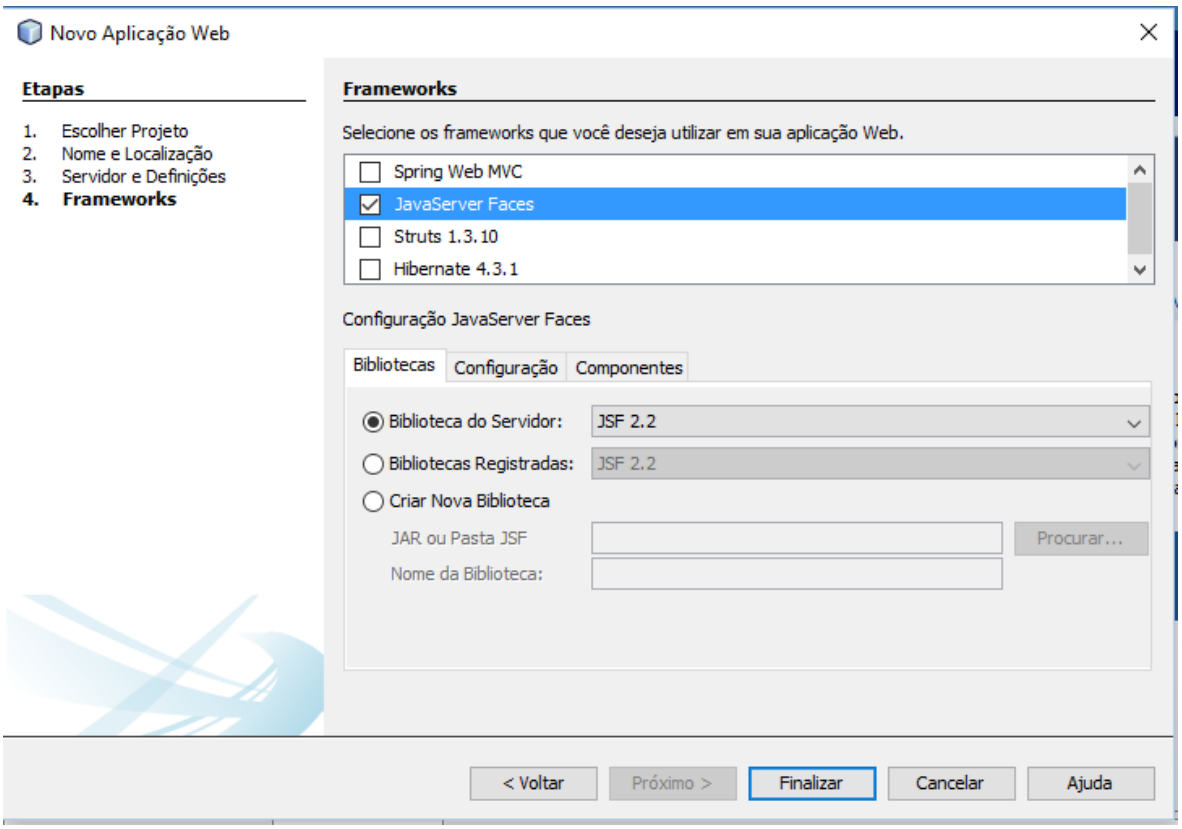


Dê o nome Aula JSF





Escolha o Framework JavaServer Faces



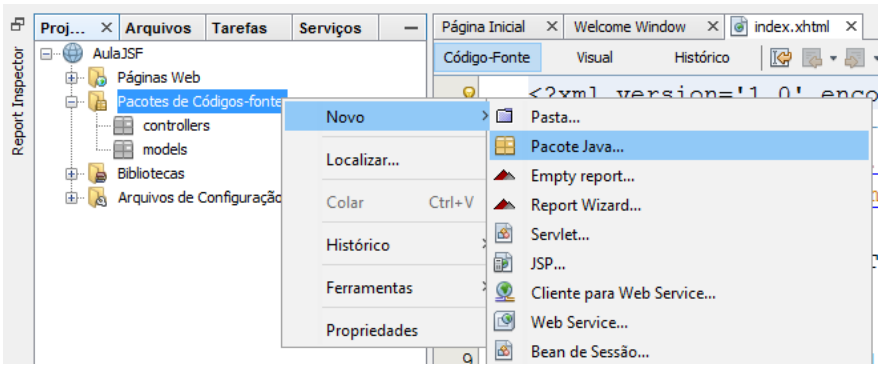
Se você mandar executar o projeto já temos um “Hello Facelets” pronto. Implante o projeto e teste.

Vamos criar um formulário simples, mas antes vamos falar um pouco mais sobre o JSF, ele é um framework possui uma arquitetura MVC(Model-View-Controller):

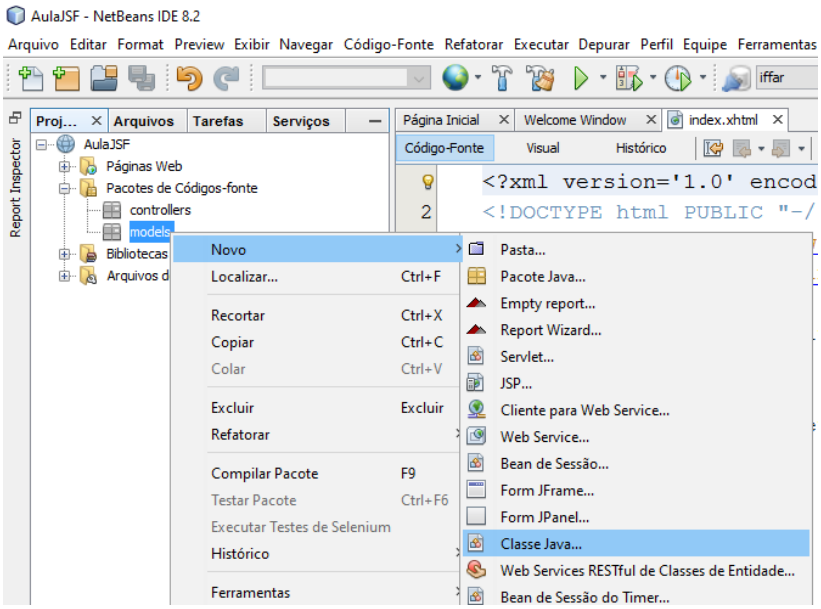
- Model será nossas classe
- View a telas de entrada e saída de dados
- Controller será o Managed Beans, ou seja Bens Gerenciados, onde vai conter nossa lógica de negócio ou programação.

Com i isso seu código fica organizado e de fácil manutenção e outros fatores, exemplo na sua View(tela) não tem código Java, somente seus componentes de tela e assim fica fácil para um design trabalhar com o visual.

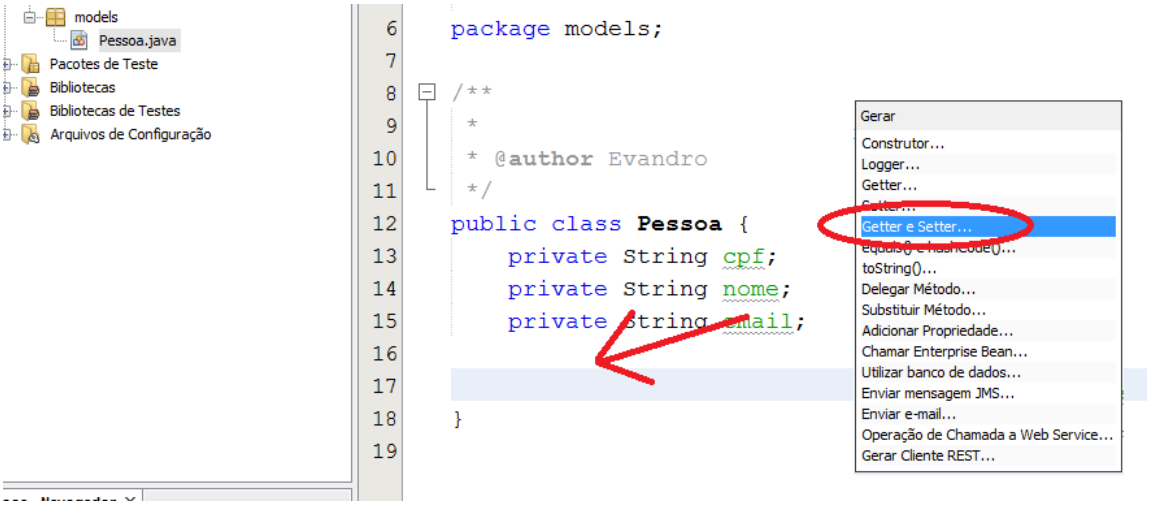
– Crie um pacote models e um controllers, para suas classes e os Managed Beans assim fica melhor visualizar seus arquivos.



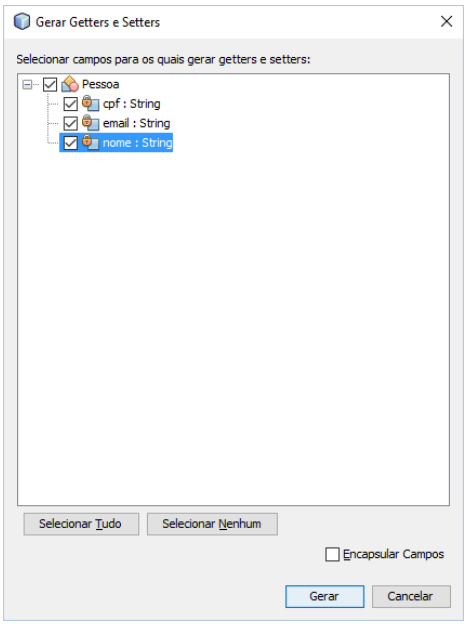
– Crie uma classe Java com o nome **Pessoa** dentro do pacote models, com os atributos: cpf, nome, email



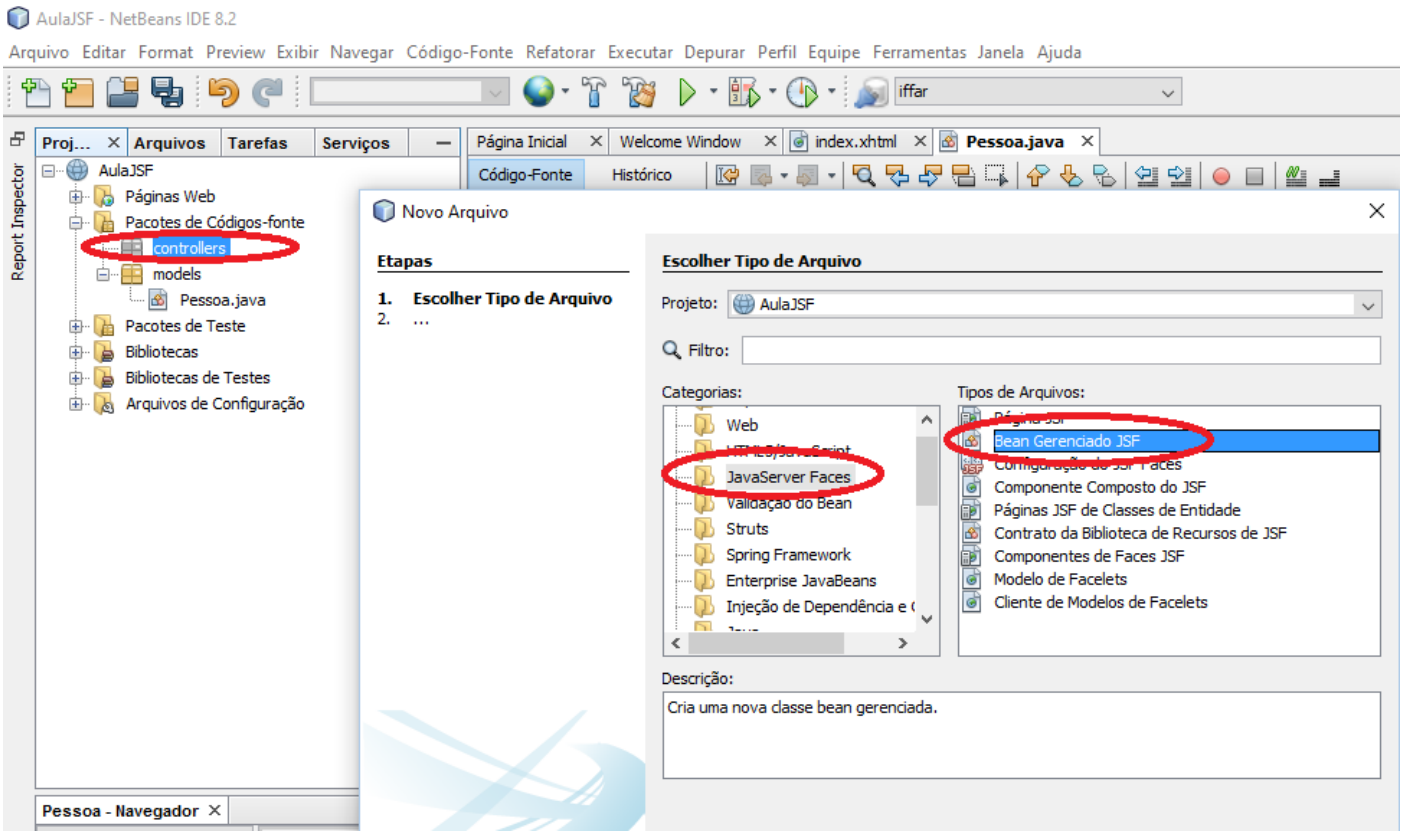
Clique com o botão direito do mouse dentro do código da classe, antes da última chave e escolha Getter e Setter:



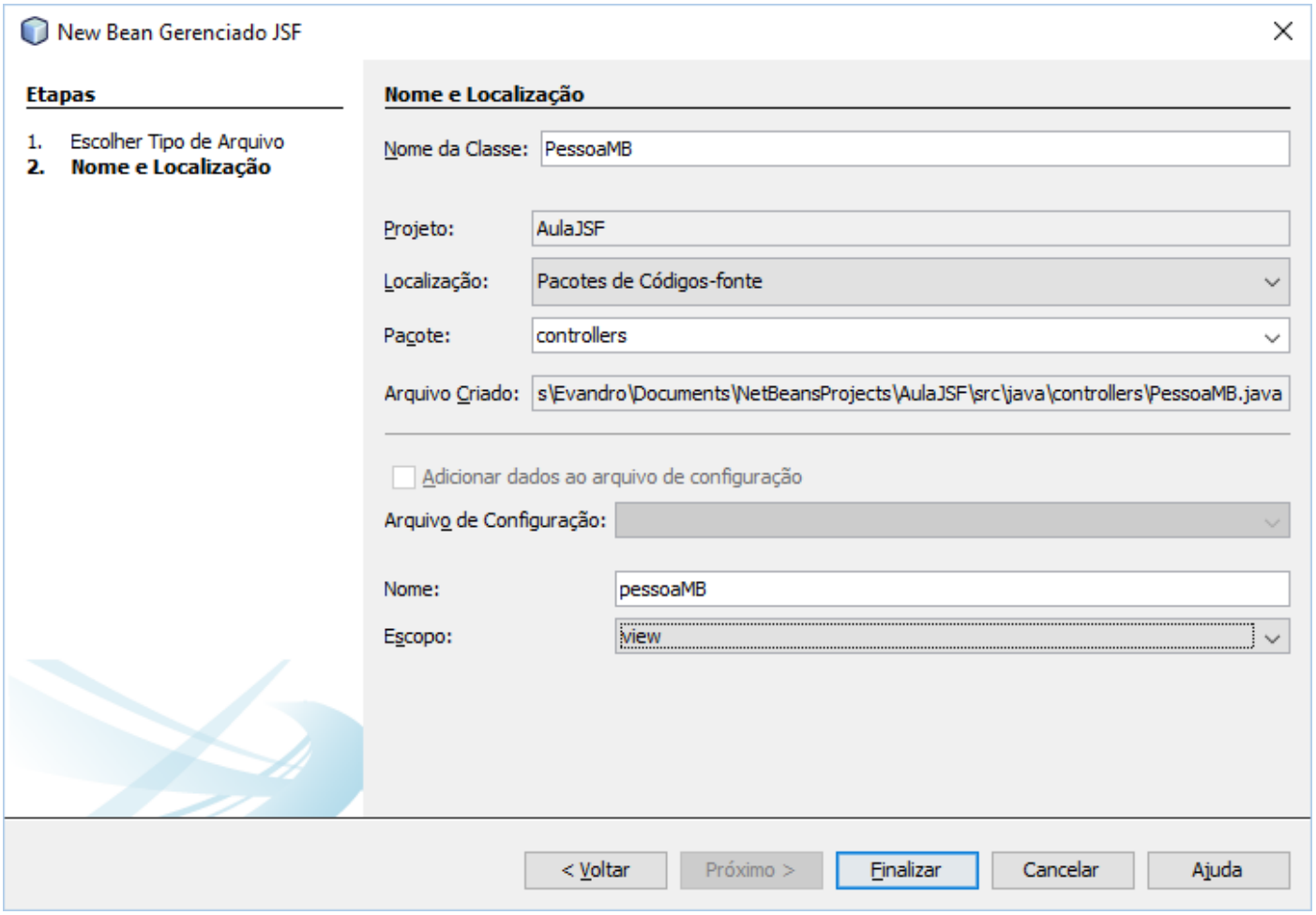
Marque todos os campos e clique em gerar



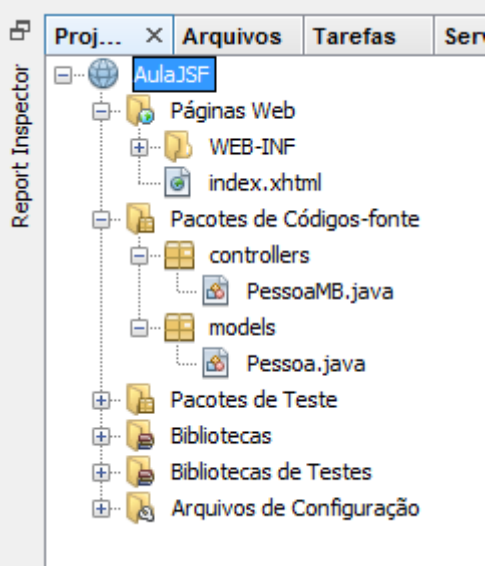
– Crie um Bean Gerenciado JSF no pacote controllers. Novo => Outros => Java Server Faces -> Bean Gerenciado JSF



– Vamos dar o nome de PessoaMB, escolha o escopo “View”



– Seu projeto desse estar com essa aparência agora, temos o index.xhtml que foi gerado ao criar o projeto, temos nossa classe modelo Pessoa e o controller PessoaMB.



– Abra o index.xhtml e vamos criar nosso formulário:

Código Index.xhtml:

```
<?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://java.sun.com/jsf/html"
 xmlns:f="http://java.sun.com/jsf/core">
 <h:head>
 <title>Cadastro</title>
 </h:head>
 <h:body>
 <h:form>
 <h:panelGrid columns="3">
 <h:outputLabel value="CPF"/>
 <h:inputText id="cpf"  size="20" />
 <h:message for="cpf" />

 <h:outputLabel value="Nome"/>
 <h:inputText id="nome"  size="20"  >
 <h:message for="nome" />

 <h:outputLabel value="E-Mail"/>
 <h:inputText id="email"  size="20" />
 <h:message for="email" />

 <h:commandButton id="btnE" value="Enviar" />
 <h:commandButton id="btnL" value="Limpar"/>

 </h:panelGrid>
 </h:form>
 </h:body>
</html>
```

Para deixar mais organizado, crie a endentação pressionando Alt Shift F

– Abra seu controller PessoaMB, vamos implementar a interface Serializable para que o mesmo possa retornar objetos como as listas e não apenas tipos simples, e instancie um objeto **pessoa**, e uma lista do tipo **pessoa** para armazenar as pessoas. Gere os get e set de cada um deles.

Código PessoaMB.java:

```
package controllers;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.inject.Named;
import javax.faces.view.ViewScoped;
import models.Pessoa;

/**
 *
 * @author Evandro
 */
@Named(value = "pessoaMB")
@ViewScoped
public class PessoaMB implements Serializable {

    private Pessoa pessoa = new Pessoa();
    private List<Pessoa> pessoaLista = new ArrayList<Pessoa>();

    /**
     * Creates a new instance of PessoaMB
     */
    public PessoaMB() {
    }


    public Pessoa getPessoa() {
        return pessoa;
    }

    public void setPessoa(Pessoa pessoa) {
        this.pessoa = pessoa;
    }

    public List<Pessoa> getPessoaLista() {
        return pessoaLista;
    }

    public void setPessoaLista(List<Pessoa> pessoaLista) {
        this.pessoaLista = pessoaLista;
    }

}
```

– Na Index.xhtml e vamos adicionar os values nos inputs.

Agora que instanciamos a classe no Bean podemos acessar os atributos através dele.

Ex: inputText id=”teste” value=”#{seuBean.classe.atributo}” />

Altere o código do Index.xhtml para que fique assim:

```
<h:inputText id="cpf" value="#{pessoaMB.pessoa.cpf}" size="20" />
 <h:message for="cpf" />
```

```
 <h:outputLabel value="Nome"/>
 <h:inputText id="nome" value="#{pessoaMB.pessoa.nome}" size="20"  >
 <h:message for="nome" />

 <h:outputLabel value="E-Mail"/>
 <h:inputText id="email" value="#{pessoaMB.pessoa.email}" size="20" />
 <h:message for="email" />
```

– No PessoaMB e vamos criar nosso método para salvar essas pessoas.

Logo após os Getters e Setters adicone o método salvar.

```
public void salvarPessoa() {

//adicionando pessoas a lista
pessoaLista.add(pessoa);
//instanciado uma nova para ser cadastrada
pessoa = new Pessoa();

}
```

– Na Index.xhtml e vamos adicionar ação para o botão.

É o mesmo sentido do value, mas nesse caso é uma action="#{meuBean.meuMetodo}".

Código parte Index.xhtml:

```
    <h:commandButton id="btnE" value="Enviar" type="submit" action="#{pessoaMB.salvarPessoa}"/>
```

Para envio de formulário utiliza o "submit" que é defaut, e no caso do segundo que é para limpeza usamos o "reset" que tem a função de limpar o formulário.

– Nosso cadastro está pronto, agora vamos apresentar na tela esses dados que inserimos na lista de pessoas, para isso utilizaremos um dataTable, veja como:

Insira o código parte no Index.xhtml, bem no final, antes do </html>

```
<h:panelGrid>
 <h:dataTable id="tablePessoa" value="#{pessoaMB.pessoaLista}" var="p" title="Pessoas
Cadastradas" border="1" rows="10" >
 <h:column>
 <f:facet name="header">
 <h:outputText value="CPF" />
 </f:facet>
 <h:outputText value="#{p.cpf}"/>
 </h:column>
 <h:column>
 <f:facet name="header">
 <h:outputText value="Nome" />
 </f:facet>
 <h:outputText value="#{p.nome}"/>
 </h:column>
 <h:column>
 <f:facet name="header">
 <h:outputText value="E-Mail" />
 </f:facet>
 <h:outputText value="#{p.email}"/>
 </h:column>
 </h:dataTable>
 </h:panelGrid>
```

Veja que no value do dataTable utilizei a lista que criamos no bean, value=”#{pessoaMB.pessoaLista}” e guardei numa variável var=”p” e através dela acesso os atributos específicos que estão armazenados na lista de pessoas.

Está pronto agora podemos visualizar o conteúdo que foi inserido na lista, podemos adicionar uma validação simples só para dar um “UP” no nosso formulário e mostrar a finalidade daquele , e que não é tão complicado em alguns casos validar no JSF.

No inputText /> que quiser que seja um campo obrigatório coloque o atributo required=”true” que torna o campo obrigatório e requiredeMessage=”Campo obrigatório” que é a mensagem que será exibida quando não satisfazer a condição, caso esqueça verá uma mensagem default do JSF. Ex:

Código validando com required:

```
<h:outputLabel value="Nome:"/>
<h:inputText id="nome" value="#{pessoaMB.pessoa.nome}" size="20"  required="true"
requiredMessage="Campo Obrigatório"/>
<h:message for="nome" />
```

O arquivo index.xhtml deve ter ficado assim:

```
<?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://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>Cadastro</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="3">
                <h:outputLabel value="CPF"/>
                <h:inputText id="cpf" value="#{pessoaMB.pessoa.cpf}" size="20" />
                <h:message for="cpf" />

                <h:outputLabel value="Nome"/>
                <h:inputText id="nome" value="#{pessoaMB.pessoa.nome}" size="20" required="true"
requiredMessage="Campo Obrigatório"/>
                    <h:message for="nome" />

                    <h:outputLabel value="E-Mail"/>
                    <h:inputText id="email" value="#{pessoaMB.pessoa.email}" size="20" />
                    <h:message for="email" />

                    <h:commandButton id="btnE" value="Enviar" type="submit"
action="#{pessoaMB.salvarPessoa}"/>
                    <h:commandButton id="btnL" value="Limpar" type="reset"/>

            </h:panelGrid>
        </h:form>
    </h:body>


    <h:panelGrid>
        <h:dataTable id="tablePessoa" value="#{pessoaMB.pessoaLista}" var="p" title="Pessoas
Cadastradas" border="1" rows="10" >
            <h:column>
                <f:facet name="header">
                    <h:outputText value="CPF" />
                </f:facet>
                <h:outputText value="#{p.cpf}"/>
            </h:column>
            <h:column>
                <f:facet name="header">
                    <h:outputText value="Nome" />
```

```
                </f:facet>
                <h:outputText value="#{p.nome}"/>
            </h:column>
            <h:column>
                <f:facet name="header">
                    <h:outputText value="E-Mail" />
                </f:facet>
                <h:outputText value="#{p.email}"/>
            </h:column>
        </h:dataTable>
    </h:panelGrid>
</html>
```

Implante e teste a aplicação. Veja se está fazendo os cadastros.

Abra em outro browser ou feche o browser e abra a aplicação novamente. Os dados continuam cadastrados?

Pois é... não fizemos a persistência dos dados... para isso teríamos que usar JDBC ou JPA...

Vamos alterar para usar JDBC que já estamos acostumados...

Onde temos que mudar? Quais arquivos? Agora é simples... como fizemos no modelo MVC, só precisamos alterar o controller... vamos lá...

Altere no **PessoaMB** os seguintes métodos para que fiquem assim:
inclua a importação import.java.sql.* para facilitar...

```
import java.sql.*;

public void salvarPessoa() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection("jdbc:mysql://localhost/javaee?user=javaee&password=javaee");
            PreparedStatement pstmt = con.prepareStatement(" INSERT INTO pessoa (cpf, nome,
email) VALUES( ?, ?, ?)");
            pstmt.setString(1, pessoa.getCpf());
            pstmt.setString(2, pessoa.getNome());
            pstmt.setString(3, pessoa.getEmail());
            pstmt.executeUpdate();
        } catch (SQLException | ClassNotFoundException err) {
            System.out.println("Erro inserindo pessoa." + err.getMessage());
        }
        pessoa = new Pessoa();
    }


public List<Pessoa> getPessoaLista() {
        pessoaLista = new ArrayList<Pessoa>();
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection("jdbc:mysql://localhost/javaee?user=javaee&password=javaee");
            PreparedStatement pstmt = con.prepareStatement("select * from pessoa order by
nome");
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()) {
                Pessoa p = new Pessoa();
                p.setCpf(rs.getString("cpf"));
                p.setNome(rs.getString("nome"));
                p.setEmail(rs.getString("email"));
                pessoaLista.add(p);
            }
        } catch (SQLException | ClassNotFoundException err) {
            System.out.println("Erro listando pessoa.");
        }
        return pessoaLista;
    }
```

Pronto!!!!

Fizemos a persistência dos dados usando JDBC..

Implante e teste...