Aulas 05 – 08/jun.

JavaEE6 – Conhecendo o JSF – Java Server Faces.

Como vimos anteriormente, a versão JavaEE5, é baseada no uso de servlets e JSP. Para a versão do JavaEE6, é mais utilizado o conceito de JSF, ou Java Server faces. Elas são páginas com a extensão ".xHTML" que recebem tags especiais para utilização do código Java. Isso tudo com o auxilio do JavaBeans. Uma vantagens sobre as JSPs é que as JSF tem a sintaxe mais "natural" ao HTML, sendo colocado muito pouco código Java nelas (apenas algumas palavras), assim facilitando a divisão de tarefas entra web designers e programadores.

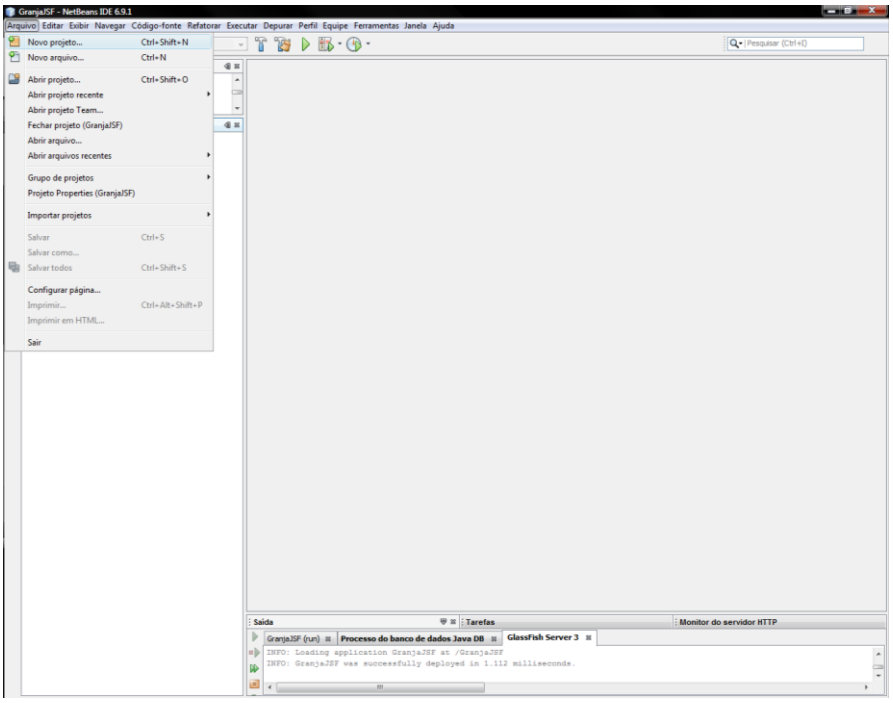
Vamos criar um exemplo baseado no CRUD em JSF feito no site "javasemcafe.blogspot.com" para entendermos como funciona o JSF.

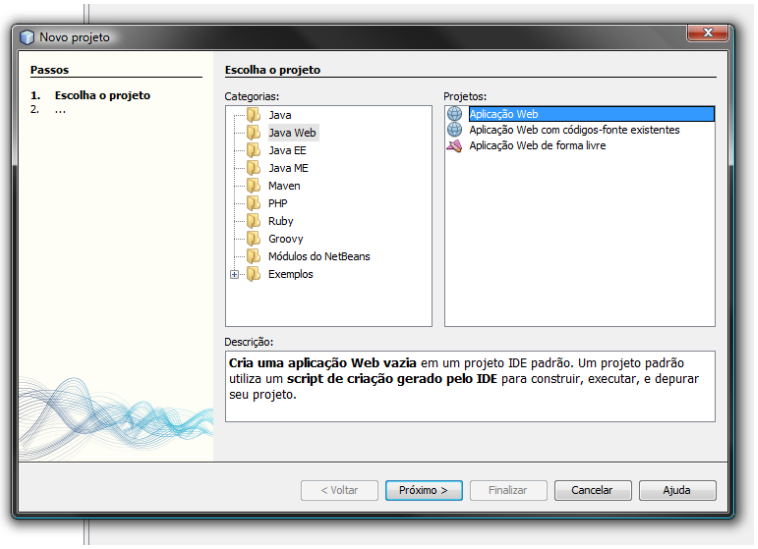
Inicialmente criaremos o banco de dados:

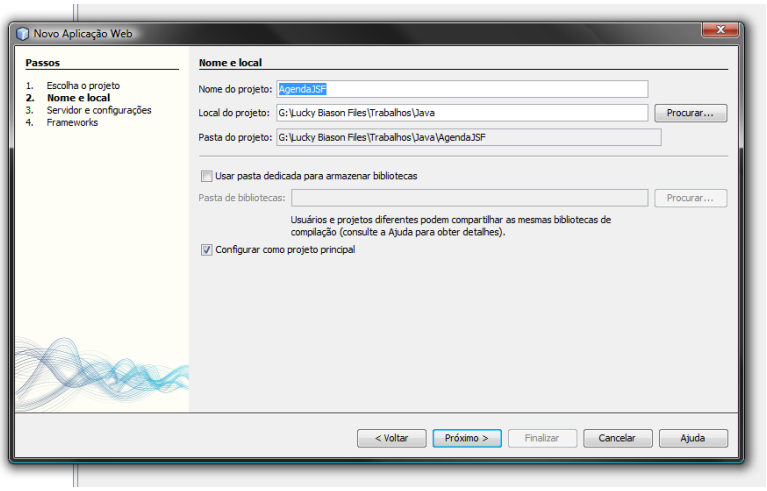
```
CREATE TABLE projetojsp.cliente (
  codigo INTEGER(11)  NOT NULL AUTO_INCREMENT,
  nome VARCHAR(255)  NOT NULL,
  telefone VARCHAR(30) ,
  PRIMARY KEY (codigo)
) ENGINE = InnoDB;
```

Criaremos uma mini Agenda de Contatos. Agora criaremos um novo projeto com nos moldes do jsf. (Atenção, faça EXATAMENTE como a "receita" a seguir).

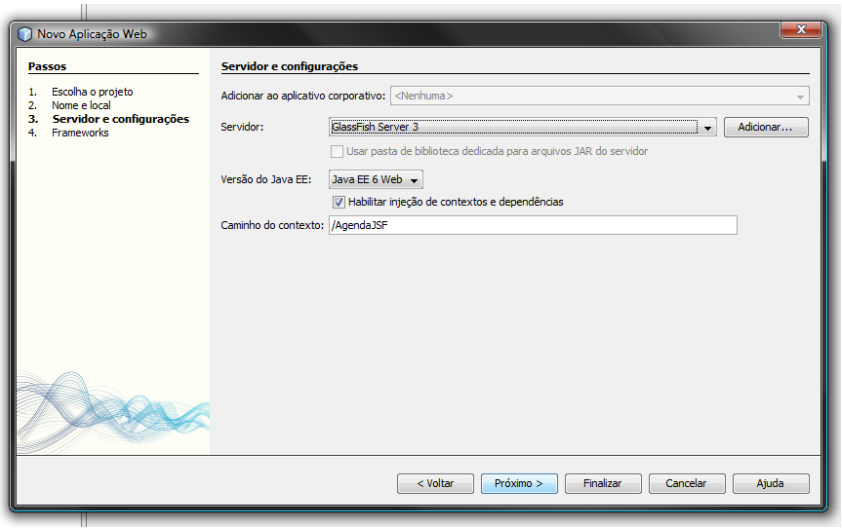
Primeiro, vamos criar um novo projeto chamado AgendaJSF:





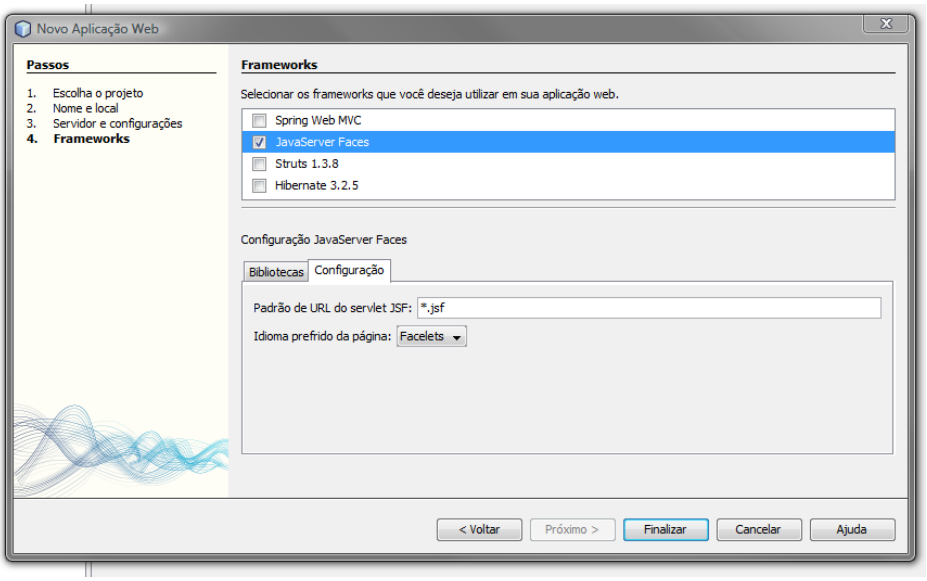


Na parte dos Servidores e Configurações escolha o GlassFish Server 3, que vem junto com o NetBeans 6.9.1 e a versão JavaEE6. Marque a caixa “Habilitar injeção de contextos e dependências”.

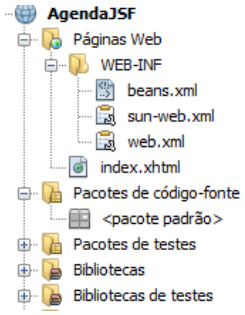


Na parte de Frameworks escolha o JavaServer Faces. Não se preocupe com a aba Biblioteca. Vamos para a aba configuração. No campo da URL digite *.jsf, e escolha a opção

Facelets.



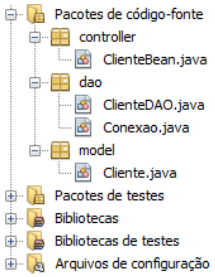
Finalize. A estrutura deve ficar assim:



Na página index.xhtml verificamos o uso na sintaxe do jsf.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.(
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html'
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Hello from Facelets
    </h:body>
</html>
```

Vamos criar três pacotes e algumas classes de acordo com a figura e os códigos a seguir.



Pacote dao, classe Conexao e classe Dao:

```
package dao;

import java.sql.*;

public class Conexao {

    public static Connection getConn(){
       try{
            Class.forName("com.mysql.jdbc.Driver");
            return (Connection)
                    DriverManager.getConnection(
                    "jdbc:mysql://localhost:3307/projetojsp", "root", "senha");
       } catch (ClassNotFoundException ex) {
            System.out.println(ex);
            return null;
        } catch (SQLException s) {
            System.out.println(s);
            return null;
        }
    }
```

```
package dao;

import java.sql.*;

public class Dao {

    public Connection getConnection(){
        return Conexao.getConn();
    }

    public Statement getStatement() throws SQLException{
        return getConnection().createStatement();
    }

    public PreparedStatement getStatement(String statement) throws SQLException{
        return getConnection().prepareStatement(statement);
    }

    public ResultSet executeQuery(String query, Object ... params) throws SQLException{
        PreparedStatement ps = getStatement(query);
        for(int i = 0; i< params.length; i++)
            ps.setObject(i+1, params[i]);
        return ps.executeQuery();
    }

    public int executeCommand(String query, Object... params) throws SQLException{
        PreparedStatement ps = getStatement(query);
        for(int i = 0; i< params.length; i++)
            ps.setObject(i+1, params[i]);
        int result = ps.executeUpdate();
        ps.close();
        return result;
    }

}
```

Pacote model, a classe que representa a tabela cliente, Cliente.java. Pacote Dao, vamos criar a manipulação da tabela cliente.

```java
package model;

public class Cliente {

    private int codigo;
    private String nome;
    private String telefone;

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }
}
```

```java
package dao;

import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import model.Cliente;

public class ClienteDAO extends Dao{

    public boolean inserir(Cliente cliente) {
        try {
            executeCommand("INSERT INTO cliente (nome, telefone) VALUES (?,?)",
                    cliente.getNome(),cliente.getTelefone());
            return true;
        } catch (Exception e) {return false;}
    }
    public boolean alterar(Cliente cliente) {
        try {
            executeCommand("UPDATE cliente SET nome = ?, telefone = ? WHERE codigo = ?",
                    cliente.getNome(),cliente.getTelefone(),cliente.getCodigo());
            return true;
        } catch (Exception e) {return false;}
    }
    public boolean remover(Cliente cliente) {
        try {
            executeCommand("DELETE FROM cliente WHERE codigo = ?",cliente.getCodigo());
            return true;
        } catch (Exception e) {return false;}
    }
    public List<Cliente> listar() {
        List<Cliente> clientes = new ArrayList<Cliente>();
        try {
            ResultSet rs = executeQuery("SELECT * FROM cliente ORDER BY nome");
            Cliente cliente;
            while (rs.next()) {
                cliente = new Cliente();
                cliente.setCodigo(rs.getInt("codigo"));
                cliente.setNome(rs.getString("nome"));
                cliente.setTelefone(rs.getString("telefone"));
                clientes.add(cliente);
            }
            return clientes;
        } catch (Exception e) {return null;}
    }
}
```

A ultima classe necessária é a ClienteBean. As classes Beans obedecem algumas regras. Sempre procure construir suas Beans nos moldes dados. Pois internamente essa classe será vincada com a página xHTML. Essa classe é obrigatória para o JSF.

```java
import dao.ClienteDAO;
import java.io.Serializable;
import java.util.List;
import javax.enterprise.context.SessionScoped;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;
import javax.inject.Named;
import model.Cliente;
```

```
@Named
@SessionScoped
public class ClienteBean implements Serializable {

    private ClienteDAO clienteDAO = new ClienteDAO();
    private Cliente cliente = new Cliente();
    private DataModel<Cliente> clientes;

    public void novo() {
        cliente = new Cliente();
    }
    public String inserir() {
        return (clienteDAO.inserir(cliente)) ? "clientes" : "falhou";
    }
    public void selecionar() {
        cliente = clientes.getRowData();
    }
    public String alterar() {
        return (clienteDAO.alterar(cliente))? "clientes" : "falhou";
    }
    public String remover() {
        return (clienteDAO.remover(cliente))? "clientes" : "falhou";
    }
    public Cliente getCliente() {
        return cliente;
    }
    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }
    public DataModel<Cliente> getClientes() {
        List<Cliente> clienteList = new ClienteDAO().listar();
        clientes = new ListDataModel<Cliente>(clienteList);
        return clientes;
    }
    public void setClientes(DataModel<Cliente> clientes) {
        this.clientes = clientes;
    }
}
```

Nos métodos inserir, alterar e remover repare que uma String é passada SEMPRE como valor de retorno. Essa String representa o nome de uma pagina para a qual, após o código ser executado, o usuário deverá ser direcionado.

O método getClientes, recebe do ClienteDao, uma List contendo os clientes no banco. E deve retornar uma ListDataModel, para ser usado com uma tag especial do Jsf que constrói tabelas dinamicamente.

Por ultimo, as páginas:

Index.xhtml, vemos aqui uma tag chamada commandButton, ela cria um botão para uma determinada página, assim como os type=”button”.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <h:commandButton action="clientes" value="Clientes"/>
        </h:form>
    </h:body>
</html>
```

Clientes.xhtml, vemos muitos novos comandos. O principal é o dataTable. Ele recebe um ListDataModel do método getClientes e cria uma tabela dinamicamente sem a necessidade do uso de um comando For ou criação da tag <table>. A cada coluna da tabela usamos o comando column, dentro dele, definimos o que é cabeçalho, usando o comando facet junto com o parâmetro name=header e o que será o espaço para os dados, usando outro comando chamado outputtext, que serve para mostrar texto na página. Repare que o uso de “código Java” se aplica as expressões javabeans : #{...} e apenas isso. As regras para o uso dos métodos é a seguinte: aqueles que forem métodos get e set, não necessitam das palavras get e set; o uso de parênteses é vetado, ou seja, métodos que precisem de parâmetros não podem ser chamados. Para entrada de dados o objeto cliente criado na classe beans deve ser vinculado com os campos inputtext com o auxilio do parâmetro value, o que é visto na página novo.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.
<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>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <h:commandButton action="novo" actionListener="#{clienteBean.novo}" value="Novo"/>
            <h:dataTable value="#{clienteBean.clientes}" var="c">
                <h:column>
                    <f:facet name="header"><h:outputText value="Nome"/></f:facet>
                    <h:outputText value="#{c.nome}"/>
                </h:column>
                <h:column>
                    <f:facet name="header"><h:outputText value="Telefone"/></f:facet>
                    <h:outputText value="#{c.telefone}"/>
                </h:column>
                <h:column>
                    <f:facet name="header"><h:outputText value="Ações"/></f:facet>
                    <h:commandButton action="alterar" actionListener="#{clienteBean.selecionar}" value="Alterar"/>
                    <h:commandButton action="remover" actionListener="#{clienteBean.selecionar}" value="Remover"/>
                </h:column>
            </h:dataTable>
            <h:commandButton action="index" value="Voltar"/>
        </h:form>
    </h:body>
</html>
```

Novo.xHTML

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xh
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputText value="Nome"/>
                <h:inputText value="#{clienteBean.cliente.nome}"/>
                <h:outputText value="Telefone"/>
                <h:inputText value="#{clienteBean.cliente.telefone}"/>
                <h:commandButton action="#{clienteBean.inserir}" value="Inserir"/>
                <h:commandButton action="clientes" immediate="True" value="Cancelar"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

Alterar.xhtml, quase não muda em relação a página anterior.

```
<?xml version='1.0' encoding='UTF-8' ?>
<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>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputText value="Nome"/>
                <h:inputText value="#{clienteBean.cliente.nome}"/>
                <h:outputText value="Telefone"/>
                <h:inputText value="#{clienteBean.cliente.telefone}"/>
                <h:commandButton action="#{clienteBean.alterar}" value="Alterar"/>
                <h:commandButton action="Clientes" immediate="True" value="Cancelar"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

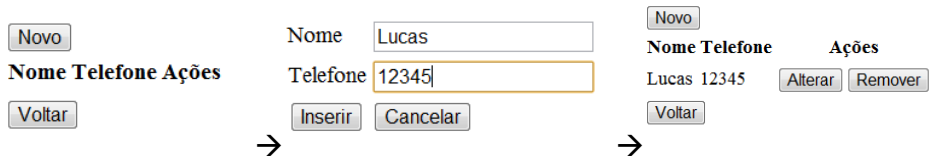
Remover.xHTML

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/
<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>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <h:outputText value="Deseja remover o cliente #{clienteBean.cliente.nome}?"/>
            <h:panelGrid columns="2">
                <h:commandButton action="#{clienteBean.remover}" value="Remover"/>
                <h:commandButton action="clientes" immediate="True" value="Cancelar"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

Falhou.xHTML, será utilizada caso ocorra algum erro com os comandos SQL

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <h:outputText value="Ocorreu um erro, tente novamente"/>
            <h:commandButton action="index" value="Tentar novamente"/>
        </h:form>
    </h:body>
</html>
```

Veja um exemplo de utilização:



Exercícios Complementares de Fixação

1) Refaça TODOS os exercícios feitos durante o curso.

Exercícios de Pesquisa

1) Explique sobre JSF e JavaEE6.
2) Faça uma tabela comparativa sobre os recursos do JSP e JSF. Quais as vantagens e desvantagens de cada tecnologia.
3) Pesquise sobre JavaBeans e Beans Validator para validar os formulários em JSF.
4) Pesquise sobre os frameworks JSF (PrimeFaces, RichFaces e ICEFaces).