

LENGUAJE DE PROGRAMACION II

Docente: Ing. Díaz Leva Teodoro

Tema: Java Server Faces(JSF)

Introducción a Java Server Faces

¿Qué es JSF?

JavaServer Faces (JSF) es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

JSF es un framework MVC (Modelo-Vista-Controlador) basado en el API de Servlets que proporciona un conjunto de componentes en forma de etiquetas definidas en páginas XHTML mediante el framework Facelets. Facelets se define en la especificación 2 de JSF como un elemento fundamental de JSF que proporciona características de plantillas y de creación de componentes compuestos. Antes de la especificación actual se utilizaba JSP para componer las páginas JSF.

JSF utiliza las páginas Facelets como vista, objetos Javabean como modelos y métodos de esos objetos como controladores. El servlet FacesServlet realiza toda la tediosa tarea de procesar las peticiones HTTP, obtener los datos de entrada, validarlos y convertirlos, colocarlos en los objetos del modelo, invocar las acciones del controlador y renderizar la respuesta utilizando el árbol de componentes.

JSF proporciona las siguientes características destacables:

- Definición de las interfaces de usuario mediante vistas que agrupan componentes gráficos.
- Conexión de los componentes gráficos con los datos de la aplicación mediante los denominados beans gestionados.
- Conversión de datos y validación automática de la entrada del usuario.
- Navegación entre vistas.
- Internacionalización
- A partir de la especificación 2.0 un modelo estándar de comunicación Ajax entre la vista y el servidor.

Tal y como hemos comentado, JSF se ejecuta sobre la tecnología de Servlets y no requiere ningún servicio adicional, por lo que para ejecutar aplicaciones JSF sólo necesitamos un contenedor de servlets tipo Tomcat o Jetty.

Para entender el funcionamiento de JSF es interesante compararlo con JSP. Recordemos que una página JSP contiene código HTML con etiquetas especiales y código Java. La página se procesa en una pasada de arriba hacia abajo y se convierte en un servlet. Los elementos JSP se procesan en el orden en que aparecen y se transforman en código Java que se incluye en el servlet. Una vez realizada la conversión, las peticiones de los usuarios a la página provocan la ejecución del servlet.

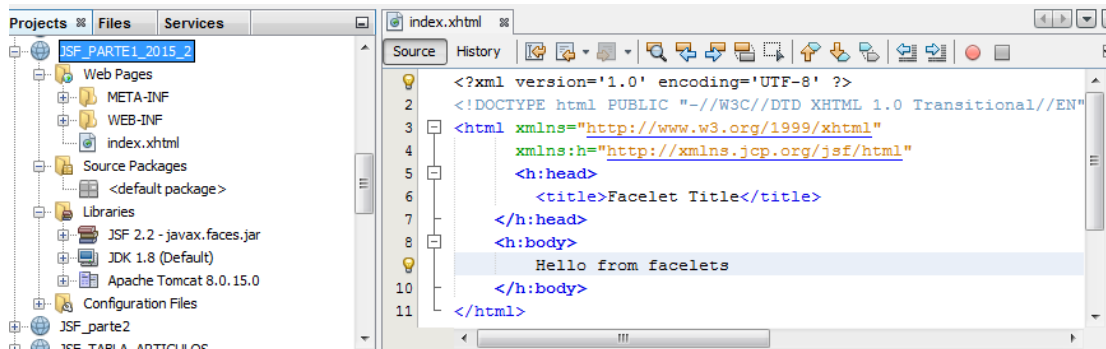
En JSF el funcionamiento es distinto. Una página JSF también contiene etiquetas especiales y código HTML, pero su procesamiento es mucho más complicado. La diferencia fundamental con JSP es el resultado del procesamiento interno, en el servidor, de la página cuando se realiza la petición. En JSP la página se procesa y se transforma en un servlet. En JSF, sin embargo, el resultado del procesamiento es un árbol de componentes, objetos Java instanciados el servidor, que son los que posteriormente se encargan de generar el HTML.

Laboratorio 1

Objetivo:

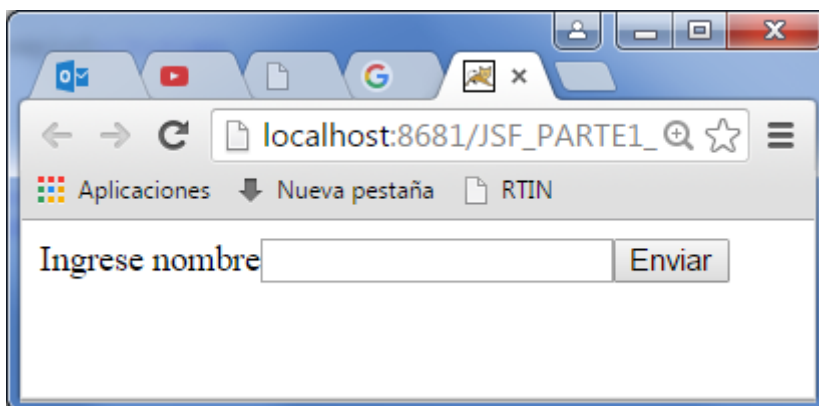
Desarrollar aplicaciones web básicas utilizando el framework Java Server Faces (JSF) para entender sus elementos y funcionamiento.

1. Crear proyecto “**JSF_PARTE1_2015_2**”, que podría tener otro nombre. En el proceso de creación seleccionar el Framework Java Server Faces.



2. Definición de vistas JSF:

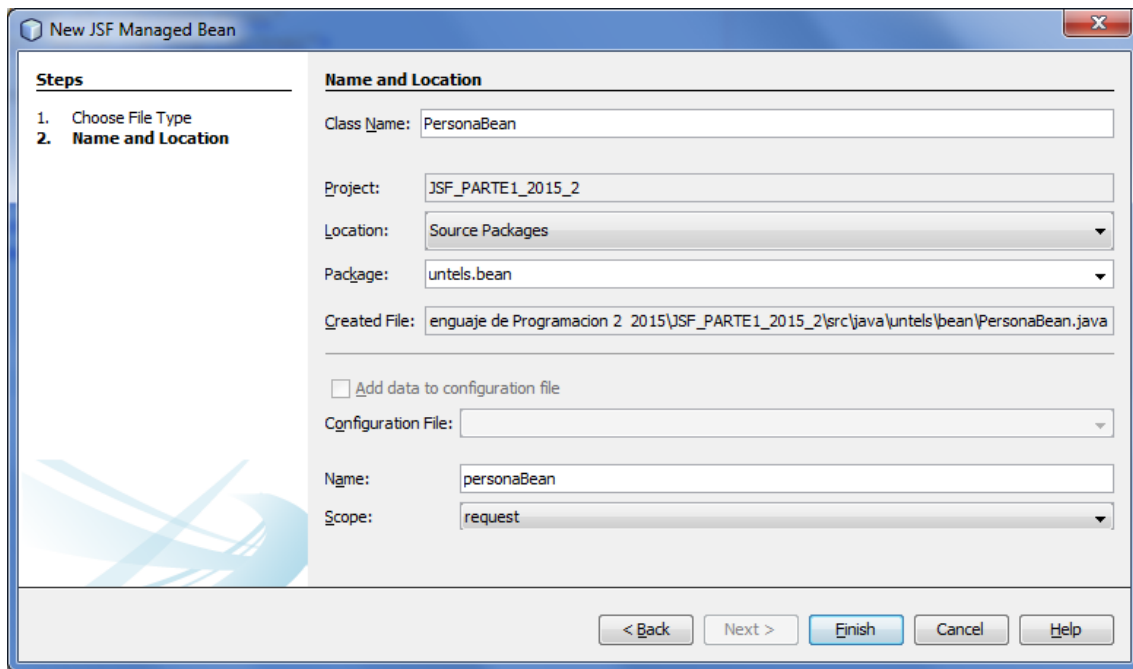
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>JSF-Ejemplo1</title>
  </h:head>
  <h:body>
    <h:form>
      <h:outputLabel value="Ingrese nombre"/>
      <h:inputText />
      <h:commandButton value="Enviar"/>
    </h:form>
  </h:body>
</html>
```



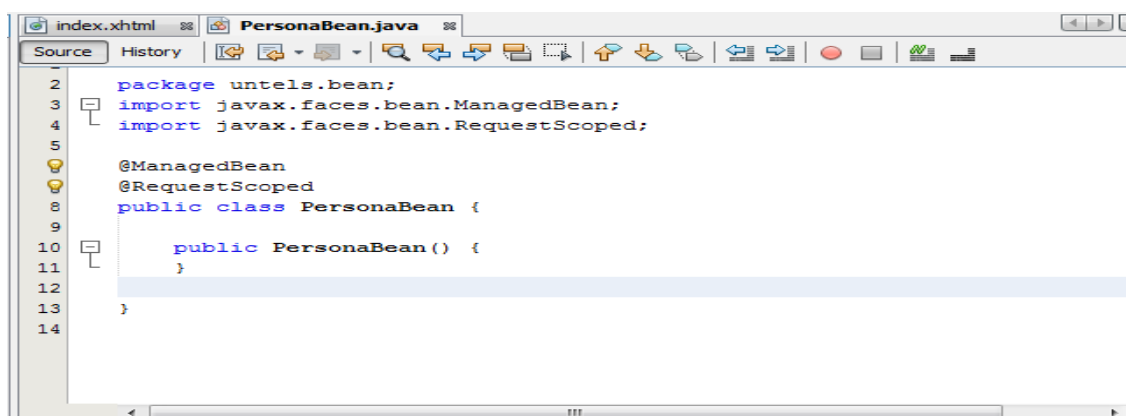
Explicación

Los componentes JSF son una etiqueta (`h:outputLabel`), una caja de texto (`h:inputText`) y para lanzar la acción de enviar el nombre a la capa de negocio se utiliza el componente botón (`h:commandButton`). Todos los componentes se encuentran dentro de un `h:form` que se traducirá a un formulario HTML.

3. Creamos un **bean gestionado** "PersonaBean" para almacenar el nombre ingresado en la vista (Establecer comunicación entre la vista y el bean gestionado). Clic derecho sobre Source Packages -> new > JSF Managed Bean. Ingrese nombre del paquete **untels.bean** y nombre de la clase **PersonaBean**



La clase **PersonaBean** tendrá la siguiente estructura:



Ahora asignamos las propiedades de nombre y mensaje a la clase **PersonaBean** y sus respectivos métodos de retorno y actualización (Getter and Setter):

```
package untels.bean;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class PersonaBean {
    //propiedades

    private String nombre;
    private String mensaje;
    //constructor

    public String getNombre() {
        return nombre;
    }
    //metodos de retorno y actualizacion de nombre y mensaje

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getMensaje() {
        return mensaje;
    }

    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }

    public PersonaBean() {
    }

    //método saludo para asignar valor a la propiedad mensaje
    public void saludo() {
        mensaje="Hola "+ nombre+ " bienvenido(a) al desarrollo de
aplicaciones web con JSF";
    }
}
```

Explicación sobre el ámbito de los beans gestionados

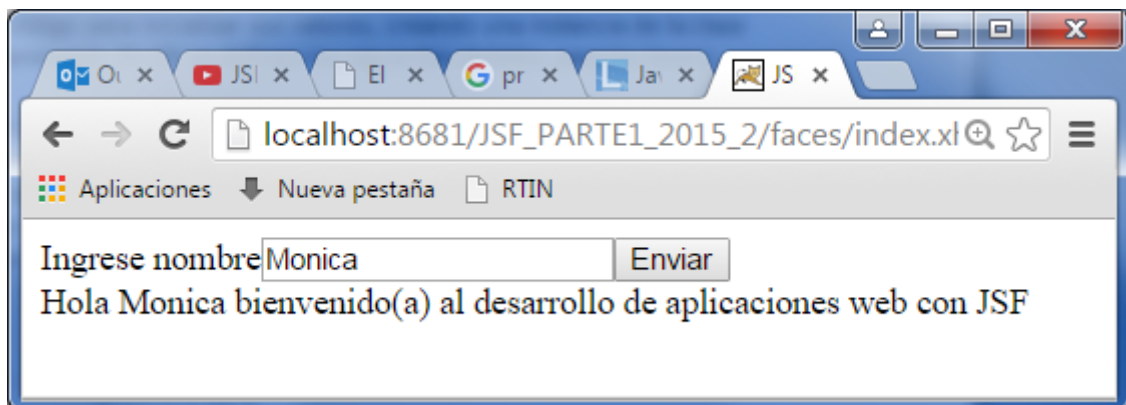
El ámbito de los beans determina su ciclo de vida: cuándo se crean y destruyen instancias del bean y cuándo se asocian a las páginas JSF. Es muy importante definir correctamente el ámbito de un bean, porque en el momento de su creación se realiza su inicialización. JSF llama al método constructor de la clase, donde habremos colocado el código para inicializar sus valores, creando una instancia de la clase que pasará a ser gestionada por el framework. En JSF se definen los siguientes ámbitos para los beans:

- **Petición:** Se define con el valor request con la anotación @RequestScoped en la clase. El bean se asocia a una petición HTTP. Cada nueva petición (cuando desde el navegador se abre una página por primera vez una página o se recarga) crea un nuevo bean y lo asocia con la página. Dada su corta vida, se recomienda usar este ámbito para el paso de mensajes (bien sea de error o de estatus), o para cualquier otro tipo de dato que no sea necesario propagar a lo largo de la aplicación

- **Sesión:** Se define con la anotación `@SessionScoped` en la clase. Las sesiones se definen internamente con el API de Servlets. Una sesión está asociada con una visita desde un navegador. Cuando se visita la página por primera vez se inicia la sesión. Cualquier página que se abra dentro del mismo navegador comparte la sesión. La sesión mantiene el estado de los elementos de nuestra aplicación a lo largo de las distintas peticiones. Se implementa utilizando cookies o reescritura de URLs, con el parámetro `sessionid`. Una sesión no finaliza hasta que se invoca el método `invalidate` en el objeto `HttpSession`, o hasta que se produce un `timeout`.
 - **Aplicación:** Se define con la anotación `@ApplicationScoped`. Los beans con este ámbito viven asociados a la aplicación. Definen *singletons* que se crean e inicializa sólo una vez, al comienzo de la aplicación. Se suelen utilizar para guardar características comunes compartidas y utilizadas por el resto de beans de la aplicación.
 - **Vista (JSF2.0):** Se define con la anotación `@ViewScoped` en la clase. Un bean en este ámbito persistirá mientras se repinte la misma página (vista = página JSF), al navegar a otra página, el bean sale del ámbito. Es bastante útil para aplicaciones que usen Ajax en parte de sus páginas.
 - **Custom (@CustomScoped):** Un ámbito al fin y al cabo no es más que un mapa que enlaza nombres y objetos. Lo que distingue un ámbito de otro es el tiempo de vida de ese mapa. Los tiempos de vida de los ámbitos estándar de JSF (sesión, aplicación, vista y petición) son gestionados por la implementación de JSF. En JSF 2.0 podemos crear ámbitos personalizados, que son mapas cuyo ciclo de vida gestionamos nosotros. Para incluirlo en ese mapa, usaremos la anotación `@CustomScoped("#{expr}")`, donde `#{expr}` indica el mapa. Nuestra aplicación será la responsable de eliminar elementos de ese mapa.
 - **Conversación (@ConversationScoped)** - provee de persistencia de datos hasta que se llega a un objetivo específico, sin necesidad de mantenerlo durante toda la sesión. Está ligado a una ventana o pestaña concreta del navegador. Así, una sesión puede mantener varias conversaciones en distintas páginas. Es una característica propia de CDI, no de JSF.
4. La conexión entre las clases Java y las páginas JSF(vista) se realiza mediante el Lenguaje de Expresiones JSF (JSF EL), una versión avanzada del lenguaje de expresiones de JSP. Con este lenguaje, podemos definir conexiones (*bindings*) entre las propiedades de los beans y los valores de los componentes que se muestran o que introduce el usuario. En nuestro ejemplo conectamos el componente `h:inputText` con la propiedad nombre de clase `PersonaBean` y para asignar valor a la propiedad mensaje ejecutamos el método `saludo` a través del componente botón (`h:commandButton`). A través del componente `h:outputLabel` conectamos a la propiedad mensaje para mostrar su valor (mensaje de saludo):

```
<?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>JSF-Ejemplo1</title>
  </h:head>
  <h:body>
    <h:form>
      <h:outputLabel value="Ingreso nombre"/>
      <h:inputText value="#{personaBean.nombre}"/>
      <h:commandButton value="Enviar"
actionListener="#{personaBean.saludo()}" /><br><br>
      <h:outputLabel value="#{personaBean.mensaje}"/>
    </h:form>
  </h:body>
</html>
```

//-----Ejecución de la aplicación-----

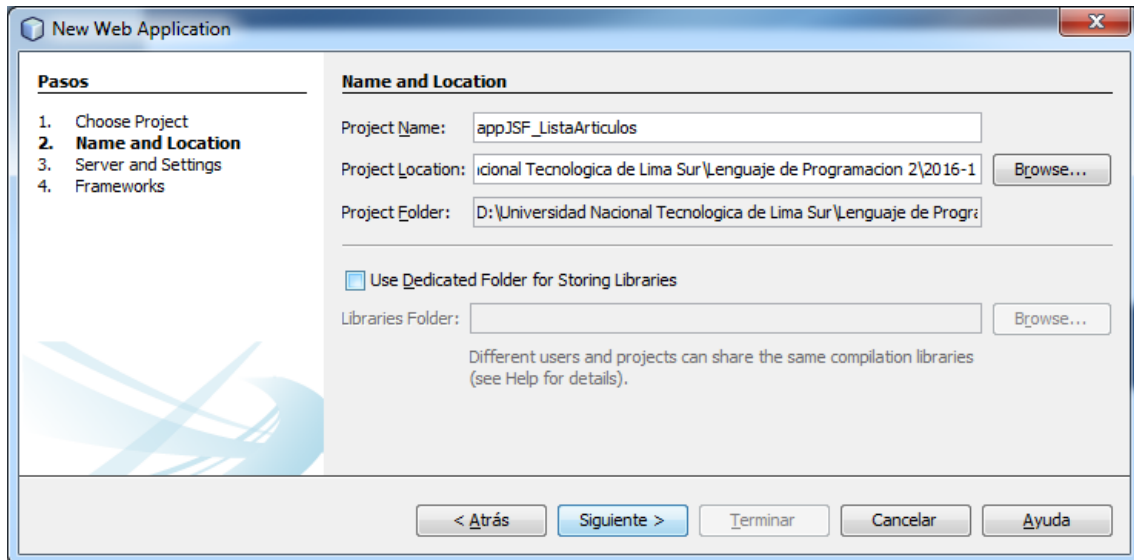


Laboratorio 2

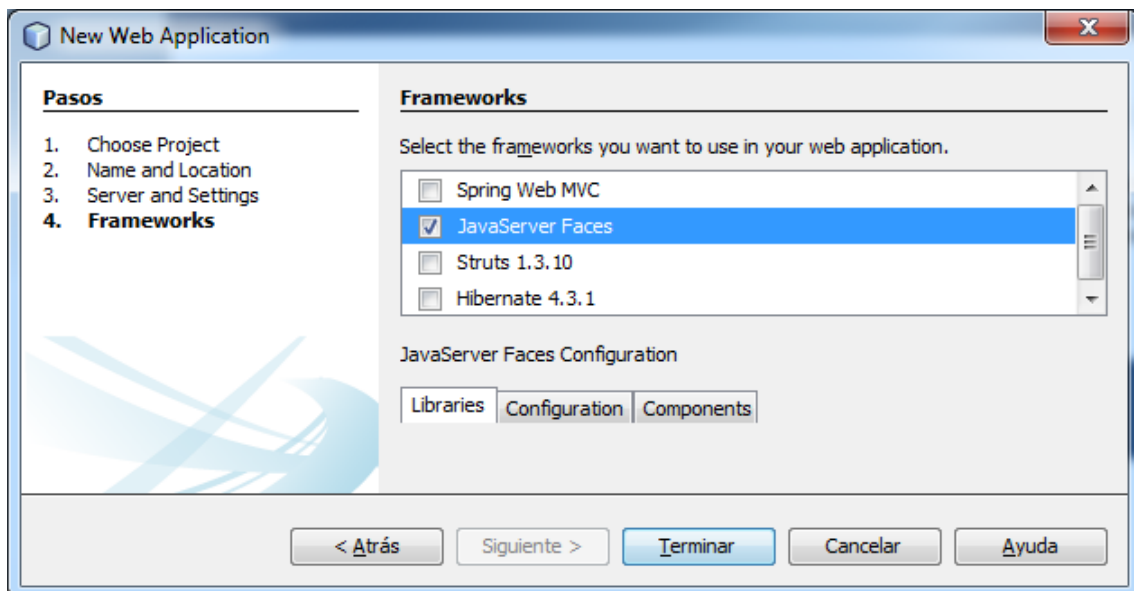
Objetivo:

Desarrollar aplicación web utilizando el framework Java Server Faces (JSF) para listar los artículos de la base de datos tienda en un dataTable de PrimerFaces.

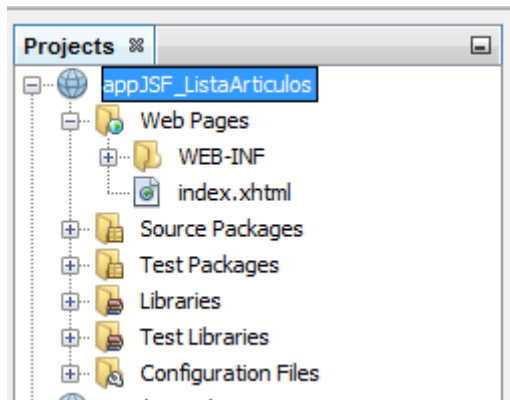
1. Crear proyecto web con el nombre de “appJSF_listaArticulos”:



2. Seleccione Framework Java server Faces:



Click en botón "Terminar" :



3. Crear paquete "tienda.conexion", en ella la clase **Conexion** :

```
package tienda.conexion;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class Conexion {

    protected Connection cn = null;
    protected String url = "jdbc:mysql://localhost/tienda";
    protected String usuario = "root";
    protected String clave = "1234";
    protected PreparedStatement stm;
    protected ResultSet rs;

    public Connection abrirConexion() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            cn = DriverManager.getConnection(url, usuario, clave);
        } catch (Exception ex) {
            System.out.println("cn:"+ex);
        }
        return cn;
    }

    protected void cerrar(Connection con) throws RuntimeException {
        try {
            if (con != null && !con.isClosed()) {
                con.close();
            }
        } catch (SQLException se) {
            System.err.println("Error: cerrarConexion: " + se);
        }
    }

    protected void cerrar(ResultSet rs) throws RuntimeException {
        try {
            if (rs != null) {
                rs.close();
            }
        }
    }
}
```



```
        } catch (SQLException se) {
            System.err.println("Error: cerrarResultSet: " + se);
        }
    }

    protected void cerrar(PreparedStatement stmt)
        throws RuntimeException {
        try {
            if (stmt != null) {
                stmt.close();
            }
        } catch (SQLException se) {
            System.err.println("Error: cerrarStatement: " + se);
        }
    }
}
```

4. Crear paquete **"tienda.modelo"**, en ella la clase **Articulo** que representa la estructura de la tabla **articulo** de la base de datos **tienda**:

```
package tienda.modelo;
public class Articulo {
    private int idarticulo;
    private int idcategoria;
    private String nombre;
    private String descripcion;
    private double precio;
    private String foto;

    public int getIdarticulo() {
        return idarticulo;
    }

    public void setIdarticulo(int idarticulo) {
        this.idarticulo = idarticulo;
    }

    public int getIdcategoria() {
        return idcategoria;
    }

    public void setIdcategoria(int idcategoria) {
        this.idcategoria = idcategoria;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }
}
```

```
public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
}

public double getPrecio() {
    return precio;
}

public void setPrecio(double precio) {
    this.precio = precio;
}

public String getFoto() {
    return foto;
}

public void setFoto(String foto) {
    this.foto = foto;
}
}
```

5. Crear paquete “**tienda.helper**”, en ella la interface **GenericCrud**, que tendrá las definiciones de los métodos comunes que implementaran las clases del paquete tienda.dao:

```
package tienda.helper;

import java.util.ArrayList;

public interface GenericCrud<T> {

    public ArrayList<T> list();

    public T get(T t);

    public void save(T t);

    public void update(T t);

    public void delete(T t);

}
```

6. Crear paquete “**tienda.dao**”, en ella la clase **Articulo**, que implementará los métodos definidos en la interface **GenericCrud**

```
package tienda.dao;
import java.sql.SQLException;
import java.util.ArrayList;
import tienda.conexion.Conexion;
import tienda.helper.GenericCrud;
import tienda.modelo.Articulo;
```

```
public class ArtículoDAO extends Conexion implements
GenericCrud<Articulo> {

    @Override
    public ArrayList<Articulo> list() {
        //arreglo de objetos tipo articulo
        ArrayList<Articulo> lista = new ArrayList<>();
        Articulo ar;
        //instrucción sql para extraer todos los articulos de la BD
        String sql = "select*from articulo";

        try {
            //abrir al conexion a la base de datos tienda
            cn = abrirConexion();
            //creamos un objeto PreparedStatement con la conexion a
            //la base de datos
            stm = cn.prepareStatement(sql);
            //ejecutar objeto PreparedStatement
            rs = stm.executeQuery();
            while (rs.next()) {
                //crear objeto ar
                ar = new Articulo();
                //encastmular datos en obejto ar
                ar.setIdarticulo(rs.getInt(1));
                ar.setIdcategoria(rs.getInt(2));
                ar.setNombre(rs.getString(3));
                ar.setDescripcion(rs.getString(4));
                ar.setPrecio(rs.getDouble(5));
                ar.setFoto(rs.getString(6));
                //asignar objeto ar arreglo de objeto lista
                lista.add(ar);
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            cerrar(cn);
            cerrar(stm);
            cerrar(rs);
        }
        return lista;
    }

    @Override
    public Articulo get(Articulo t) {
        Articulo ar = null;
        //instruccion sql para extraer todos los articulos de la BD
        String sql = "select*from articulo where idarticulo=?";
        //objeto preparastamet para ejecutar instruccion sql a traves
        //de su metodo
        //executequery y la conexion cn
        try {
            //abrir al conexion a la base de datos tienda
            cn = abrirConexion();
            //creamos un objeto PreparedStatement con la conexion a
            //la base de datos
            stm = cn.prepareStatement(sql);
            stm.setInt(1, t.getIdarticulo());
            //ejecutar objeto preparedstament
            rs = stm.executeQuery();
            //leer resulset
```

```
        if (rs.next()) {
            //crear objeto ar
            ar = new Artículo();
            //encapsular datos en oyecto ar
            ar.setIdarticulo(rs.getInt(1));
            ar.setIdcategoria(rs.getInt(2));
            ar.setNombre(rs.getString(3));
            ar.setDescripcion(rs.getString(4));
            ar.setPrecio(rs.getDouble(5));
            ar.setFoto(rs.getString(6));
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally {
        cerrar(cn);
        cerrar(stm);
        cerrar(rs);
    }
    return ar;
}

@Override
public void save(Artículo t) {
    String sql = "insert into
articulo(nombre,idcategoria,descripcion,precio) values(?,?,?,?)";
    try {
        //abrir al conexion a la base de datos tienda
        cn = abrirConexion();
        //creamos un objeto PreparedStatement con la conexion a la
        //base de datos
        stm = cn.prepareStatement(sql);
        stm.setString(1, t.getNombre());
        stm.setInt(2, t.getIdcategoria());
        stm.setString(3, t.getDescripcion());
        stm.setDouble(4, t.getPrecio());
        //ejecutar sentencia sql
        stm.executeUpdate();
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally {
        cerrar(cn);
        cerrar(stm);
        cerrar(rs);
    }
}

@Override
public void update(Artículo t) {
    String sql = "update articulo set
nombre=?,idcategoria=?,descripcion=?,precio=? where idarticulo=?";

    try {
        //abrir al conexion a la base de datos tienda
        cn = abrirConexion();
        //creamos un objeto PreparedStatement con la conexion a
        //la base de datos
        stm = cn.prepareStatement(sql);
        stm.setString(1, t.getNombre());
        stm.setInt(2, t.getIdcategoria());
        stm.setString(3, t.getDescripcion());
        stm.setDouble(4, t.getPrecio());
    }
```

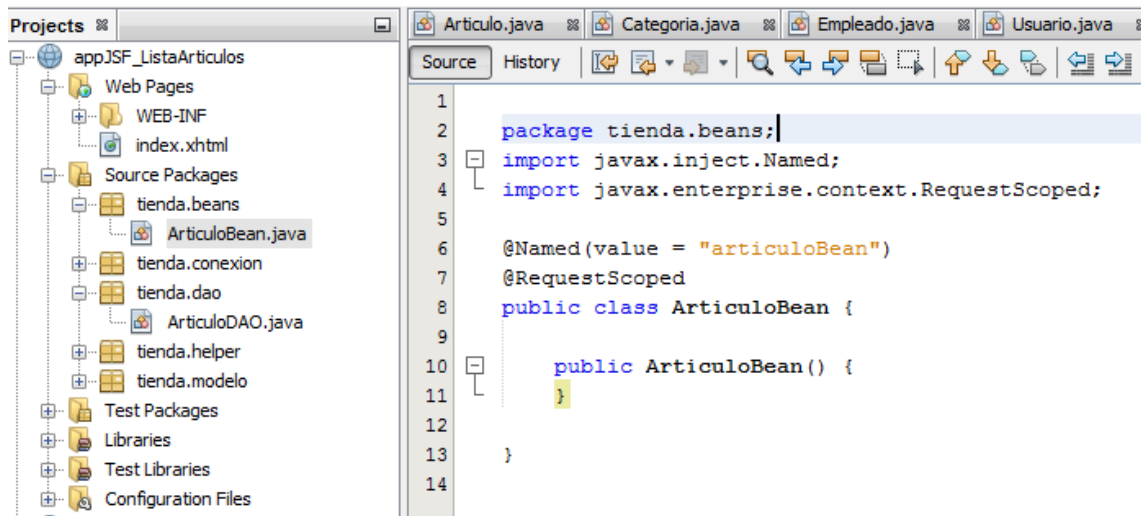
```
        stm.setInt(5, t.getIdarticulo());
        //ejecutar sentencia sql
        stm.executeUpdate();
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally {
        cerrar(cn);
        cerrar(stm);
        cerrar(rs);
    }
}

@Override
public void delete(Articulo t) {
    String sql = "delete from articulo where idarticulo=?";
    try {
        //abrir al conexion a la base de datos tienda
        cn = abrirConexion();
        //creamos un objeto PreparedStatement con la conexion a la
        //base de datos
        stm = cn.prepareStatement(sql);
        stm.setInt(1, t.getIdarticulo());
        //ejecutar sentencia sql
        stm.executeUpdate();
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally {
        cerrar(cn);
        cerrar(stm);
        cerrar(rs);
    }
}
}
```

7. Creamos un **bean gestionado** “articuloBean” para establecer comunicación entre la vista y el bean gestionado:

Name and Location	
Class Name:	ArticuloBean
Project:	appJSF_ListaArticulos
Location:	Source Packages
Package:	tienda.beans
Created File:	Programacion 2\2016-1\appJSF_ListaArticulos\src\java\tienda\beans\ArticuloBean.java
<input type="checkbox"/> Add data to configuration file	
Configuration File:	
Name:	articuloBean
Scope:	request

8. Al presionar el botón “Terminar”, se tiene el código básico del **articuloBean**:



Codificación en articuloBean

```
package tienda.beans;
import java.util.ArrayList;
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import tienda.dao.ArticuloDAO;
import tienda.modelo.Articulo;

@Named(value = "articuloBean")
@RequestScoped
public class ArticuloBean {

    public ArticuloBean() {
    }
    //objeto tipo Articulo para tener acceso a sus atributos
    private Articulo articulo = new Articulo();
    //variable tipo lista de articulos
    private ArrayList<Articulo> lstArticulos;

    public ArrayList<Articulo> getLstArticulos() {
        return lstArticulos;
    }

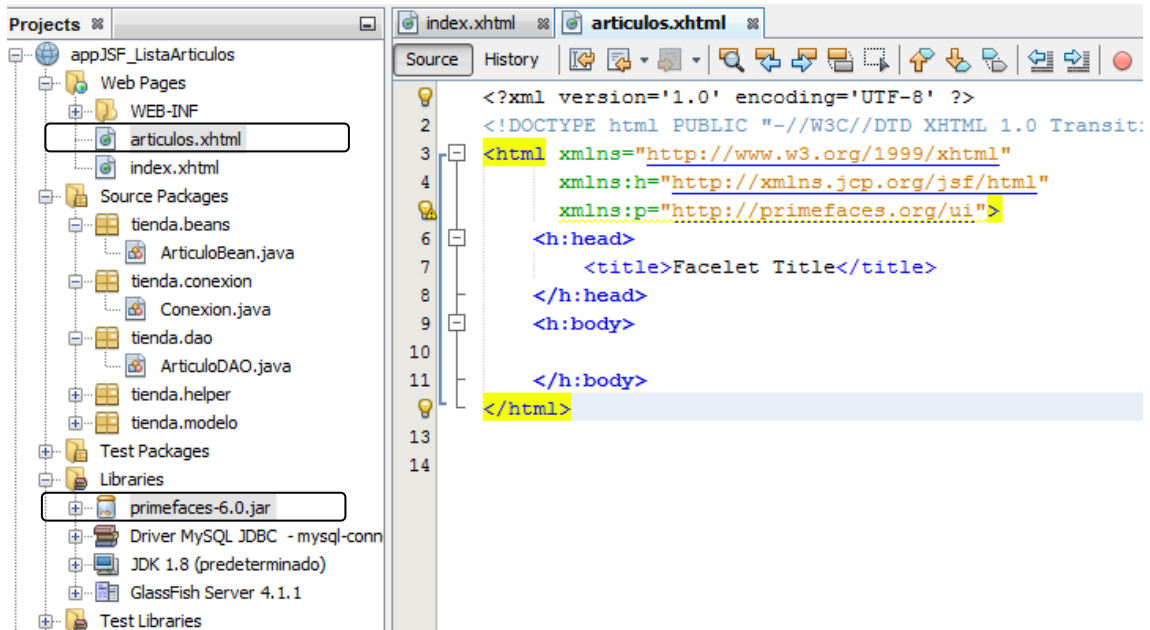
    public void setLstArticulos(ArrayList<Articulo> lstArticulos) {
        this.lstArticulos = lstArticulos;
    }

    public Articulo getArticulo() {
        return articulo;
    }

    public void setArticulo(Articulo articulo) {
        this.articulo = articulo;
    }
}
```

```
//método para asignar datos al arreglo de articulos lstArticulos
public void listar() {
    ArtículoDAO obja = new ArtículoDAO();
    lstArticulos = obja.list();
}
}
```

9. Crear página JSF con el nombre de “artículos”:



De la dirección web <http://primefaces.org/downloads>, descargar la librería **primefaces** utilizado en este laboratorio.

```
<?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:p="http://primefaces.org/ui">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
```

```

<h:form>
    <p:commandButton value="Listar articulos"
actionListener="#{articuloBean.listar()}" update="datos" />

    <p:dataTable id="datos"
value="#{articuloBean.lstArticulos}" var="ar">
        <p:column headerText="Codigo" width="50">
            <p:outputLabel value="#{ar.idarticulo}" />
        </p:column>
        <p:column headerText="Nombre" width="200">
            <p:outputLabel value="#{ar.nombre}" />
        </p:column>
        <p:column headerText="Descripcion">
            <p:outputLabel value="#{ar.descripcion}" />
        </p:column>
        <p:column headerText="Precio" width="50">
            <p:outputLabel value="#{ar.precio}" />
        </p:column>
    </p:dataTable>
</h:form>
</h:body>
</html>

```

//*****Ejecutar aplicación *****



Clic en el botón listar:

Listar articulos			
Codigo	Nombre	Descripcion	Precio
1	Polo	Polo Adidas color blanco cuello v	120.0
2	Zapatillas	Zapatillas adidas color blanco	160.0
3	Zapatillas	zapatilla tigre color negro y blanco	90.0
4	Buso	buso nike color negro	250.0
5	GUANTES DE PESCA	Guantes de pesca spinning Hart excelentes acabados diseñados por pescadores. Exterior en lycra elástica mayor confort con el detalle de que pueden utilizarse como toalla para secar el sudor del rostro. Guantes de pesca spinning zona de muñequera con malla ventilada de secado rápido y cierre de velcro. Palma de amara para un mejor agarre con orificios de ventilación	85.0
6	GORRA DE PESCA	Gorra HART PEAK de visera con faldón de protección solar trasero. Tira ajustable elástica en nuca y malla transpirable lateral	120.0
7	CHAQUETAS	(Impermeabilidad 6.000 mm; transpirabilidad 3.000 gr/m2 24 horas) - Clásica chaqueta de wading todoterreno construida con materiales de primera calidad, proporcionando un buen confort térmico en cualquier circunstancia	320.0
8	Mizuno Volleyball Shoes	Zapatillas Mizuno para Voley (botón)	130.0

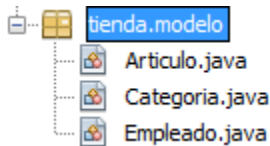
Laboratorio 3

Objetivo:

Desarrollar aplicación web utilizando el framework Java Server Faces (JSF) para realizar el mantenimiento de la tabla “Empeados” de la base de datos “tienda”.

Desarrollo:

1. En paquete “tienda.modelo” cree la clase Empleado con los métodos set y get:



```
package tienda.modelo;

public class Empleado {
    private int idempleado;
    private String nombre;
    private String paterno;
    private String materno;
    private String cargo;

    public int getIdempleado() {
        return idempleado;
    }

    public void setIdempleado(int idempleado) {
        this.idempleado = idempleado;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getPaterno() {
        return paterno;
    }

    public void setPaterno(String paterno) {
        this.paterno = paterno;
    }

    public String getMaterno() {
        return materno;
    }

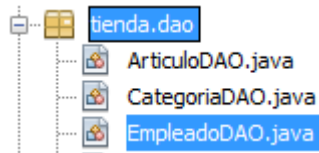
    public void setMaterno(String materno) {
        this.materno = materno;
    }

    public String getCargo() {
```

```
        return cargo;
    }

    public void setCargo(String cargo) {
        this.cargo = cargo;
    }
}
```

2. En paquete “tienda.dao” cree la clase EmpleadoDAO e implemente los métodos de la interface GenericCrud:



```
package tienda.dao;

import java.util.ArrayList;

import tienda.conexion.Conexion;
import tienda.helper.GenericCrud;
import tienda.modelo.Empleado;
public class EmpleadoDAO extends Conexion implements
GenericCrud<Empleado> {

    @Override
    public ArrayList<Empleado> list() {
        //comando sql para seleccionar todos los empleados
        String sql = "SELECT * FROM EMPLEADO";
        ArrayList<Empleado> lista = new ArrayList<>();
        Empleado emp;
        try {
            //abrir al conexion a la base de datos tienda
            cn = abrirConexion();
            //creamos un objeto PreparedStatement con la conexion a la base de
            //datos
            stm = cn.prepareStatement(sql);
            //ejecutar objeto PreparedStatement
            rs = stm.executeQuery();
            while (rs.next()) {
                emp = new Empleado();
                emp.setIdEmpleado(rs.getInt(1));
                emp.setNombre(rs.getString(2));
                emp.setPaterno(rs.getString(3));
                emp.setMaterno(rs.getString(4));
                emp.setCargo(rs.getString(5));
                lista.add(emp);
            }
        } catch (Exception ex) {
            System.out.println("Error lista empleado:" + ex);
        } finally {
            cerrar(cn);
            cerrar(stm);
            cerrar(rs);
        }
        return lista;
    }
}
```

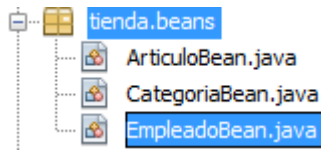
```
@Override
public Empleado get(Empleado t) {
    //comando sql para buscar empleado
    String sql = "SELECT * FROM EMPLEADO WHERE IDEMPLEADO=?";
    Empleado emp = null;
    try {
        //abrir al conexion a la base de datos tienda
        cn = abrirConexion();
        //creamos un objeto PreparedStatement con la conexion a la
        //base de datos
        stm = cn.prepareStatement(sql);
        //asignamos valores a los parametros ?
        stm.setInt(1, t.getIdempleado());
        //ejecutar objeto PreparedStatement
        rs = stm.executeQuery();
        if (rs.next()) {
            emp = new Empleado();
            emp.setIdempleado(rs.getInt("idempleado"));
            emp.setNombre(rs.getString("nombre"));
            emp.setPaterno(rs.getString("apepaterno"));
            emp.setMaterno(rs.getString("apematerno"));
            emp.setCargo(rs.getString("cargo"));
        }
    } catch (Exception ex) {
        System.out.println("Error en buscar empleado:" + ex);
    } finally {
        cerrar(cn);
        cerrar(stm);
        cerrar(rs);
    }
    return emp;
}

@Override
public void save(Empleado t) {
    //comando sql para insertar empleado
    String sql = "INSERT INTO EMPLEADO (NOMBRE,APEPATERNO,
APEMATERNO,CARGO) VALUES (?, ?, ?, ?) ";
    try {
        //abrir al conexion a la base de datos tienda
        cn = abrirConexion();
        //creamos un objeto PreparedStatement con la conexion a
        //la base de datos
        stm = cn.prepareStatement(sql);
        //asignamos valores a los parametros ?,?,?,?
        stm.setString(1, t.getNombre());
        stm.setString(2, t.getPaterno());
        stm.setString(3, t.getMaterno());
        stm.setString(4, t.getCargo());
        //ejecutar objeto PreparedStatement
        stm.executeUpdate();
    } catch (Exception ex) {
        System.out.println("Error en grabar empleado:" + ex);
    } finally {
        cerrar(cn);
        cerrar(stm);
        cerrar(rs);
    }
}
```

```
@Override
public void update(Empleado t) {
    //comando sql para modificar empleado
    String sql = "UPDATE EMPLEADO SET NOMBRE=?, APEPATERNO=?,
APEMATERNO=?, CARGO=? WHERE IDEMPLEADO=?";
    try {
        //abrir al conexion a la base de datos tienda
        cn = abrirConexion();
        //creamos un objeto PreparedStatement con la conexion a
        //la base de datos
        stm = cn.prepareStatement(sql);
        //asignamos valores a los parametros ?,?,?,?,?
        stm.setString(1, t.getNombre());
        stm.setString(2, t.getPaterno());
        stm.setString(3, t.getMaterno());
        stm.setString(4, t.getCargo());
        stm.setInt(5, t.getIdempleado());
        //ejecutar objeto PreparedStatement
        stm.executeUpdate();
    } catch (Exception ex) {
        System.out.println("Error en modificar empleado:" + ex);
    } finally {
        cerrar(cn);
        cerrar(stm);
        cerrar(rs);
    }
}

@Override
public void delete(Empleado t) {
    //comando sql para eliminar empleado
    String sql = "DELETE FROM EMPLEADO WHERE IDEMPLEADO=?";
    try {
        //abrir al conexion a la base de datos tienda
        cn = abrirConexion();
        //creamos un objeto PreparedStatement con la conexion a la base de
        //datos
        stm = cn.prepareStatement(sql);
        //asignamos valores a los parametro ?
        stm.setInt(1, t.getIdempleado());
        //ejecutar objeto PreparedStatement
        stm.executeUpdate();
    } catch (Exception ex) {
        System.out.println("Error en eliminar empleado:" + ex);
    } finally {
        cerrar(cn);
        cerrar(stm);
        cerrar(rs);
    }
}
}
```

3. Creamos un bean gestionado “empleadoBean” para establecer comunicación entre la vista y el bean gestionado:



```
package tienda.beans;

import java.util.ArrayList;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import tienda.dao.EmpleadoDAO;
import tienda.modelo.Empleado;

@ManagedBean
@ViewScoped
public class EmpleadoBean {

    public EmpleadoBean() {
        this.listar();
    }

    private Empleado empleado = new Empleado();
    ArrayList<Empleado> lstEmpleados;
    //variable para determinar accion de modificar o agregar nuevo
    //empleado
    private String accion;

    public String getAccion() {
        return accion;
    }

    public void setAccion(String accion) {
        this.limpiar();
        this.accion = accion;
    }

    public Empleado getEmpleado() {
        return empleado;
    }

    public void setEmpleado(Empleado empleado) {
        this.empleado = empleado;
    }

    public ArrayList<Empleado> getLstEmpleados() {
        return lstEmpleados;
    }

    public void setLstEmpleados(ArrayList<Empleado> lstEmpleados) {
        this.lstEmpleados = lstEmpleados;
    }

    public void listar() {

        EmpleadoDAO empDAO = new EmpleadoDAO();
        lstEmpleados = empDAO.list();
    }
}
```

```
public void getEmpleadoID(Empleado e) {
    //para mostrar en botón del cuadro de dialogo dlgEmpleados
    accion="Actualizar";
    EmpleadoDAO empDAO = new EmpleadoDAO();
    empleado = empDAO.get(e);
}

private void grabar() {
    EmpleadoDAO empDAO = new EmpleadoDAO();
    empDAO.save(empleado);
    this.listar();
}

private void actualizar() {
    EmpleadoDAO empDAO = new EmpleadoDAO();
    empDAO.update(empleado);
    this.listar();
}

//según valor de variable accion se ejecutara operación nuevo empleado
//o modificar empleado
public void operacion() {
    switch (accion) {
        case "Grabar":
            grabar();
            this.limpiar();
            break;
        case "Actualizar":
            actualizar();
            this.limpiar();
            break;
    }
}

//limpia el objeto empleado
private void limpiar() {
    this.empleado.setIdempleado(0);
    this.empleado.setNombre("");
    this.empleado.setPaterno("");
    this.empleado.setMaterno("");
    this.empleado.setCargo("");
}

public void eliminar(Empleado e) {
    EmpleadoDAO empDAO = new EmpleadoDAO();
    empDAO.delete(e);
    this.listar();
}
}
```



```

<!--cuadro de dialogo para ingresar o modificar empleado-->

<p:dialog header="Empleado" widgetVar="wdlgEmpleados"
id="dlgEmpleados">
    <h:form >
        <h:panelGrid columns="2">
            <p:outputLabel value="Nombre"/>
            <p:inputText
value="#{empleadoBean.empleado.nombre}" required="true"/>
            <p:outputLabel value="Apellido Paterno"/>
            <p:inputText
value="#{empleadoBean.empleado.paterno}" required="true"/>
            <p:outputLabel value="Apellido Materno"/>
            <p:inputText
value="#{empleadoBean.empleado.materno}" required="true"/>
            <p:outputLabel value="Cargo"/>
            <p:inputText
value="#{empleadoBean.empleado.cargo}" required="true"/>
            <p:outputLabel/>
            <p:commandButton value="#{empleadoBean.accion}"
actionListener="#{empleadoBean.operacion()}"
oncomplete="PF('wdlgEmpleados').hide();" update=":frmEmp:datos"/>
        </h:panelGrid>
    </h:form>
</p:dialog>

</h:body>
</html>

```

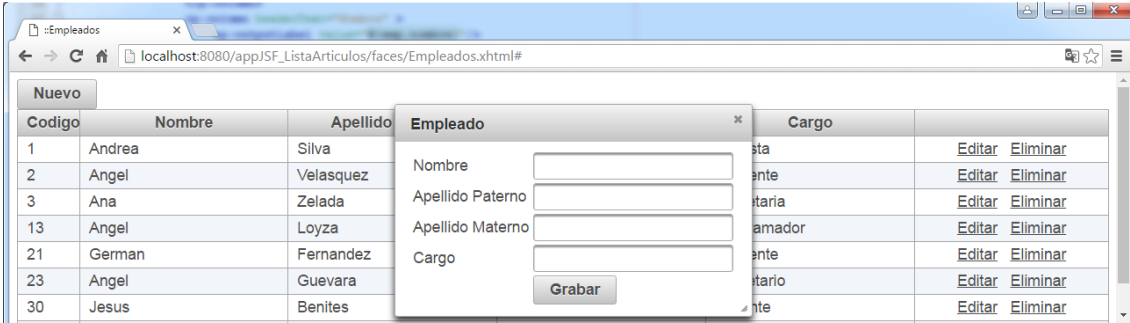
//*****Ejecución de la aplicación*****

Lista de empleados:



Codigo	Nombre	Apellido Paterno	Apellido Materno	Cargo	Editar	Eliminar
1	Andrea	Silva	Fernandez	Analista	Editar	Eliminar
2	Angel	Velasquez	Tello	Asistente	Editar	Eliminar
3	Ana	Zelada	Vela	Secretaria	Editar	Eliminar
13	Angel	Loyza	Delgado	Programador	Editar	Eliminar
21	German	Fernandez	Silva	Asistente	Editar	Eliminar
23	Angel	Guevara	Diaz	Secretario	Editar	Eliminar
30	Jesus	Benites	Alvarado	Gerente	Editar	Eliminar

Nuevo empleado:



Codigo	Nombre	Apellido	Cargo	Editar	Eliminar
1	Andrea	Silva	Analista	Editar	Eliminar
2	Angel	Velasquez	Asistente	Editar	Eliminar
3	Ana	Zelada	Secretaria	Editar	Eliminar
13	Angel	Loyza	Programador	Editar	Eliminar
21	German	Fernandez	Asistente	Editar	Eliminar
23	Angel	Guevara	Secretario	Editar	Eliminar
30	Jesus	Benites	Gerente	Editar	Eliminar

Empleado

Nombre:

Apellido Paterno:

Apellido Materno:

Cargo:

Ingresar empleado al cuadro de diálogo:

The screenshot shows a web browser window with a table of employees. A modal dialog box titled "Empleado" is open, allowing the user to add a new employee. The dialog contains the following fields:

- Nombre: José
- Apellido Paterno: Valverde
- Apellido Materno: Lozano
- Cargo: Administrador

There is a "Grabar" (Save) button at the bottom of the dialog. The background table shows existing employees with columns for Codigo, Nombre, Apellido, and Cargo.

Después de Grabar:

The screenshot shows the same web browser window, but the modal dialog is closed, and the employee list has been updated. The new employee, José Valverde Lozano, is now listed with the codigó 32 and the cargo of Administrador.

Codigo	Nombre	Apellido Paterno	Apellido Materno	Cargo	Editar	Eliminar
1	Andrea	Silva	Fernandez	Analista	Editar	Eliminar
2	Angel	Velasquez	Tello	Asistente	Editar	Eliminar
3	Ana	Zelada	Vela	Secretaria	Editar	Eliminar
13	Angel	Loyza	Delgado	Programador	Editar	Eliminar
21	German	Fernandez	Silva	Asistente	Editar	Eliminar
23	Angel	Guevara	Diaz	Secretario	Editar	Eliminar
30	Jesus	Benites	Alvarado	Gerente	Editar	Eliminar
32	José	Valverde	Lozano	Administrador	Editar	Eliminar

Editar Empleado

The screenshot shows the same web browser window. A modal dialog box titled "Empleado" is open, allowing the user to edit an existing employee. The dialog contains the following fields:

- Nombre: Jesus
- Apellido Paterno: Benites
- Apellido Materno: Alvarado
- Cargo: Gerente

There is an "Actualizar" (Update) button at the bottom of the dialog. The background table shows the current list of employees.

Cambiamos apellido del empleado con código treinta:

The screenshot shows the same web browser window. A modal dialog box titled "Empleado" is open, allowing the user to update the last name of an existing employee. The dialog contains the following fields:

- Nombre: Jesus
- Apellido Paterno: Villar
- Apellido Materno: Alvarado
- Cargo: Gerente

There is an "Actualizar" (Update) button at the bottom of the dialog. The background table shows the current list of employees.

Después de actualizar:



Codigo	Nombre	Apellido Paterno	Apellido Materno	Cargo	Editar	Eliminar
2	Angel	Velasquez	Tello	Asistente	Editar	Eliminar
3	Ana	Zelada	Vela	Secretaria	Editar	Eliminar
13	Angel	Loyza	Delgado	Programador	Editar	Eliminar
21	German	Fernandez	Silva	Asistente	Editar	Eliminar
23	Angel	Guevara	Diaz	Secretario	Editar	Eliminar
30	Jesus	Villar	Alvarado	Gerente	Editar	Eliminar
32	José	Valverde	Lozano	Administrador	Editar	Eliminar

Trabajo de Investigación:

Cree una aplicación para realizar el mantenimiento de la tabla Artículo de la base de datos tienda.

Consideraciones:

- Para registrar un nuevo artículo debe tener en cuenta en una lista desplegable las categorías de la tabla categoría de la base de datos tienda.
Asimismo deberá subir el archivo correspondiente al campo foto de la tabla Artículo.
- La eliminación de un artículo debe realizarse previa confirmación a través de un cuadro de diálogo.