

## Boletín 4. Creación de una aplicación J2EE

BDSW Tema 4 –La capa de acceso a los datos (DAO)

4 de marzo de 2015

## Índice

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Configuración del <i>pool</i> de conexiones en Glassfish</b>	<b>2</b>
<b>3</b>	<b>Creación de la aplicación empresarial</b>	<b>3</b>
<b>4</b>	<b>Creación del proyecto EJB</b>	<b>4</b>
4.1	Creación de los <i>Data Transfer Objects</i>	4
4.2	El DAO de la clase Empleado	4
4.3	El DAO de la clase Departamento	8
4.4	La clase de utilidad FinalString	10
4.5	El EJB EmpleadoBo	11
4.6	El EJB DepartamentoBo	14
<b>5</b>	<b>Creación de la aplicación web dinámica</b>	<b>16</b>
5.1	El módulo de gestión de empleados	16
5.1.1	Mostrar la lista de todos los empleados	16
5.1.2	Añadir un nuevo empleado	19
5.1.3	Buscar empleados por apellidos	21
5.1.4	Buscar empleados por departamento	22
5.2	El módulo de gestión de departamentos	24
5.2.1	Mostrar la lista de todos los departamentos	24
5.2.2	Añadir un nuevo departamento	25
5.3	Desplegar y ejecutar la aplicación	26
<b>6</b>	<b>Ejercicio</b>	<b>27</b>

## Índice de figuras

1	Panel de creación del proyecto empresarial jdbcPersonal.	3
2	Paneles de creación del proyecto EJB jdbcPersonalEJB.	4
3	Creación del EJB stateless EmpleadoBo y de la interface remota EmpleadoBoRemote.	11
4	Panel de creación del <i>Dynamic Web Project</i> jdbcPersonalWA.	17
5	Panel de configuración del <i>Build Path</i> del proyecto jdbcPersonalWA.	17
6	Diagrama de secuencia del caso de uso que obtiene la lista de todos los empleados.	18
7	Diagrama de secuencia del caso de uso que obtiene la lista de todos los empleados.	19
8	Diagrama de secuencia del caso de que permite buscar empleados por sus apellidos.	21
9	Diagrama de secuencia del caso de que permite buscar empleados adscritos a un departamento.	23
10	Lista de empleados tal y como se presenta a través del navegador.	27
11	Vista desde el navegador del formulario para añadir un nuevo empleado	28

## Índice de listados

1	Código de la interface EmpleadoDao.java. . . . .	5
2	Código de la clase MySQLEmpleadoDao.java. . . . .	5
3	Código de la interface DepartamentoDao.java. . . . .	8
4	Código de la clase MySQLDepartamentoDao.java. . . . .	9
5	Código de la clase de utilidad FinalString.java. . . . .	10
6	Código de la interface EmpleadoBoRemote.java. . . . .	11
7	Código de la clase EmpleadoBo.java. . . . .	12
8	Código de la interface DepartamentoBoRemote.java. . . . .	14
9	Código de la clase DepartamentoBo.java. . . . .	14
10	Código del <i>servlet</i> EmpleadoList.java. . . . .	18
11	Código del <i>servlet</i> EmpleadoNew.java. . . . .	19
12	Código del <i>servlet</i> EmpleadoFindByName.java. . . . .	21
13	Código del <i>servlet</i> EmpleadoFindByDept.java. . . . .	22
14	Código del <i>servlet</i> DepartamentoList.java. . . . .	24
15	Código del <i>servlet</i> DepartamentoNew.java. . . . .	25

## 1 Introducción

Siguiendo con el ejemplo de la actividad empresarial simple y teniendo en cuenta que:

- En el Tema 1 se introdujeron los VOs (DTOs, DTs) Empleado y Departamento.
- En el Tema 2 se creó la base de datos Empresa.
- En el Tema 3 estudiamos cómo Java interactúa con una base de datos relacional mediante JDBC.
- En el Tema 4 estamos descubriendo el concepto de DAO.

En el siguiente ejercicio vamos a implementar los DAOs de las clases Empleado y Departamento como EJBs en un servidor de aplicaciones utilizando el driver JDBC de MySQL para las operaciones básicas de la base de datos.

Desarrollaremos después una pequeña aplicación web dinámica dedicada a la gestión de recursos humanos (gestión de empleados, departamentos y asignación de proyectos) que se apoyará sobre los EJB reutilizables que hemos desarrollado.

## 2 Configuración del *pool* de conexiones en Glassfish

Para que las aplicaciones que se ejecutan en el servidor Glassfish tengan acceso a la base de datos Empresa es necesario configurar un *pool* de conexiones en el servidor de aplicaciones. En esta sección vamos a crear el *pool* de conexiones personalpool que será el punto de acceso a la base de datos de los EJB de nuestra aplicación de gestión de recursos humanos. El procedimiento es el siguiente:

1. Entramos en la página de administración de Glassfish. Para ello iniciamos el servidor desde Eclipse y una vez arrancado pulsamos sobre el servidor con el botón derecho. En el menú emergente seleccionamos la entrada **Glassfish Enterprise Server** → **View Admin Console**. Al entorno de administración se accede mediante usuario admin y password isaw.
2. En la página de administración vamos a **Recursos** → **JDBC** → **Connection Pool** y seleccionamos **New**.
3. Introducimos los siguientes datos:
  - Name: personalpool.
  - Resource Type: javax.sql.DataSource.
  - Database Vendor: MySQL.
 y pulsamos sobre **Next**.
4. En la segunda página del asistente introducimos los siguientes datos:
  - Database Name: empresa.
  - User: recursos.
  - Password: recpwd.
  - Server Name: localhost.
  - Port Number: 3306.
  - Url y url: jdbc:mysql://localhost:3306/empresa

Una vez configurado, pulsamos sobre **Finish** y comprobamos que tenemos acceso a la base de datos pulsando sobre el botón **Ping**.

**Aviso:** Si no es posible contactar con la base de datos deberá repasar todo el proceso hasta que la configuración sea correcta.

5. En la consola de administración de Glassfish vamos a **Resources** → **JDBC** → **JDBC Resources** y seleccionamos **New**.

6. En la página de recursos JNDI proporcionamos los siguientes datos para el nuevo recurso:

- JNDI Name: jdbc/personalpool.
- Pool Name: personalpool.
- Status: Enabled.

Para acabar pulsamos sobre **Finish**.

Este *pool* de conexiones es el recurso que utilizarán nuestros EJBs para acceder a la base de datos y que inyectaremos mediante la directiva "@Resource".

### 3 Creación de la aplicación empresarial

En este apartado y en los siguientes se explica paso a paso el proceso de creación de la aplicación J2EE empresarial que vamos a desarrollar. Como veremos, la aplicación estará formada por varios proyectos de Eclipse interrelacionados que crearemos de forma secuencial.

Primero crearemos el proyecto empresarial que servirá de núcleo alrededor del cual se organizan los otros subproyectos. Para ello, en Eclipse seleccionamos el menú **File** → **New** → **Enterprise Application Project**. Crear el proyecto jdbcPersonal a través del panel correspondiente como se muestra en la figura 1.

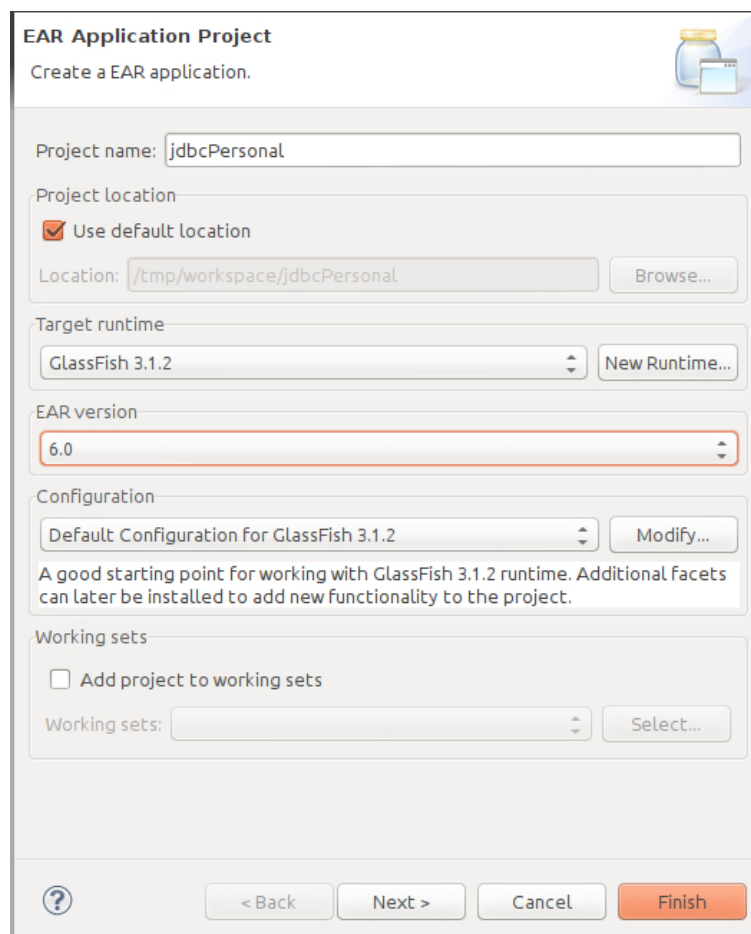


Figura 1: Panel de creación del proyecto empresarial jdbcPersonal.

## 4 Creación del proyecto EJB

En Eclipse seleccionamos el menú **File** → **New** → **EJB Project**. A través del panel pondremos nombre al proyecto (jdbcPersonalEJB) y lo asignaremos al proyecto EAR jdbcPersonal marcando la casilla **EAR Membership** como se muestra en la figura 2

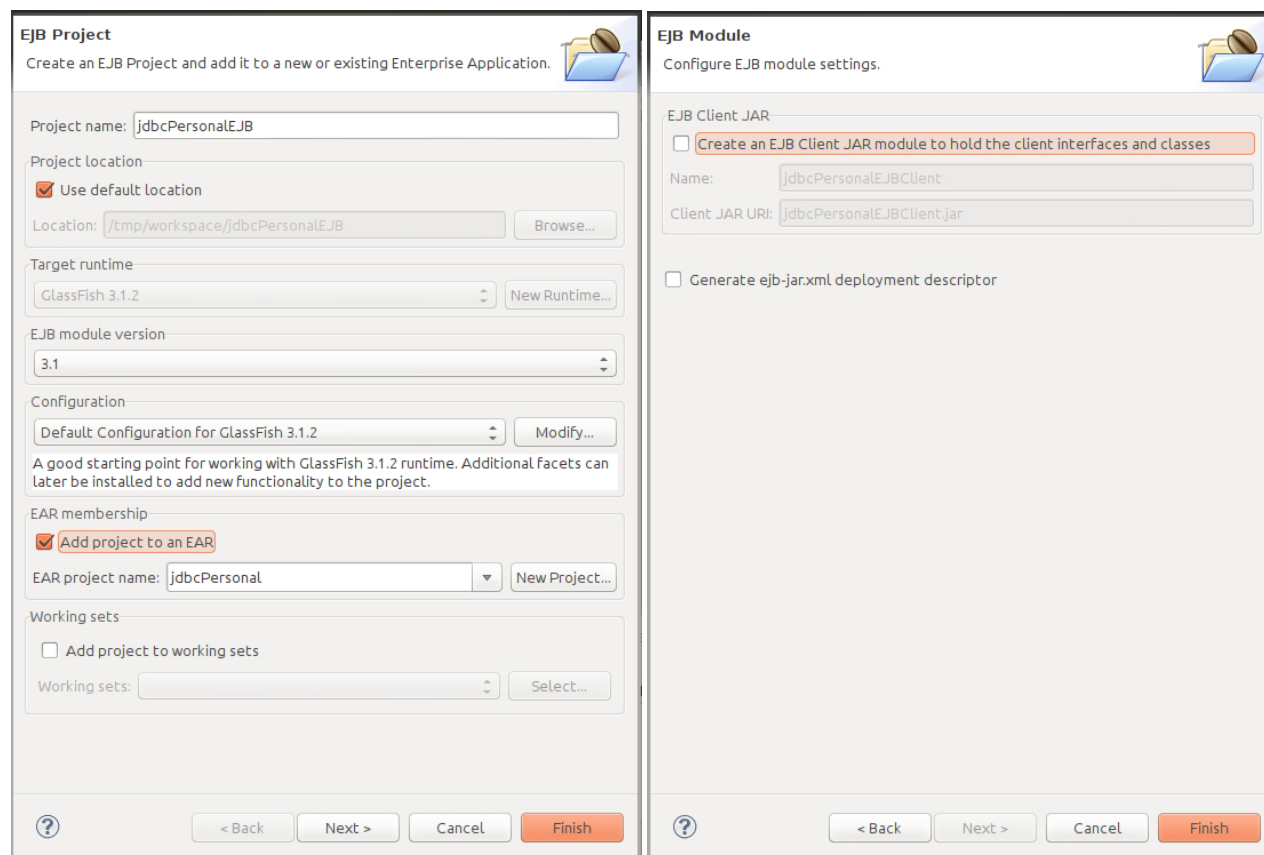


Figura 2: Paneles de creación del proyecto EJB jdbcPersonalEJB.

### Nota:

Aunque siempre es posible configurar el *membership* del proyecto EJB posteriormente, en este momento es importante seguir cuidadosamente las instrucciones para evitar problemas de configuración y errores inesperados difíciles de trazar.

### 4.1 Creación de los Data Transfer Objects

Los DTO son los mismos de la práctica anterior, por lo que bastará con copiarlos en la nueva ubicación. Para crear los DTO, procederemos del siguiente modo:

- Seleccionamos con el botón derecho el icono del proyecto jdbcPersonalEJB y en el menú contextual seleccionamos **New** → **Package** y creamos el paquete es.uv.bds.dto.
- Copiamos los ficheros \*.java (que implementan las clases Empleado y Departamento) al directorio /workspace/jdbcPersonalEJB/ejbModule/es/uv/bds/dto bajo el espacio de trabajo (*workspace*).
- Refrescamos (*Refresh*) el catálogo de Eclipse para que tenga en cuenta los nuevos ficheros. Para ello puede seleccionar la entrada correspondiente bajo el menú **File** o pulsar la tecla **F5**.

### 4.2 El DAO de la clase Empleado

Todos los DAOs se definen e implementan en el paquete es.uv.bds.dao, que crearemos previamente de la misma forma que en el punto anterior.

El primer paso consiste en la creación de la interface EmpleadoDao, en la que se definirán todos los métodos que caracterizan el comportamiento del DAO y que deberán implementar las clases DAO específicas.

Para crear la interface pulsamos con el botón derecho sobre el ícono jdbcPersonalEJB del **Project Explorer** y en el menú emergente seleccionamos **New** → **Interface**. Esta interface define varios métodos CRUD y una serie de métodos adicionales:

- El método `getAllEmpleados` que obtiene y devuelve una lista con todos los empleados de la base de datos.
- El método `findEmpleadosByApellidos` que obtiene la lista (posiblemente vacía) con todos los empleados cuyos apellidos empiezan por la cadena pasada como parámetros. Observe la forma en que se trata la cadena para adecuarla al criterio de búsqueda utilizado en la base de datos.
- El método `findEmpleadosByDepto` que obtiene la lista de empleados que trabajan en un determinado departamento. El método `keysEmpleado` es un método de apoyo que utilizaremos para construir las listas desplegables de nuestra aplicación web.

El código de la interface se muestra en la figura 1

Listado 1: Código de la interface `EmpleadoDao.java`.

```

1 package es.uv.bds.dao;
2
3 import java.sql.SQLException;
4 import java.util.List;
5 import java.util.TreeMap;
6
7 import es.uv.bds.dto.Empleado;
8
9 public interface EmpleadoDao {
10     public Empleado findById(int id) throws SQLException;
11     public void persist(Empleado e) throws SQLException;
12     public void remove(Empleado e) throws SQLException;
13     public List<Empleado> getAllEmpleados() throws SQLException;
14     public List<Empleado> findEmpleadoByApellidos(String apellidos) throws SQLException;
15     public List<Empleado> findEmpleadosByDepto(int id) throws SQLException;
16     public TreeMap<String, Integer> keysEmpleado() throws SQLException;
17 }

```

Ahora definiremos la clase `MySQLEmpleadoDao` que implementa la interface `EmpleadoDao` y que corresponde a la implementación específica del DAO que trabaja con una base de datos MySQL.

Para crear la clase pulsamos con el botón derecho sobre el ícono `jdbcPersonalEJB` del `Project Explorer` y en el menú contextual seleccionamos `New` → `Class`. La clase `MySQLEmpleadoDao` se implementa en el mismo paquete que la interface.

El código de la clase `MySQLEmpleadoDao` se muestra en el listado 2. Como se observa en el código, el constructor del DAO requiere un parámetro de tipo `Connection` que será suministrado por el *Business Object* y que representará una conexión abierta con la base de datos correspondiente. La implementación de los métodos utiliza sentencias SQL planas y métodos JDBC para acceder a la base de datos.

Listado 2: Código de la clase `MySQLEmpleadoDao.java`.

```

1 package es.uv.bds.dao;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.util.ArrayList;
8 import java.util.List;
9 import java.util.TreeMap;
10
11 import es.uv.bds.dto.Empleado;
12
13 public class MySQLEmpleadoDao implements EmpleadoDao {
14     private static final String FINDBYID =
15         "SELECT idEmpleado,nombre,apellidos,departamento," +
16         " fechaContrato,puesto,nivelEducacion," +
17         " sueldo,complemento" +
18         " FROM Empleados" +
19         " WHERE idEmpleado = ?";
20     private static final String INSERT =
21         "INSERT INTO Empleados(nombre, apellidos, departamento, fechaContrato," +
22         " puesto, nivelEducacion, sueldo, complemento)" +
23         " VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
24     private static final String DELETE =
25         "DELETE FROM Empleados WHERE idEmpleado = ?";

```

```

26 private static final String READALL =
27     "SELECT idEmpleado, nombre, apellidos, departamento, fechaContrato," +
28     "    puesto, nivelEducacion, sueldo, complemento" +
29     " FROM Empleados" +
30     " ORDER BY apellidos";
31 private static final String FINDBYAPELLIDOS =
32     "SELECT idEmpleado, nombre, apellidos, departamento, fechaContrato," +
33     "    puesto, nivelEducacion, sueldo, complemento" +
34     " FROM Empleados" +
35     " WHERE apellidos LIKE ?" +
36     " ORDER BY apellidos";
37 private static final String FINDBYDEPTO =
38     "SELECT idEmpleado, nombre, apellidos, departamento, fechaContrato," +
39     "    puesto, nivelEducacion, sueldo, complemento" +
40     " FROM Empleados" +
41     " WHERE departamento = ?" +
42     " ORDER BY apellidos";
43 private static final String FINDKEYS =
44     "SELECT idEmpleado, nombre, apellidos" +
45     " FROM Empleados";
46
47 protected Connection con;
48
49 public MySQLEmpleadoDao(Connection con) {
50     this.con = con;
51 }
52
53 @Override
54 public Empleado findById(int id) throws SQLException {
55     Empleado empleado = new Empleado();
56     PreparedStatement sqlQuery = con.prepareStatement(FINDBYID);
57     sqlQuery.setInt(1, id);
58     ResultSet rs = sqlQuery.executeQuery();
59     rs.last ();
60     int rowcount = rs.getRow();
61     if (rowcount == 1) {
62         rs.first ();
63         empleado.setIdEmpleado(rs.getInt(1));
64         empleado.setNombre(rs.getString(2));
65         empleado.setApellidos(rs.getString(3));
66         empleado.setDepartamento(rs.getInt(4));
67         empleado.setFechaContrato(rs.getDate(5));
68         empleado.setPuesto(rs.getString(6));
69         empleado.setNivelEducacion(rs.getShort(7));
70         empleado.setSueldo(rs.getFloat(8));
71         empleado.setComplemento(rs.getFloat(9));
72     }
73     return empleado;
74 }
75
76 @Override
77 public void persist(Empleado e) throws SQLException {
78     PreparedStatement sqlQuery = con.prepareStatement(INSERT);
79
80     sqlQuery.setString(1, e.getNombre());
81     sqlQuery.setString(2, e.getApellidos());
82     sqlQuery.setInt(3, e.getDepartamento());
83     sqlQuery.setDate(4, (java.sql.Date)e.getFechaContrato());
84     sqlQuery.setString(5, e.getPuesto());
85     sqlQuery.setInt(6, e.getNivelEducacion());
86     sqlQuery.setDouble(7, e.getSueldo());
87     sqlQuery.setDouble(8, e.getComplemento());
88
89     sqlQuery.executeUpdate();
90     con.commit();
91 }
92

```

```
93  @Override
94  public void remove(Empleado e) throws SQLException {
95      PreparedStatement sqlQuery = con.prepareStatement(DELETE);
96      sqlQuery.setInt(1, e.getIdEmpleado());
97      sqlQuery.executeUpdate();
98  }
99
100  @Override
101  public List<Empleado> getAllEmpleados() throws SQLException {
102      List<Empleado> empleados = new ArrayList<Empleado>();
103
104      PreparedStatement sqlQuery = con.prepareStatement(READALL);
105
106      ResultSet rs = sqlQuery.executeQuery();
107      rs.beforeFirst();
108      while (rs.next()) {
109          Empleado empleado = new Empleado();
110          empleado.setIdEmpleado(rs.getInt(1));
111          empleado.setNombre(rs.getString(2));
112          empleado.setApellidos(rs.getString(3));
113          empleado.setDepartamento(rs.getInt(4));
114          empleado.setFechaContrato(rs.getDate(5));
115          empleado.setPuesto(rs.getString(6));
116          empleado.setNivelEducacion(rs.getShort(7));
117          empleado.setSueldo(rs.getFloat(8));
118          empleado.setComplemento(rs.getFloat(9));
119
120          empleados.add(empleado);
121      }
122      return empleados;
123  }
124
125  @Override
126  public List<Empleado> findEmpleadoByApellidos(String apellidos)
127  throws SQLException {
128      List<Empleado> empleados = new ArrayList<Empleado>();
129
130      PreparedStatement sqlQuery = con.prepareStatement(FINDBYAPELLIDOS);
131      sqlQuery.setString(1, apellidos);
132
133      ResultSet rs = sqlQuery.executeQuery();
134      rs.beforeFirst();
135      while (rs.next()) {
136          Empleado empleado = new Empleado();
137          empleado.setIdEmpleado(rs.getInt(1));
138          empleado.setNombre(rs.getString(2));
139          empleado.setApellidos(rs.getString(3));
140          empleado.setDepartamento(rs.getInt(4));
141          empleado.setFechaContrato(rs.getDate(5));
142          empleado.setPuesto(rs.getString(6));
143          empleado.setNivelEducacion(rs.getShort(7));
144          empleado.setSueldo(rs.getFloat(8));
145          empleado.setComplemento(rs.getFloat(9));
146
147          empleados.add(empleado);
148      }
149      return empleados;
150  }
151
152  @Override
153  public List<Empleado> findEmpleadosByDepto(int id) throws SQLException {
154      List<Empleado> empleados = new ArrayList<Empleado>();
155
156      PreparedStatement sqlQuery = con.prepareStatement(FINDBYDEPTO);
157      sqlQuery.setInt(1, id);
158
159      ResultSet rs = sqlQuery.executeQuery();
```

```

160     rs.beforeFirst();
161     while (rs.next()) {
162         Empleado empleado = new Empleado();
163         empleado.setIdEmpleado(rs.getInt(1));
164         empleado.setNombre(rs.getString(2));
165         empleado.setApellidos(rs.getString(3));
166         empleado.setDepartamento(rs.getInt(4));
167         empleado.setFechaContrato(rs.getDate(5));
168         empleado.setPuesto(rs.getString(6));
169         empleado.setNivelEducacion(rs.getShort(7));
170         empleado.setSueldo(rs.getFloat(8));
171         empleado.setComplemento(rs.getFloat(9));
172
173         empleados.add(empleado);
174     }
175     return empleados;
176 }
177
178 @Override
179 public TreeMap<String, Integer> keysEmpleado() throws SQLException {
180     String nombre;
181     int id;
182     TreeMap<String, Integer> keyEmp = new TreeMap<String, Integer>();
183
184     PreparedStatement sqlQuery = con.prepareStatement(FINDKEYS);
185
186     ResultSet rs = sqlQuery.executeQuery();
187     rs.beforeFirst();
188     while (rs.next()) {
189         nombre = rs.getString("apellidos") + ", " + rs.getString("nombre");
190         id = rs.getInt("idEmpleado");
191         keyEmp.put(nombre, id);
192     }
193     return keyEmp;
194 }
195 }

```

### 4.3 El DAO de la clase Departamento

Siguiendo un procedimiento similar al descrito en el apartado anterior, vamos a crear el DAO local de la clase Departamento. Para ello creamos la interface EmpleadoDao que declarará todos los métodos que un DAO de Departamento debe implementar. El código de la interface se muestra en el listado 3. Como en el caso anterior, el método keysDepartamento se utilizará para crear las listas desplegables en los formularios de nuestra aplicación web.

Listado 3: Código de la interface DepartamentoDao.java.

```

1 package es.uv.bds.w.dao;
2
3 import java.sql.SQLException;
4 import java.util.List;
5 import java.util.TreeMap;
6
7 import es.uv.bds.w.dto.Departamento;
8
9 public interface DepartamentoDao {
10     public Departamento findById(int id) throws SQLException;
11     public void persist(Departamento d) throws SQLException;
12     public void remove(Departamento d) throws SQLException;
13     public List<Departamento> getAllDepartamentos() throws SQLException;
14     public TreeMap<String, Integer> keysDepartamento() throws SQLException;
15 }

```

Ahora definiremos la clase MySQLDepartamentoDao que implementa la interface DepartamentoDao y que corresponde a la implementación específica del DAO que trabaja con una base de datos MySQL.

El código de la clase MySQLDepartamentoDao se muestra en el listado 4. Como en el caso anterior, el constructor del DAO requiere un parámetro de tipo Connection que será suministrado por el *Business Object* y que representará una



conexión abierta con la base de datos correspondiente.

Listado 4: Código de la clase MySQLDepartamentoDao.java.

```
1 package es.uv.bds.dao;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.util.ArrayList;
8 import java.util.List;
9 import java.util.TreeMap;
10
11 import es.uv.bds.dto.Departamento;
12
13 public class MySQLDepartamentoDao implements DepartamentoDao {
14
15     private static String FINDBYID =
16         "SELECT idDepartamento, nombre, manager" +
17         " FROM Departamentos" +
18         " WHERE idDepartamento = ?";
19     private static String INSERT =
20         "INSERT INTO Departamentos(nombre, manager) " +
21         "VALUES (?, ?)";
22     private static String DELETE =
23         "DELETE FROM Departamentos" +
24         " WHERE idDepartamento = ?";
25     private static String READALL =
26         "SELECT idDepartamento, nombre, manager" +
27         " FROM Departamentos" +
28         " ORDER BY nombre";
29     private static String FINDKEYS =
30         "SELECT idDepartamento, nombre" +
31         " FROM Departamentos;";
32
33     protected Connection con;
34
35     public MySQLDepartamentoDao(Connection con) {
36         this.con = con;
37     }
38
39     @Override
40     public Departamento findById(int id) throws SQLException {
41         Departamento departamento = new Departamento();
42
43         PreparedStatement sqlQuery = con.prepareStatement(FINDBYID);
44         sqlQuery.setInt(1, id);
45
46         ResultSet rs = sqlQuery.executeQuery();
47         rs.last();
48         int rowcount = rs.getRow();
49         if (rowcount == 1) {
50             rs.first();
51             departamento.setIdDepartamento(rs.getInt(1));
52             departamento.setNombre(rs.getString(2));
53             departamento.setManager(rs.getInt(3));
54         }
55         return departamento;
56     }
57
58     @Override
59     public void persist(Departamento d) throws SQLException {
60         PreparedStatement sqlQuery = con.prepareStatement(INSERT);
61         sqlQuery.setString(1, d.getNombre());
62         sqlQuery.setInt(2, d.getManager());
63
64         sqlQuery.executeUpdate();
65     }
66 }
```

```

65     con.commit();
66 }
67
68 @Override
69 public void remove(Departamento d) throws SQLException {
70     PreparedStatement sqlQuery = con.prepareStatement(DELETE);
71     sqlQuery.setInt(1, d.getIdDepartamento());
72     sqlQuery.executeUpdate();
73 }
74
75 @Override
76 public List<Departamento> getAllDepartamentos() throws SQLException {
77     List <Departamento> departamentos = new ArrayList<Departamento>();
78
79     PreparedStatement sqlQuery = con.prepareStatement(READALL);
80     ResultSet rs = sqlQuery.executeQuery();
81     rs.beforeFirst ();
82     while (rs.next()) {
83         Departamento departamento = new Departamento();
84         departamento.setIdDepartamento(rs.getInt(1));
85         departamento.setNombre(rs.getString(2));
86         departamento.setManager(rs.getInt(3));
87         departamentos.add(departamento);
88     }
89     return departamentos;
90 }
91
92 @Override
93 public TreeMap<String, Integer> keysDepartamento() throws SQLException {
94     TreeMap<String, Integer> keyEmp = new TreeMap<String, Integer>();
95
96     PreparedStatement sqlQuery = con.prepareStatement(FINDKEYS);
97     ResultSet rs = sqlQuery.executeQuery();
98     rs.beforeFirst();
99     while (rs.next()) {
100         keyEmp.put(rs.getString("nombre"), rs.getInt("idDepartamento"));
101     }
102     return keyEmp;
103 }
104 }

```

#### 4.4 La clase de utilidad FinalString

La clase FinalString es una pequeña clase de utilidad cuya misión es dar formato a los String que se recogen a través de los formularios html. FinalString hace un tratamiento muy simple de las cadenas de caracteres:

- Elimina los caracteres en blanco al principio y final de línea.
- Consolida cualquier combinación de uno o más espacios en blanco en un sólo blanco.
- Convierte los caracteres de la cadena a mayúsculas.

Esta clase se creará dentro del paquete `es.uv.bdsu.util`, cuenta con un único método *static* `arregla` y su código se muestra en el listado 5

Listado 5: Código de la clase de utilidad FinalString.java.

```

1 package es.uv.bdsu.util;
2
3 public class FinalString {
4     public static String arregla(String s) {
5         String n;
6         n = s.trim().toUpperCase();
7         n = n.replaceAll("( )+", " ");
8         return n;
9     }
10 }

```

## 4.5 El EJB EmpleadoBo

Del mismo modo que los DAOs encapsulan el acceso a la base de datos, los *Business Objects* encapsulan las reglas de negocio. En nuestro caso, el EJB EmpleadoBo se encarga, por ejemplo, del “negocio” de suministrar la lista de Empleado que utilizará nuestro *servlet* o de comprobar los valores de los atributos a la hora de añadir un nuevo empleado a la base de datos. El EJB presentará una interface remota, de modo que responderá a las invocaciones externas al servidor de aplicaciones que realizará el *servlet*.

Dentro del proyecto jdbcPersonalEJB crearemos el paquete es.uv.bdsb.bo. Para crear el EJB pulsamos con el botón derecho sobre el ícono jdbcPersonalEJB del Project Explorer y en el menú contextual seleccionamos New → Session Bean (EJB 3.x). Utilizaremos las opciones que se muestran en la figura 3.

La interface EmpleadoBoRemote, que se puede ver en el listado 6, define cinco métodos:

- Un método newEmpleado que convertirá en persistente un objeto de la clase Empleado.
- Un método para obtener la lista de todos los empleados de la base de datos: listaEmpleados.
- Tres métodos de búsqueda:
  - findEmpleado que busca un empleado a través de su clave primaria idEmpleado.
  - buscarEmpleado que obtiene la lista de todos los empleados cuyos apellidos comienzan por un determinado String.
  - findEmpleadosByDepto, que devuelve la lista de todos los empleados adscritos a un departamento identificado por su idDepartamento.

Figura 3: Creación del EJB stateless EmpleadoBo y de la interface remota EmpleadoBoRemote.

Listado 6: Código de la interface EmpleadoBoRemote.java.

```

1 package es.uv.bdsb.bo;
2
3 import java.sql.Date;
4
5 @Remote
6 public interface EmpleadoBoRemote {
7     List<Empleado> listaEmpleados();
8     public Empleado findEmpleado(int id);
9     public List<Empleado> buscarEmpleado(String apellidos);
10    public void newEmpleado(String nombre, String apellidos, String puesto,
11    Date date, Short nivelEducacion, float sueldo,
12    float complemento, int depto);
13    public TreeMap<String, Integer> keysEmpleado();
14    public List<Empleado> findEmpleadosByDepto(int id);
15 }

```

En el listado 7 se muestra el código del EJB EmpleadoBo en el que se implementan los métodos definidos en la interface remota.

**Nota:** Con la anotación `@Resource(lookup = "jdbc/personalpool")`, el BO obtiene del servidor de aplicaciones una instancia de la clase `DataSource` que se utiliza para abrir una conexión a la base de datos. Esta instancia `Connection` se pasa como parámetro para instanciar el DAO `EmpleadoDao` correspondiente a través del cual se realizarán todas las operaciones de acceso a la base de datos. Observe como se hace uso de los métodos etiquetados como `@PostConstruct` y `@PreDestroy` para realizar estas tareas de inicialización.

Listado 7: Código de la clase `EmpleadoBo.java`.

```

1 package es.uv.bdsb.bo;
2
3 import java.sql.Connection;
4 import java.sql.Date;
5 import java.sql.SQLException;
6 import java.util.List;
7 import java.util.TreeMap;
8 import javax.annotation.PostConstruct;
9 import javax.annotation.PreDestroy;
10 import javax.annotation.Resource;
11 import javax.ejb.Stateless;
12 import com.sun.appserv.jdbc.DataSource;
13
14 import es.uv.bdsb.dao.MySQLEmpleadoDao;
15 import es.uv.bdsb.dto.Empleado;
16 import es.uv.bdsb.util.FinalString;
17 /**
18  * Session Bean implementation class EmpleadoBo
19  */
20 @Stateless(mappedName = "EmpleadoBO")
21 public class EmpleadoBo implements EmpleadoBoRemote {
22     @Resource(lookup = "jdbc/personalpool")
23     private DataSource ds;
24     private Connection con;
25     private MySQLEmpleadoDao edao = null;
26
27     @PostConstruct
28     public void inicia() {
29         try {
30             con = ds.getConnection();
31             edao = new MySQLEmpleadoDao(con);
32         }
33         catch (SQLException e) {
34             System.out.println("EmpleadoBo: Error en PostConstruct");
35             e.printStackTrace();
36         }
37     }
38     @PreDestroy
39     public void finaliza() {
40         try {
41             con.close();
42         }
43         catch (SQLException e) {
44             System.out.println("EmpleadoBo: Error en PreDestroy");
45             e.printStackTrace();
46         }
47     }
48     /**
49     * Default constructor.
50     */
51     public EmpleadoBo() {}
52
53     @Override
54     public List<Empleado> listaEmpleados() {
55         List<Empleado> empleados = null;
56         try {
57             empleados = edao.getAllEmpleados();
58         }

```

```

59     catch (SQLException e) {
60         System.out.println("EmpleadoBo: Error en listaEmpleados");
61         e.printStackTrace();
62     }
63     return empleados;
64 }
65
66 @Override
67 public Empleado findEmpleado(int id) {
68     Empleado empleado = null;
69     try {
70         empleado = edao.findById(id);
71     }
72     catch (SQLException e) {
73         System.out.println("EmpleadoBo: Error en findEmpleado");
74         e.printStackTrace();
75     }
76     return empleado;
77 }
78
79 @Override
80 public List<Empleado> buscarEmpleado(String apellidos) {
81     List<Empleado> empleados = null;
82     // Adaptamos el string y concatenamos el comodin al final
83     apellidos = FinalString.arregla(apellidos) + "%";
84     try {
85         empleados = edao.findEmpleadoByApellidos(apellidos);
86     }
87     catch (SQLException e) {
88         System.out.println("EmpleadoBo: Error en buscarEmpleado");
89         e.printStackTrace();
90     }
91     return empleados;
92 }
93
94 @Override
95 public void newEmpleado(String nombre, String apellidos, String puesto,
96 Date date, Short nivelEducacion, float sueldo,
97 float complemento, int depto) {
98     Empleado nuevo = new Empleado();
99     nuevo.setNombre(FinalString.arregla(nombre));
100    nuevo.setApellidos(FinalString.arregla(apellidos));
101    nuevo.setPuesto(puesto);
102    nuevo.setFechaContrato(date);
103    nuevo.setNivelEducacion(nivelEducacion);
104    nuevo.setSueldo(sueldo);
105    nuevo.setComplemento(complemento);
106    nuevo.setDepartamento(depto);
107    try {
108        // Salvamos el nuevo empleado
109        edao.persist(nuevo);
110    }
111    catch (SQLException e) {
112        System.out.println("EmpleadoBo: Error en nuevoEmpleado");
113        e.printStackTrace();
114    }
115 }
116
117 @Override
118 public TreeMap<String, Integer> keysEmpleado() {
119     TreeMap<String, Integer> keyEmp = new TreeMap<String, Integer>();
120     try {
121         keyEmp = edao.keysEmpleado();
122     }
123     catch (SQLException e) {
124         System.out.println("EmpleadoBo: Error en keysEmpleado");
125         e.printStackTrace();

```

```

126     }
127     return keyEmp;
128 }
129
130 @Override
131 public List<Empleado> findEmpleadosByDeppto(int id) {
132     List<Empleado> empleados = null;
133     try {
134         empleados = edao.findEmpleadosByDeppto(id);
135     }
136     catch (SQLException e) {
137         System.out.println("EmpleadoBo: Error en findEmpleadosByDeppto");
138         e.printStackTrace();
139     }
140     return empleados;
141 }
142 }

```

**Aviso:** Es importante que analice y comprenda el uso de las funciones inyectadas con “@PostConstruct” y “@PreDestroy”.

## 4.6 El EJB DepartamentoBo

Ahora crearemos el EJB DepartamentoBo, que se encargará de las reglas de “negocio” que implican departamentos. En este caso la interface DepartamentoBoRemote sólo define cuatro métodos.

- El método listaDepartamentos que devuelve la información de todos los departamentos de la base de datos.
- El método de búsqueda por *primary key* findDepartamento.
- newDepartamento, que se encarga de crear un nuevo departamento.
- El método de utilidad keysDepartamento que se utilizará para la generación de formularios en la aplicación web.

El código de la interface se muestra en el listado 8 y el código del EJB en el listado 9. Como se puede observar, las técnicas de programación utilizadas en este EJB son similares a las utilizadas en EmpleadoBo.

Listado 8: Código de la interface DepartamentoBoRemote.java.

```

1 package es.uv.bdsb.bo;
2
3 import java.util.List;
4 import java.util.TreeMap;
5 import javax.ejb.Remote;
6
7 import es.uv.bdsb.dto.Departamento;
8
9 @Remote
10 public interface DepartamentoBoRemote {
11     public List<Departamento> listaDepartamentos();
12     public Departamento findDepartamento(int id);
13     public void newDepartamento(String nombre, int manager);
14     public TreeMap<String, Integer> keysDepartamento();
15 }

```

Listado 9: Código de la clase DepartamentoBo.java.

```

1 package es.uv.bdsb.bo;
2
3 import java.sql.Connection;
4 import java.sql.SQLException;
5 import java.util.List;
6 import java.util.TreeMap;
7 import javax.annotation.PostConstruct;
8 import javax.annotation.PreDestroy;
9 import javax.annotation.Resource;
10 import javax.ejb.Stateless;
11 import com.sun.appserv.jdbc.DataSource;
12

```

```
13 import es.uv.bds.dao.MySQLDepartamentoDao;
14 import es.uv.bds.dto.Departamento;
15 import es.uv.bds.util.FinalString;
16
17 /**
18  * Session Bean implementation class DepartamentoBo
19  */
20 @Stateless(mappedName = "DepartamentoBO")
21 public class DepartamentoBo implements DepartamentoBoRemote {
22     @Resource(lookup = "jdbc/personalpool")
23     private DataSource ds;
24     private Connection con;
25     private MySQLDepartamentoDao ddao = null;
26
27     @PostConstruct
28     public void inicia() {
29         try {
30             con = ds.getConnection();
31             ddao = new MySQLDepartamentoDao(con);
32         }
33         catch (SQLException e) {
34             System.out.println("DepartamentoBo: Error en PostConstruct");
35             e.printStackTrace();
36         }
37     }
38     @PreDestroy
39     public void finaliza() {
40         try {
41             con.close();
42         }
43         catch (SQLException e) {
44             System.out.println("DepartamentoBo: Error en PreDestroy");
45             e.printStackTrace();
46         }
47     }
48
49     /**
50     * Default constructor.
51     */
52     public DepartamentoBo() {}
53
54     @Override
55     public List<Departamento> listaDepartamentos() {
56         List<Departamento> departamentos = null;
57         try {
58             departamentos = ddao.getAllDepartamentos();
59         }
60         catch (SQLException e) {
61             System.out.println("DepartamentoBo: Error en listaDepartamentos");
62         };
63         e.printStackTrace();
64     }
65     return departamentos;
66 }
67
68 @Override
69 public Departamento findDepartamento(int id) {
70     Departamento departamento = null;
71     try {
72         departamento = ddao.findById(id);
73     }
74     catch (SQLException e) {
75         System.out.println("DepartamentoBo: Error en findDepartamento");
76         e.printStackTrace();
77     }
78     return departamento;
79 }
```

```

80
81 @Override
82 public void newDepartamento(String nombre, int manager) {
83     Departamento nuevo = new Departamento();
84     nuevo.setNombre(FinalString.arregla(nombre));
85     nuevo.setManager(manager);
86     try {
87         ddao.persist(nuevo);
88     }
89     catch (SQLException e) {
90         System.out.println("DepartamentoBo: Error en newDepartamento");
91         e.printStackTrace();
92     }
93 }
94
95 @Override
96 public TreeMap<String, Integer> keysDepartamento() {
97     TreeMap<String, Integer> keyDepto = new TreeMap<String, Integer>
98     ();
99     try {
100         keyDepto = ddao.keysDepartamento();
101     }
102     catch (SQLException e) {
103         System.out.println("DepartamentoBo: Error en keyDepartamento");
104         e.printStackTrace();
105     }
106     return keyDepto;
107 }
108 }

```

## 5 Creación de la aplicación web dinámica

La última fase del desarrollo del programa consiste en crear una aplicación web dinámica que interactuará con los objetos EmpleadoBo y DepartamentoBo a través de un servlet para leer y escribir datos en la base de datos.

Para crear el proyecto, en el menú **File** de Eclipse seleccionamos **New** → **Dynamic Web Project** y creamos el proyecto jdbcPersonalWA como se muestra en la figura 4.

Una vez creado el proyecto, realizamos las siguientes operaciones:

1. Para que nuestro proyecto tenga acceso a los DTOs y los BOs, añadimos el proyecto jdbcPersonalEJB al *Build Path* del proyecto como se muestra en la figura 5.
2. Descomprima los ficheros suministrados en el fichero comprimido boletin\_t4b4\_ficheros\_WA.zip que acompaña a la práctica. Copie el contenido de la carpeta WebContent en su proyecto:

```
1 cp -r WebContent/* ~/workspace/jdbcPersonalWA/WebContent
```

y refresque el directorio de Eclipse (tecla **F5**).

### 5.1 El módulo de gestión de empleados

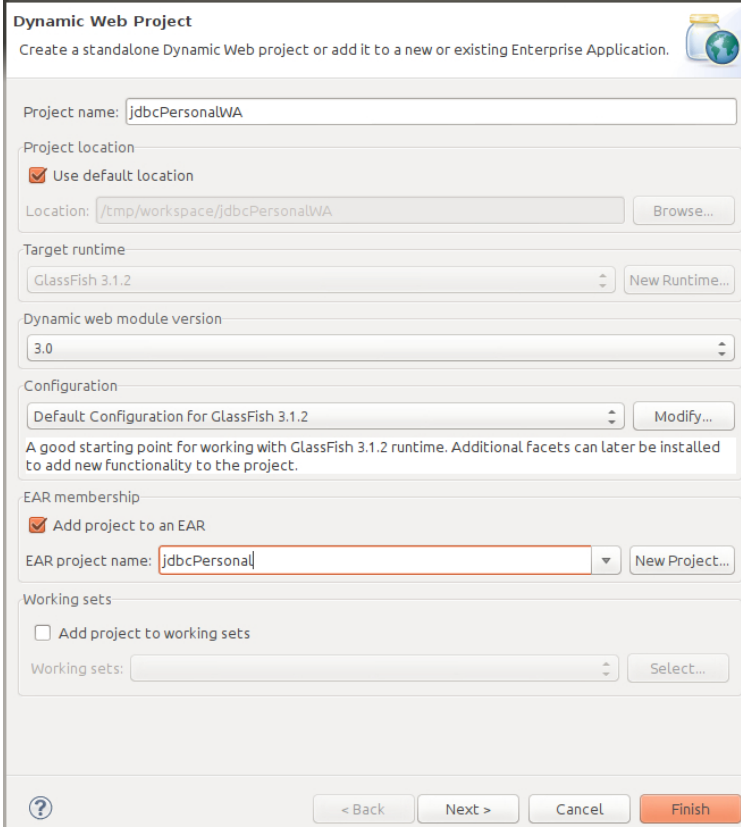
En primer lugar, crearemos el paquete `es.uv.bds.w.servlet` en el que implementaremos los servlet. Dentro del paquete crearemos los diferentes *servlets* que implementan los casos de uso de la gestión de empleados y a los que se podrá acceder desde el menú principal de la aplicación. El menú asociado a los empleados contempla los siguientes casos de uso:

- Mostrar la lista de todos los empleados.
- Añadir un nuevo empleado, para lo que presenta un formulario solicitando la información del nuevo empleado.
- Buscar empleado por apellidos.
- Mostrar la lista de empleados adscritos a un departamento.

#### 5.1.1 Mostrar la lista de todos los empleados

En el diagrama 6 se representa el diagrama de secuencia de este caso de uso. En el listado 10 se proporciona el código del servlet que coordina la acción.





**Dynamic Web Project**

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: jdbcPersonalWA

Project Location

☒ Use default location

Location: /tmp/workspace/jdbcPersonalWA [Browse...](#)

Target runtime

GlassFish 3.1.2 [New Runtime...](#)

Dynamic web module version

3.0

Configuration

Default Configuration for GlassFish 3.1.2 [Modify...](#)

A good starting point for working with GlassFish 3.1.2 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☒ Add project to an EAR

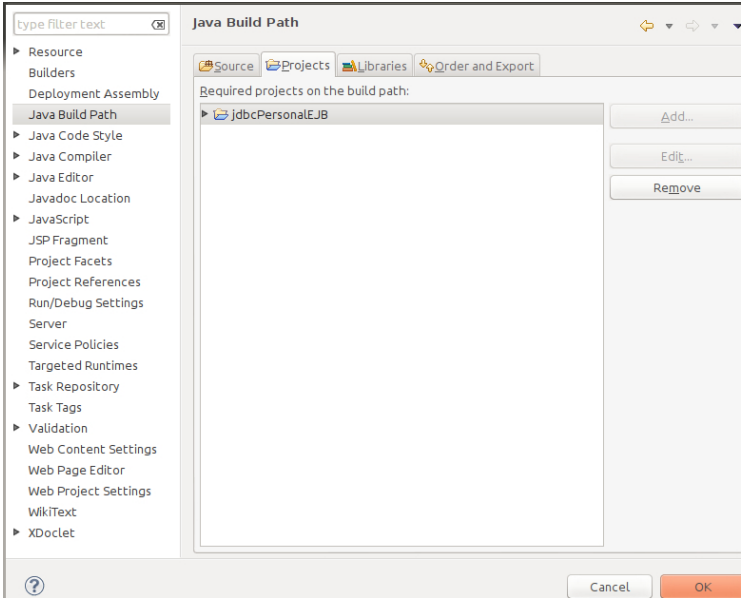
EAR project name: jdbcPersonal [New Project...](#)

Working sets

☐ Add project to working sets

Working sets: [Select...](#)

[? < Back](#) [Next >](#) [Cancel](#) [Finish](#)

Figura 4: Panel de creación del *Dynamic Web Project* jdbcPersonalWA.

type filter text

- Resource
- Builders
- Deployment Assembly
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- JavaScript
- JSP Fragment
- Project Facets
- Project References
- Run/Debug Settings
- Server
- Service Policies
- Targeted Runtimes
- Task Repository
- Task Tags
- Validation
- Web Content Settings
- Web Page Editor
- Web Project Settings
- WikiText
- XDoclet

**Java Build Path**

Source Projects Libraries Order and Export

Required projects on the build path:

- jdbcPersonalEJB

[Add...](#) [Edit...](#) [Remove](#)

[? Cancel](#) [OK](#)

Figura 5: Panel de configuración del *Build Path* del proyecto jdbcPersonalWA.

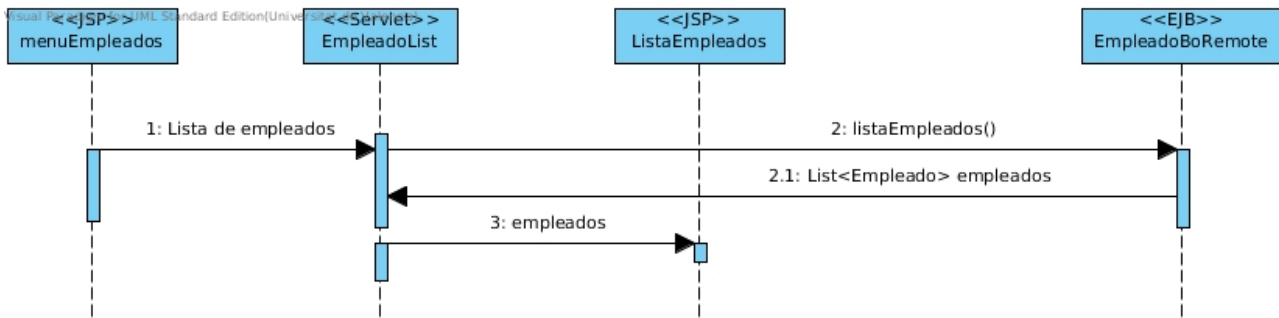


Figura 6: Diagrama de secuencia del caso de uso que obtiene la lista de todos los empleados.

Listado 10: Código del *servlet* EmpleadoList.java.

```

1 package es.uv.bds.w.servlet;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 import javax.ejb.EJB;
7 import javax.servlet.ServletException;
8 import javax.servlet.annotation.WebServlet;
9 import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12
13 import es.uv.bds.w.bo.EmpleadoBoRemote;
14 import es.uv.bds.w.dto.Empleado;
15
16 /**
17  * Servlet implementation class EmpleadoList
18  */
19 @WebServlet("/EmpleadoList")
20 public class EmpleadoList extends HttpServlet {
21     private static final long serialVersionUID = 1L;
22     @EJB(mappedName = "EmpleadoBo")
23     private EmpleadoBoRemote empleadoBO;
24
25     /**
26      * @see HttpServlet#HttpServlet()
27      */
28     public EmpleadoList() {
29         super();
30     }
31
32     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33         throws ServletException, IOException {
34         List<Empleado> empleados;
35
36         empleados = empleadoBO.listaEmpleados();
37         request.setAttribute("empleados", empleados);
38         request.getRequestDispatcher("/empleados/listaEmpleados.jsp").forward(request, response);
39     }
40
41     /**
42      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
43      */
44     protected void doGet(HttpServletRequest request, HttpServletResponse response)
45         throws ServletException, IOException {
46         processRequest(request, response);
47     }
48
49     /**
50      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

```

```

51  */
52  protected void doPost(HttpServletRequest request, HttpServletResponse response)
53  throws ServletException, IOException {
54      processRequest(request, response);
55  }
56  }

```

### 5.1.2 Añadir un nuevo empleado

La figura 7 muestra el diagrama de secuencia del caso de uso añadir un nuevo empleado. En el listado 11 se proporciona el código del servlet que coordina la acción.

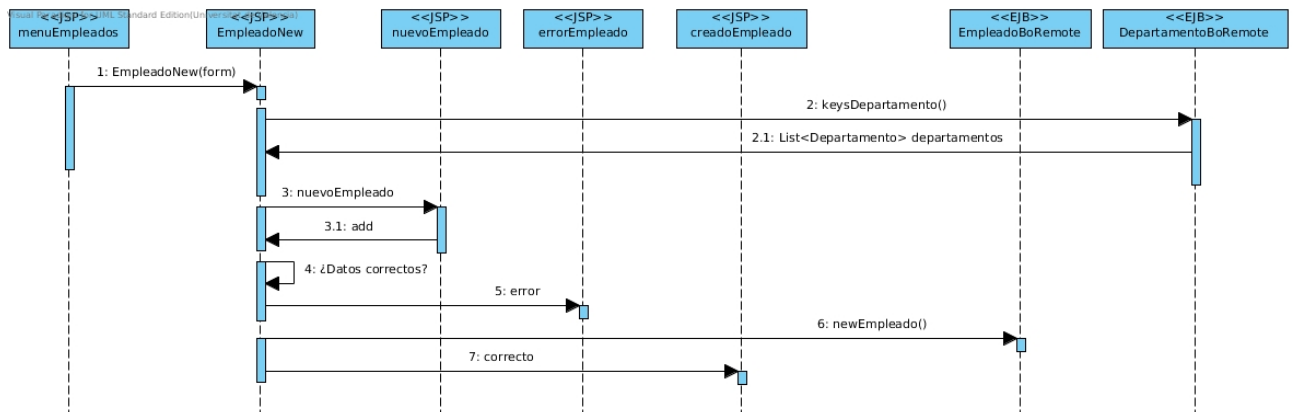


Figura 7: Diagrama de secuencia del caso de uso que obtiene la lista de todos los empleados.

Listado 11: Código del *servlet* EmpleadoNew.java.

```

1  package es.uv.bdsweb.servlet;
2
3  import java.io.IOException;
4  import java.sql.Date;
5  import java.util.TreeMap;
6
7  import javax.ejb.EJB;
8  import javax.servlet.ServletException;
9  import javax.servlet.annotation.WebServlet;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13
14 import es.uv.bdsweb.bo.DepartamentoBoRemote;
15 import es.uv.bdsweb.bo.EmpleadoBoRemote;
16
17 /**
18  * Servlet implementation class EmpleadoNew
19  */
20 @WebServlet("/EmpleadoNew")
21 public class EmpleadoNew extends HttpServlet {
22     private static final long serialVersionUID = 1L;
23     @EJB(mappedName = "EmpleadoBo")
24     private EmpleadoBoRemote empleadoBO;
25     @EJB(mappedName = "DepartamentoBo")
26     private DepartamentoBoRemote departamentoBO;
27
28     /**
29      * @see HttpServlet#HttpServlet()
30      */
31     public EmpleadoNew() {
32         super();
33     }
34

```

```

35 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
36     String nombre, apellidos, puesto, ano, mes, dia;
37     Date date;
38     Short nivelEd = 0;
39     float sueldo = 0, complemento = 0;
40     int depto = 0;
41     TreeMap<String, Integer> departamentos;
42     Boolean error = false;
43
44     String accion = request.getParameter("accion");
45     switch (accion) {
46
47         case "form":
48             departamentos = departamentoBO.keysDepartamento();
49             request.setAttribute("departamentos", departamentos);
50             request.getRequestDispatcher("/empleados/nuevoEmpleado.jsp").forward(request, response);
51             break;
52
53         case "add":
54             // Recogemos los áparmetros y comprobamos si son ávlidos
55             nombre = request.getParameter("nombre");
56             apellidos = request.getParameter("apellidos");
57             puesto = request.getParameter("puesto");
58             ano = request.getParameter("ano");
59             mes = request.getParameter("mes");
60             dia = request.getParameter("dia");
61             date = Date.valueOf(ano + "-" + mes + "-" + dia);
62             try {
63                 nivelEd = Short.parseShort(request.getParameter("nivelEducacion"));
64                 sueldo = Float.parseFloat(request.getParameter("sueldo"));
65                 complemento = Float.parseFloat(request.getParameter("complemento"));
66                 depto = Integer.parseInt(request.getParameter("depto"));
67             }
68             catch (NumberFormatException e) {
69                 error = true;
70             }
71             if ((nombre == null) ||
72                 (apellidos == null) ||
73                 (puesto == null) || error) {
74                 System.out.println("Error en el empleado...");
75                 request.getRequestDispatcher("/empleados/errorEmpleado.jsp").forward(request,
76                     response);
77             }
78             else {
79                 // Construimos y salvamos el nuevo empleado
80                 empleadoBO.newEmpleado(nombre, apellidos, puesto, date, nivelEd, sueldo, complemento
81                     , depto);
82                 request.getRequestDispatcher("/empleados/creadoEmpleado.jsp").forward(request,
83                     response);
84             }
85             break;
86
87         default:
88             throw new ServletException("óAccin no reconocida o no especificada");
89     }
90 }
91
92 /**
93  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
94  */
95 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    processRequest(request, response);
}

```

```

96  /**
97  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
98  */
99  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
100     processRequest(request, response);
101 }
102 }

```

### 5.1.3 Buscar empleados por apellidos

La figura 8 muestra el diagrama de secuencia del caso de uso añadir un nuevo empleado. En el listado 12 se proporciona el código del servlet que coordina la acción.

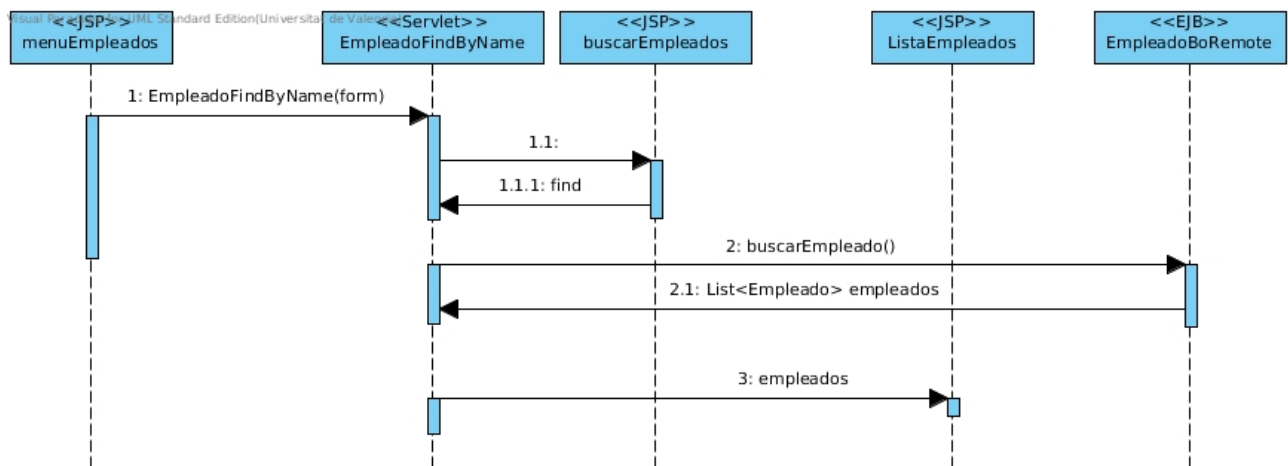


Figura 8: Diagrama de secuencia del caso de que permite buscar empleados por sus apellidos.

Listado 12: Código del servlet EmpleadoFindByName.java.

```

1  package es.uv.bdsweb.servlet;
2
3  import java.io.IOException;
4  import java.util.List;
5
6  import javax.ejb.EJB;
7  import javax.servlet.ServletException;
8  import javax.servlet.annotation.WebServlet;
9  import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12
13 import es.uv.bdsweb.bo.DepartamentoBoRemote;
14 import es.uv.bdsweb.bo.EmpleadoBoRemote;
15 import es.uv.bdsweb.dto.Empleado;
16
17 /**
18 * Servlet implementation class EmpleadoFindByName
19 */
20 @WebServlet("/EmpleadoFindByName")
21 public class EmpleadoFindByName extends HttpServlet {
22     private static final long serialVersionUID = 1L;
23     @EJB(mappedName = "EmpleadoBo")
24     private EmpleadoBoRemote empleadoBO;
25     @EJB(mappedName = "DepartamentoBo")
26     private DepartamentoBoRemote departamentoBO;
27
28     /**
29     * @see HttpServlet#HttpServlet()
30     */

```

```

31 public EmpleadoFindByName() {
32     super();
33 }
34
35 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
36     throws ServletException, IOException {
37     List<Empleado> empleados;
38     String apellidos;
39
40     String accion = request.getParameter("accion");
41     switch (accion) {
42         case "form":
43             request.getRequestDispatcher("/empleados/buscarEmpleados.jsp").forward(request,
44                 response);
45             break;
46         case "find":
47             apellidos = request.getParameter("apellidos");
48             empleados = empleadoBO.buscarEmpleado(apellidos);
49             request.setAttribute("empleados", empleados);
50             request.getRequestDispatcher("/empleados/listaEmpleados.jsp").forward(request,
51                 response);
52             break;
53         default:
54             throw new ServletException("óAccin no reconocida o no especificada");
55     }
56 }
57
58 /**
59  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
60  */
61 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
62     ServletException, IOException {
63     processRequest(request, response);
64 }
65
66 /**
67  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
68  */
69 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
70     ServletException, IOException {
71     processRequest(request, response);
72 }

```

#### 5.1.4 Buscar empleados por departamento

La figura 9 muestra el diagrama de secuencia del caso de uso añadir un nuevo empleado. En el listado 12 se proporciona el código del servlet que coordina la acción.

Listado 13: Código del *servlet* EmpleadoFindByDept.java.

```

1 package es.uv.bdsweb.servlet;
2
3 import java.io.IOException;
4 import java.util.List;
5 import java.util.TreeMap;
6
7 import javax.ejb.EJB;
8 import javax.servlet.ServletException;
9 import javax.servlet.annotation.WebServlet;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13

```

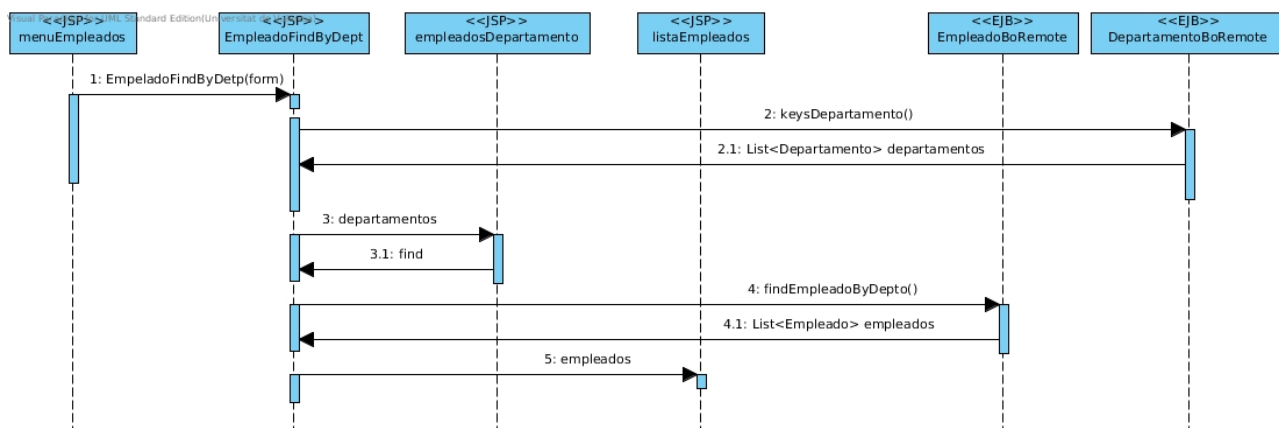


Figura 9: Diagrama de secuencia del caso de que permite buscar empleados adscritos a un departamento.

```

14 import es.uv.bdsb.bo.DepartamentoBoRemote;
15 import es.uv.bdsb.bo.EmpleadoBoRemote;
16 import es.uv.bdsb.dto.Empleado;
17
18 /**
19  * Servlet implementation class EmpleadoFindByDept
20  */
21 @WebServlet("/EmpleadoFindByDept")
22 public class EmpleadoFindByDept extends HttpServlet {
23     private static final long serialVersionUID = 1L;
24     @EJB(mappedName = "EmpleadoBo")
25     private EmpleadoBoRemote empleadoBO;
26     @EJB(mappedName = "DepartamentoBo")
27     private DepartamentoBoRemote departamentoBO;
28
29     /**
30     * @see HttpServlet#HttpServlet()
31     */
32     public EmpleadoFindByDept() {
33         super();
34     }
35
36     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
37         throws ServletException, IOException {
38         List<Empleado> empleados;
39         int depto = 0;
40         TreeMap<String, Integer> departamentos;
41
42         String accion = request.getParameter("accion");
43         // Bucle de óseleccin de la óaccin
44         switch (accion) {
45             case "form":
46                 departamentos = departamentoBO.keysDepartamento();
47                 request.setAttribute("departamentos", departamentos);
48                 request.getRequestDispatcher("/empleados/empleadosDepartamento.jsp").forward(request,
49                     response);
50                 break;
51             case "find":
52                 depto = Integer.parseInt(request.getParameter("depto"));
53                 empleados = empleadoBO.findEmpleadosByDepto(depto);
54                 request.setAttribute("empleados", empleados);
55                 request.getRequestDispatcher("/empleados/listaEmpleados.jsp").forward(request,
56                     response);
57                 break;
58             default:
  
```

```

59         throw new ServletException("óAccin no reconocida o no especificada");
60     }
61 }
62
63 /**
64  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
65  */
66 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
67     processRequest(request, response);
68 }
69
70 /**
71  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
72  */
73 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
74     processRequest(request, response);
75 }
76 }

```

## 5.2 El módulo de gestión de departamentos

También crearemos los *servlet* necesarios para gestionar los departamentos. Siguiendo el mismo procedimiento explicado en el caso anterior, cree los *servlets* cuyo código se muestra a continuación y cuya misión es gestionar las operaciones que nuestra aplicación realiza sobre los departamentos. El código es bastante más sencillo ya que el número de operaciones implementadas es menor:

- Mostrar la lista de todos los departamentos.
- Añadir un nuevo departamento.

### 5.2.1 Mostrar la lista de todos los departamentos

Este caso de uso es muy similar al caso de uso que muestra la lista de todos los empleados. En el listado 14 se muestra el código del *servlet* que controla los elementos implicados.

Listado 14: Código del *servlet* DepartamentoList . java.

```

1 package es.uv.bdsweb.servlet;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 import javax.ejb.EJB;
7 import javax.servlet.ServletException;
8 import javax.servlet.annotation.WebServlet;
9 import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12
13 import es.uv.bdsweb.bo.DepartamentoBoRemote;
14 import es.uv.bdsweb.dto.Departamento;
15
16 /**
17  * Servlet implementation class DepartamentoList
18  */
19 @WebServlet("/DepartamentoList")
20 public class DepartamentoList extends HttpServlet {
21     private static final long serialVersionUID = 1L;
22     @EJB(mappedName = "DepartamentoBo")
23     private DepartamentoBoRemote departamentoBO;
24
25     /**
26      * @see HttpServlet#HttpServlet()
27      */
28     public DepartamentoList() {

```



```

29     super();
30 }
31
32 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33     throws ServletException, IOException {
34     List<Departamento> departamentos;
35     departamentos = departamentoB0.listaDepartamentos();
36     request.setAttribute("departamentos", departamentos);
37     request.getRequestDispatcher("/departamentos/listaDepartamentos.jsp").forward(request,
38         response);
39 }
40
41 /**
42  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
43  */
44 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
45     ServletException, IOException {
46     processRequest(request, response);
47 }
48
49 /**
50  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
51  */
52 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
53     ServletException, IOException {
54     processRequest(request, response);
55 }
56 }

```

### 5.2.2 Añadir un nuevo departamento

En el listado 15 se muestra el código del servlet que controla la creación de un nuevo departamento. La secuencia de acciones es similar a la mostrada para el caso de uso añadir empleado, por lo que como ejercicio adicional, puede intentar dibujar el diagrama de secuencia asociado a este caso de uso.

Listado 15: Código del *servlet* DepartamentoNew. java.

```

1 package es.uv.bdsweb.servlet;
2
3 import java.io.IOException;
4 import java.util.TreeMap;
5
6 import javax.ejb.EJB;
7 import javax.servlet.ServletException;
8 import javax.servlet.annotation.WebServlet;
9 import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12
13 import es.uv.bdsweb.bo.DepartamentoBoRemote;
14 import es.uv.bdsweb.bo.EmpleadoBoRemote;
15
16 /**
17  * Servlet implementation class DepartamentoNew
18  */
19 @WebServlet("/DepartamentoNew")
20 public class DepartamentoNew extends HttpServlet {
21     private static final long serialVersionUID = 1L;
22     @EJB(mappedName = "EmpleadoBo")
23     private EmpleadoBoRemote empleadoBo;
24     @EJB(mappedName = "DepartamentoBo")
25     private DepartamentoBoRemote departamentoBo;
26
27     /**
28      * @see HttpServlet#HttpServlet()
29      */

```

```

30 public DepartamentoNew() {
31     super();
32 }
33
34 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
35     TreeMap<String, Integer> empleados;
36     String nombre;
37     int manager;
38
39     String accion = request.getParameter("accion");
40     switch (accion) {
41         case "form":
42             empleados = empleadoB0.keysEmpleado();
43             request.setAttribute("empleados", empleados);
44             request.getRequestDispatcher("/departamentos/nuevoDepartamento.jsp").forward(request,
                response);
45             break;
46
47         case "add":
48             nombre = request.getParameter("nombre");
49             manager = Integer.parseInt(request.getParameter("manager"));
50             if (nombre == null) {
51                 System.out.println("Error en el departamento...");
52                 request.getRequestDispatcher("/departamentos/errorDepartamento.jsp").forward(request,
                    response);
53             }
54             else {
55                 departamentoB0.newDepartamento(nombre, manager);
56                 request.getRequestDispatcher("/departamentos/creadoDepartamento.jsp").forward(
                    request, response);
57             }
58             break;
59
60         default:
61             throw new ServletException("DepartamentoNew: acción desconocida o no especificada.");
62     }
63 }
64
65 /**
66  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
67  */
68 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
69     processRequest(request, response);
70 }
71
72 /**
73  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
74  */
75 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
76     processRequest(request, response);
77 }
78 }

```

**Nota:**

¡Enhorabuena! Ha completado el ejercicio casi en su totalidad. Prepárese para disfrutar del resultado...

### 5.3 Desplegar y ejecutar la aplicación

Ahora sólo queda desplegar la aplicación en el servidor de aplicaciones y ejecutar el servlet. Para ello hacemos click con el botón derecho sobre el fichero `index.jsp` en la ventana del *Project Explorer*. En el menú emergente seleccionamos

**Run As** → **Run on Server**. En el panel de configuración del servidor de aplicaciones, seleccione su servidor Glassfish local. En las figuras 10 y 11 se muestran dos ejemplos de ejecución de la aplicación.

## 6 Ejercicio

El objetivo de este ejercicio es ampliar nuestra aplicación web con el siguiente caso de uso:

- **Detalles de empleado**, que implica:
  1. El sistema solicitará al usuario un código de empleado.
  2. El sistema buscará el empleado con el código introducido.
  3. Si el usuario existe, se mostrará un panel con toda la información asociada al usuario.
  4. Si el usuario no existe, se mostrará una pantalla informativa notificando la incidencia.
- **Cambiar manager**, cuyo proceso es:
  1. El usuario selecciona un departamento de una lista desplegable.
  2. El sistema muestra los datos del manager actual y la posibilidad de seleccionar un nuevo manager a partir de una lista desplegable.
  3. Si el usuario modifica el manager, el sistema deberá reflejar los cambios en la base de datos.



Principal

Empleados

Departamentos

Proyectos

Menú

Lista de empleados

Nuevo empleado

Buscar por apellidos

Buscar por departamento

Lista de Empleados

Código	Nombre	Puesto	Antigüedad	Departamento
21	ALFARO VIDAL, DAVID	ANL	2005-10-23	4
28	ALGARIN LOPEZ, FRANCISCO LAZARO	DES	2006-12-05	3
49	ALMENAR BALLESTER, MARIA	DES	2005-03-18	3
13	ANGULLO GARCIA, JUAN	DES	2005-06-19	3
14	ANGULO SANCHEZ, REBECA	FLD	2004-09-04	4
44	ANSON FERRER, FRANCISCO JAVIER	FLD	2005-08-19	2
55	ARANDIGA SANCHEZ, CARLOS	FLD	2007-11-20	4
59	BARBERA RUIZ, JOSE IGNACIO	MGR	2004-02-04	5
23	BENET TORAN, EDUARD	MGR	2007-07-05	5
33	BOLUDA ALVAREZ, RAFAEL	SLS	2004-04-02	2
51	BOU VILAR, DAVID	PRES	2007-06-22	2
20	CASCALES ABAD, LUCIA	SLS	2005-11-18	4
30	COLLADO LOPEZ, JOAQUIN	DES	2006-08-23	2
47	CRUZ, ANGEL	DES	2004-01-09	1
41	EDO LORENTE, SERGIO	SLS	2005-03-12	4
36	ESTEVE CARBO, CHRISTOPHE	SLS	2007-01-03	2
29	FUSTER CAMPOS, ELVIRA	SLS	2005-07-09	4
11	GANDIA BARROSO, ANA ISABEL	ANL	2004-08-25	4
38	GARCIA FONFRIA, ZULEMA	SLS	2006-04-06	4
62	GARCIA SOLER, ADOLFO JOSE	DES	2004-08-24	3
7	GIL BAILEN, JORGE	FLD	2006-07-15	3
12	HENNCHEN BELENGUER, ROSA ARANCHA	MGR	2004-10-04	2
3	IVARS SORIANO, PABLO	MGR	2006-07-06	4
56	JORNET ALOCEN, TOBIAS	FLD	2006-01-09	2
4	LLOPIS BARBERA, SERGIO	FLD	2005-05-10	4
52	LLOREDA BOU, ENRIQUE CARLOS	MGR	2006-05-08	3
22	LLUNA IRANZO, FERNANDO AGUSTIN	PRES	2006-01-04	2
1	LOPEZ NAVARRO, JURGEN	SLS	2006-06-21	1
53	LOPEZ PENALVA, ENRIQUE	ANL	2007-09-02	3
5	LOPEZ TROYANO, ALEJONSO CHRISTIAN	DES	2004-11-26	2

Figura 10: Lista de empleados tal y como se presenta a través del navegador.

**ENVY Inc.**

Principal Empleados Departamentos Proyectos

**Menú**

- Lista de empleados
- Nuevo empleado
- Buscar por apellidos
- Buscar por departamento

**Alta de empleados**

Nombre:

Apellidos:

Puesto: Comercial

Fecha contratación: 01 : 01 : 2006 (dd-mm-yyyy).

Titulación:

Sueldo:

Complemento:

Departamento: CENTRO DE INFORMACION

Crear empleado Limpiar formulario

@ 2013 - Departament d'Informàtica. Universitat de València. Mon Feb 25 12:46:24 CET 2013

Figura 11: Vista desde el navegador del formulario para añadir un nuevo empleado