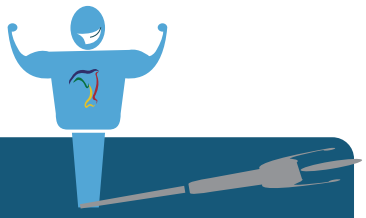
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

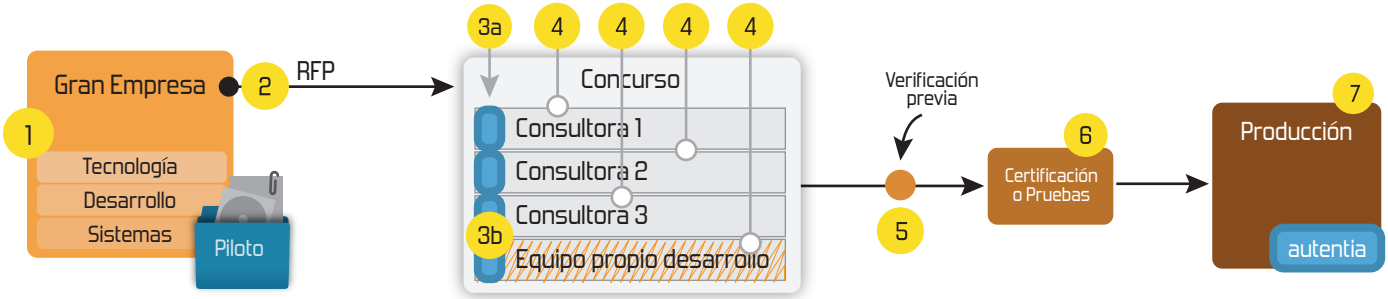
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

DESARROLLADO POR:
Alberto Barranco Ramón

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Fecha de publicación del tutorial: 2009-02-26 8

Ejemplo de uso con JSF 2.0, Primefaces e Hibernate

0. Índice de contenidos.

1. Enunciado del ejemplo

En este ejemplo vamos a suponer que somos unos cuantos amigos, a los que nos gustan los videojuegos y queremos mantener una lista actualizada de los juegos que nos compramos. Los datos que nos interesan son: una fotografía, el nombre, una descripción y el precio. En un principio queremos mostrar todos los juegos que tenemos y poder añadir nuevos juegos a medida que nos los vamos comprando y que de forma dinámica se añadan a la lista.

2. Tecnología que vamos a utilizar

Para este ejemplo, aunque sencillo, vamos a usar el patrón de diseño: modelo-vista-controlador.

- Modelo: Para mantener los datos de los videojuegos, vamos a utilizar como gestor de bases de datos MySQL 5.1. Vamos a acceder a estos mediante Hibernate 3.2.5.
- Vista: Para pintar los videojuegos en pantalla y permitir que se agreguen otros nuevos vamos a utilizar JSF 2.0 y Primefaces 2.1.
- Controlador: En controlador tendremos nuestra lógica de negocio. El IDE que vamos a utilizar es Netbeans en su versión 6.8. Además como servidor utilizaremos GlassFish 3.

3. Empecemos con el Modelo

Lo decía Jack "el destripador" (vayamos por partes) y todos los profesores de algorítmica (divide y vencerás). Para afrontar un problema informático hay que dividirlo en partes más sencillas.

3.1 Instalación y configuración de Hibernate

Lo primero de todo es crear una base de datos para almacenar los datos de los videojuegos. Si no tenemos MySQL instalado podemos bajarlo de la página web oficial: http://www.mysql.com/downloads/. En la instalación es importante que nos quedemos con el puerto. En este ejemplo se ha dejado el puerto por defecto (3306). A continuación:

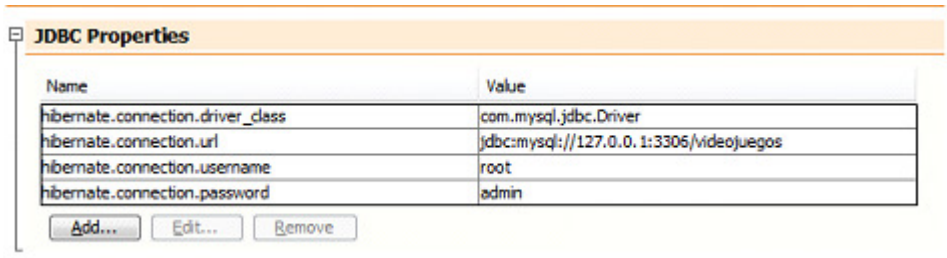
- 1. Nos creamos una Base de datos de nombre videojuegos.
- 2. Creamos una tabla de nombre juegos con los siguientes atributos: id, nombre, sinapsis, ruta (para guardar la ruta de la foto o carátula del juego) y precio.

Ahora vamos a configurar Hibernate para que pueda acceder a la base de datos que nos acabamos de crear. En Netbeans contamos con una interfaz visual bastante amigable aparte del XML.

Para una conexión directa con la base de datos serían necesarios los siguientes parámetros:

- Driver para conexión con mysql.
- La url compuesta por:
  - 1. Dirección: 127.0.0.1 ó podríamos poner también localhost.
  - 2. El puerto. Como no lo cambiamos sigue siendo el puerto por defecto 3306.
  - 3. El nombre de la base de datos.
- El usuario: root.
- La contraseña: admin.

Si nos fijamos en el archivo hibernate.cfg.xml, las propiedades quedarían de la siguiente forma.

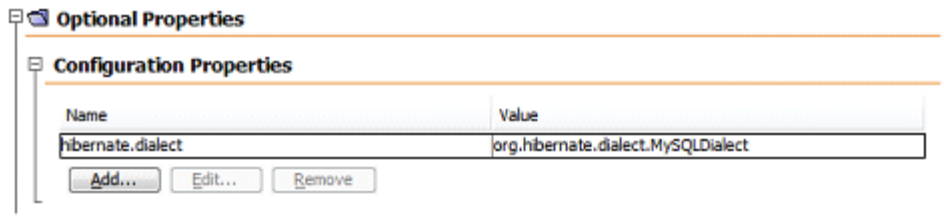


En el XML tendría su correspondencia:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http
3  <hibernate-configuration>
4    <session-factory>
5      <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
6      <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
7      <property name="hibernate.connection.url">jdbc:mysql://127.0.0.1:3306/videojuegos</property>
8      <property name="hibernate.connection.username">root</property>
9       <property name="hibernate.connection.password">admin</property>
```

Nota: es muy recomendable que la contraseña no sea admin, sino algo más seguro.

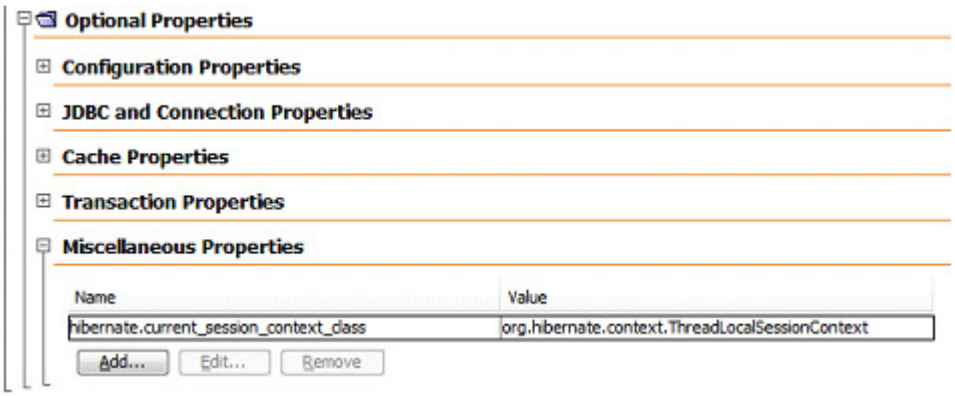
Si nos fijamos en la línea 5 de nuestro XML tenemos el dialecto que es el correspondiente a MySQL. En la parte gráfica estaría dentro de Optional Properties en la subsección Configuration Properties.



Vamos a añadir una propiedad más a la configuración de Hibernate. Queremos que Hibernate linkee la session al hilo correspondiente de manera automática, por lo que añadimos lo siguiente:

```
12  <property name="hibernate.current_session_context_class">org.hibernate.context.ThreadLocalSessionContext
```

En la interfaz gráfica esta propiedad puede verse dentro de Optional Properties en la subsección Miscellaneus Properties.



En este momento podemos conectar Hibernate con nuestra base de datos en MySQL. Ahora necesitamos una clase que tenga los mismos atributos que nuestra tabla juegos, para poder guardar los datos que escribimos o leemos de la base de datos. A estas clases se las conoce como POJO.

3.2 Creación del POJO

POJO o Plain Old Java Object, son clases que no dependen del framework, es decir, no heredan interfaces ni implementan clases del framework, que en este caso es Hibernate. Diríamos entonces que Hibernate es un framework no intrusivo. A la vez, Hibernate exige que las clases sean Bean, es decir, clases simples. Estas clases no tienen constructores con parámetros y se caracterizan por tener sus atributos marcados como privados, y una serie de métodos conocidos como getters y setters para leer o escribir dichos atributos. Vamos a llamar a nuestra nueva clase VideojuegoPojo que será de la siguiente forma.

```
10  public class VideojuegoPojo{
11
12
13      private int id;
14      private String nombre;
15      private String ruta;
16      private String sinapsis;
17      private float precio;
18
19
20      public int getId() {
21          return id;
22      }
23
24      public String getNombre() {
25          return nombre;
26      }
27
28      public String getRuta() {
29          return ruta;
30      }
```

Los setters deberían quedarnos como los siguientes.

```
52      public void setSinapsis(String sinapsis) {
53          this.sinapsis = sinapsis;
54      }
55
56      public void setPrecio(float precio) {
57          this.precio = precio;
58      }
59  }
60
```

3.3 Mapeo entre la base de datos y nuestra clase POJO

Ya tenemos configurada la conexión a la Base de Datos y tenemos nuestra clase Pojo para guardar información. Ahora solo nos queda hacer corresponder los atributos de la clase con los distintos campos de la tabla juegos. A esto se le llama mapping.

Para hacerlo, en nuestro package actual, pincharemos en New, y en Hibernate Mapping Wizard. Esto nos creará un archivo al que nosotros hemos llamado VideojuegoPojo.hbm.xml.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0/
3  <hibernate-mapping>
4    <class name="VideojuegoPojo" table="juegos">
5      <!-- Hibernate genera mediante hilo una clave de 4 bytes de
6           tipo entero que es la primary key de juegos. -->
7      <id access="field" column="id" name="id" type="java.lang.Integer">
8        <generator class="org.hibernate.id.TableHiLoGenerator">
9          <param name="table">jid</param>
10         <param name="column">next</param>
11       </generator>
12     </id>
```

- En la línea 4 estamos fijando el nombre de la clase (VideojuegoPojo) que queremos hacer corresponder con una tabla de la base de datos (juegos).
- Si nos fijamos en las líneas 7-12, lo que estamos haciendo es generar la clave primaria de la tabla juegos, que se llama id, y es de tipo integer. Lo hacemos mediante el algoritmo Hilo que proporciona Hibernate. Este algoritmo genera un número único, que vale para claves primarias, a partir de una tabla auxiliar en la base de datos que se encarga de mantener el siguiente número a aplicar el algoritmo. Esta tabla en nuestro ejemplo es jid, que tendría un solo campo que es next, y que hemos inicializado en un principio a 1.

Nota: Podemos poner el campo id, que es primary key de la tabla juegos con AUTOINCREMENT. Si le damos esa propiedad cuando creamos la tabla juegos, no necesitaremos generar nosotros la clave.

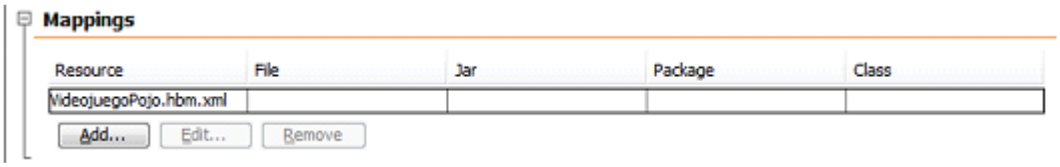
Los demás atributos irían mapeados de la siguiente forma:

```
13       <!-- Un juego debe tener un nombre, de 30 caracteres a lo sumo. -->
14       <property access="field" name="nombre" type="java.lang.String">
15         <column length="30" name="nombre" not-null="true"/>
16       </property>
17       <property access="field" name="ruta" type="java.lang.String">
18         <column name="ruta"/>
19       </property>
20       <property access="field" name="sinapsis" type="java.lang.String">
21         <column name="sinapsis"/>
22       </property>
23       <property access="field" name="precio" type="java.lang.Float">
24         <column name="precio"/>
25       </property>
26     </class>
27   </hibernate-mapping>
```

Por último tan solo nos queda hacer constancia en la configuración de hibernate de este mapeo. Lo podemos hacer incluyendo en el archivo hibernate.cfg.xml la siguiente línea.

```
14       <!-- Archivo de mapeo -->
15       <mapping resource="VideojuegoPojo.hbm.xml"/>
```

En la parte gráfica se verían reflejados todos los mapeos en la pestaña de Mappings.



3.4 Creándonos nuestro DAO

DAO (Data Access Object) va a ser un componente mediante el cual accederemos a los datos. Creándonos el DAO, nuestra capa de negocio no necesitará conocer el destino final de la información que maneja. Aunque está considerado una buena práctica, si estamos en entornos donde el rendimiento es una prioridad no deberíamos utilizarlo.

Lo que queremos obtener para el ejemplo que estamos tratando es una lista de videojuegos con todos los juegos que tiene la base de datos, así como poder introducir uno nuevo. Estos métodos deberían ser parecidos a lo siguiente:

```
41      /*
42       * Método getListaJuegos.
43       * Descripción: Devuelve una lista de VideojuegoPojo, que contiene todos
44       * los videojuegos que hay en la base de datos.
45       */
46      static List<VideojuegoPojo> getListaJuegos() {
47          List<VideojuegoPojo> listaJuegos = new ArrayList<VideojuegoPojo>();
48          Session session = ConfHibernate.getSessionFactory().getCurrentSession();
49          try {
50              session.beginTransaction();
51              Criteria criteria = session.createCriteria(VideojuegoPojo.class);
52              listaJuegos = criteria.list();
53              session.getTransaction().commit();
54          } catch (Exception ex) {
55              Logger.getLogger(Controlador.class.getName()).log(Level.SEVERE,"Problema
56              session.getTransaction().rollback();
57          }
58          return listaJuegos;
59      }
```

Si nos fijamos a partir de la línea 49 tenemos el modo estándar de hacer una transacción.

- 1. Empezamos la transacción.
- 2. Hacemos las operaciones en la base de datos.
- 3. Hacemos el Commit.
- 4. Si algo ha ido mal lo notificamos en el Logger y hacemos un Rollback para que la base de datos vuelva al mismo estado previo a las operaciones.

En la línea 48 vemos que está implicada la clase ConfHibernate. Esto no es más que el Session Factory. Está detallado a continuación.

```
4   /**********************************************************************************
5    * Class ConfHibernate:                                                           *
6    * Descripción: Esta se trata del sessionFactory. Gracias a esta clase vamos      *
7    * a asegurar que solo hay una única instancia de session mediante el patrón      *
8    * de diseño singleton.                                                           *
9    * ********************************************************************************/
10  public class ConfHibernate {
11
12      private static final SessionFactory sessionFactory;
13
14      static {
15          try {
16              // Creamos la Sesión para hibernate.cfg.xml
17              sessionFactory = new Configuration().configure().buildSessionFactory();
18          } catch (Throwable ex) {
19              // Aviso de fallo al crear la sesión
20              System.err.println("Initial SessionFactory creation failed." + ex);
21              throw new ExceptionInInitializerError(ex);
22          }
23      }
24
25      public static SessionFactory getSessionFactory() {
26          return sessionFactory;
27      }
28  }
```

NOTA: Tenemos solo una session, por eso esta clase no tiene constructor. Cuando queramos usar la session lo haremos mediante la llamada getCurrentSession(); Al finalizar deberemos cerrar la session, en este ejemplo en el destructor del DAO.

Ya solo nos quedaría la parte necesaria para introducir un juego nuevo en la base de datos. Completaríamos nuestro DAO con el siguiente método.

```
61      /*
62       * Método introducirVideojuegoPojo.
63       * Descripción: Método de clase, que se encarga de introducir los datos
64       * de un videojuego en la base de datos.
65       * Parámetros de entrada: Nombre, ruta, sinapsis y precio del videojuego a introducir
66       */
67      public static void introducirVideojuegoPojo(String nombre, String ruta, String sinapsis, float preci
68          Session session = ConfHibernate.getSessionFactory().getCurrentSession();
69          VideojuegoPojo miVideojuego = new VideojuegoPojo();
70          miVideojuego.setNombre(nombre);
71          miVideojuego.setRuta(ruta);
72          miVideojuego.setSinapsis(sinapsis);
73          miVideojuego.setPrecio(precio);
74          try {
75              session.beginTransaction();
76              session.save(miVideojuego);
77              session.getTransaction().commit();
78          } catch (Exception ex) {
79              Logger.getLogger(Controlador.class.getName()).log(Level.SEVERE,"Problema guardar el juego nu
80              session.getTransaction().rollback();
81          }
82      }
```

4. Creando la Vista y el Controlador

La vista va a estar dividida en dos partes. En una primera parte vamos a tener los videojuegos y debajo vamos a dar la opción de introducir uno nuevo. Para mostrar los videojuegos nos vamos a servir de Primefaces, ya que sus componentes nos van a resultar muy útiles. Para poder utilizarlos primero vamos a tener que configurar el archivo web.xml, para añadir los filtros necesarios para subir ficheros, en donde podremos indicar filtros para los tamaños de los ficheros, y el Servlet de Primefaces. Web.xml nos quedaría de la siguiente forma.

General Servlets Filters Pages References Security XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>

    <filter>
        <filter-name>PrimeFaces FileUpload Filter</filter-name>
        <filter-class>
            org.primefaces.webapp.filter.FileUploadFilter
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>PrimeFaces FileUpload Filter</filter-name>
        <servlet-name>Faces Servlet</servlet-name>
    </filter-mapping>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <!-- Servlet de Primefaces -->
    <servlet>
        <servlet-name>Resource Servlet</servlet-name>
        <servlet-class>org.primefaces.resource.ResourceServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Resource Servlet</servlet-name>
        <url-pattern>/primefaces_resource/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>

</web-app>
```

Ahora ya podemos usar Primefaces en nuestro index.xhtml. Para la primera parte vamos a usar un componete de Primefaces llamado dataTable. Este componente se rellena con una lista de objetos, que van a ser nuestros videojuegos, de la siguiente forma.

```
12      <!--Tabla con la lista de juegos -->
13      <p:graphicText value="Lista de Juegos" fontName="Broadway" fontSize="26" style="margin-left: 60px" /
14      <!-- Componenta dataTable id=table, se rellena con los datos de listaVideojuegos, atributo de la cla
15      TableVideojuegoPojo -->
16      <p:dataTable id="table" value="#{tableVideojuegoPojo.listaVideojuegos}" var="videojuegoPojo" paginat
17                   rows="5" dynamic="true" >
18          <p:column>
19              <f:facet name="header">
20                  <h:outputText value="Juego"/>
21              </f:facet>
22              <p:graphicImage value="#{dao.carpeta}#{videojuegoPojo.ruta}" />
23              <p:graphicText value="#{videojuegoPojo.nombre} " fontName="Arial" fontSize="18" style="margi
24          </p:column>
25          <p:column>
26              <f:facet name="header">
27                  <h:outputText value="Sinapsis"/>
28              </f:facet>
29              <h:outputText value="#{videojuegoPojo.sinapsis}"/>
30          </p:column>
```

NOTA: Primefaces es un Api con más de 90 componentes para JSF que podeis descargaros de http://www.primefaces.org/downloads.html.

Podemos observar en la línea 16 como la tabla se va a pintar a partir de un atributo de la clase TableVideojuegoPojo, que se llama listaVideojuegos. La clase TableVideojuegoPojo sería pues nuestro controlador, encargado de pedir al DAO los datos, y si fuera necesario modificarlos o hacer cualquier tipo de tratamiento antes de mostrarlos en la vista. En la línea 22 si nos fijamos estamos construyendo la dirección de la foto a pintar teniendo en cuenta la carpeta que contiene DAO y la ruta de la foto. Esto se detalla más adelante. Nuestro controlador es muy simple en este caso.

```
8     * TableVideojuegoPojo es la clase encargada de solicitar la lista de videojuegos
9     * al DAO, mediante el método getListaVideojuegos
10    */
11  @ManagedBean
12  public class TableVideojuegoPojo implements Serializable {
13
14      //Atributo listaVidejojuegos: Contiene la lista de videojuegos existentes
15      // en la base de datos.
16      private List<VideojuegoPojo> listaVideojuegos;
17
18
19      /*
20       * Método getListaVideojuegos: Invoca el método getListaJuegos() de la clase
21       * Dao para obtener la lista de videojuegos, cuyo valor se guarda en
22       * listaVideojuegos.
23       */
24      public List<VideojuegoPojo> getListaVideojuegos() {
25          listaVideojuegos = Dao.getListaJuegos();
26          return listaVideojuegos;
27      }
28  }
```

Quizás lo más importante sea la línea 11 @ManagedBean, con lo que estamos indicando que la vista va a tener acceso a sus atributos, precisamente mediante los getters y setters.

Para la segunda parte simplemente vamos a dar la opción al usuario de rellenar los datos de un nuevo videojuego, subir una foto y darlo de alta en la base de datos. Después debería actualizarse la tabla para mostrarlo. Nos quedaría algo parecido a lo siguiente.

```
38  <!-- Inserción de nuevo juego -->
39      <p:graphicText value="Insertar Juego Nuevo" fontName="Broadway" fontSize="26" style="margin-bottom:
40          <h:panelGrid columns="2" >
41              <p:graphicText value="Nombre" fontName="Arial" fontSize="18" style="margin-left: 10px" />
42              <h:inputText id="inputNombre" value="#{nuevoJuegoBean.nombre}" required="true"/>
43              <p:graphicText value="Sinapsis" fontName="Arial" fontSize="18" style="margin-left: 10px" />
44              <h:inputTextarea id="inputSinapsis" value="#{nuevoJuegoBean.sinapsis}"/>
45              <p:graphicText value="Precio" fontName="Arial" fontSize="18" style="margin-left: 10px" />
46              <h:inputText id="inputPrecio" value="#{nuevoJuegoBean.precio}"/>
47              <p:graphicText value="Foto" fontName="Arial" fontSize="18" style="margin-left: 10px" />
48              <p:fileUpload id="imagenUpload"
49                            fileUploadListener="#{nuevoJuegoBean.accionFileUpload}"
50                            allowTypes="*.jpg;*.png;*.gif" description="Imágenes"/>
51              <h:outputLabel/>
52          </h:panelGrid>
53      <p:commandButton value="Guardar Juego" actionListener="#{nuevoJuegoBean.grabarJuego}" update="table"
```

Como vemos, los datos que introduce el usuario los vamos a ir guardando en la clase nuevoJuegoBean, que también estaría actuando de controlador. Le estaríamos a su vez encargando la subida de la foto como se puede ver en la línea 49, así como grabar el juego (nuevoJuegoBean.grabarJuego) y actualizar la tabla (update="table"), acciones ambas contempladas en la línea 53. Vamos ahora a ver cómo podemos grabar todos los atributos de un juego en una misma instancia de nuevoJuegoBean, gestionar la foto y grabarlo todo en la base de datos.

```
14  @ManagedBean
15  @ViewScoped
16  public class NuevoJuegoBean {
17
18      private String nombre;
19      private String sinapsis;
20      private float precio;
21      private String ruta;
22
23      public void setNombre(String nombre) {
24          this.nombre = nombre;
25      }
26
27      public void setSinapsis(String sinapsis) {
28          this.sinapsis = sinapsis;
29      }
30
31      public void setPrecio(float precio) {
32          this.precio = precio;
33      }
```

Un punto importante es la línea 15. Con @ViewScoped lo que estamos indicando es que todos los parámetros que introdujo el usuario en el form anterior se van a guardar en la misma instancia de NuevoJuegoBean, y se va a gestionar la foto y se va a subir con los atributos que se han introducido. Si no lo pusiéramos se nos crearían varias instancias de NuevoJuegoBean y cuando fuéramos a grabar el juego no podríamos ver el Nombre, la sinapsis, la descripción y la ruta de la foto. Igual que vemos en la foto algunos setters, estarían los getters.

```
55          /*
56           * Método grabarJuego: Este método ejecuta una llamada a
57           * Dao.introducirVideojuegoPojo pasándole los atributos
58           * nombre, ruta, sinapsis y precio que son propios de esta clase.
59           */
60          public void grabarJuego() {
61              Dao.introducirVideojuegoPojo(nombre,ruta,sinapsis,precio);
62          }
63
64          public void accionFileUpload (FileUploadEvent event) {
65              UploadedFile file = event.getFile();
66              ruta = file.getFileName();
67              Dao.gestionarFotoSubida(file.getContents(), ruta);
68          }
69
70      }
```

En cuanto a grabarJuego, simplemente tendríamos que pasarle al DAO los datos indicados por el usuario. De esta forma estamos ganando abstracción. A este nivel no sabemos donde se van a guardar esos datos, el DAO se encarga de ello. En esta línea va también el método que hay a continuación, el accionFileUpload. Este método simplemente se encarga de quedarse con la ruta de la foto, que es lo que se guarda en la base de datos, y le encarga al DAO la administración de la misma. Así desde este nivel no conocemos la carpeta o directorio donde se va a guardar la misma, ganando en abstracción. Vamos ahora a completar nuestro DAO que sí tendrá esas direcciones tanto virtuales como físicas donde se guardan las imágenes y a su método gestionarFotoSubida.

La carpeta virtual sería /Imagenes/ y la carpeta física para este ejemplo sería C:\\Users\\Alberto\\Documents\\NetBeansProjects\\Gamer\\build\\web\\Imagenes\\ y es donde guardaremos físicamente la foto que manda el usuario de la siguiente forma.

```
92          public static void gestionarFotoSubida(byte[] datos, String nombre) {
93              File file = new File(carpetaFisica+nombre);
94              try {
95                  file.createNewFile();
96                  FileOutputStream fout = new FileOutputStream(file);
97                  fout.write(datos);
98                  fout.close();
99              } catch (IOException ex) {
100                 Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, "Carpeta: "+Syst
101             }
102         }
```

Con esto tendríamos completo el DAO.

5. Conclusiones

Después de haber probado la librería he podido observar las siguientes ventajas:

- Primefaces cuenta con un buen número de componentes (más de 90) que resultan estéticamente muy agradables.
- Primefaces es compatible con JSF 1.2 en su versión 1, y compatible con JSF 2.0 en su versión 2.
- Soporta Ajax sin que el desarrollador tenga que ver o hacer casi nada. Ya hemos visto que con indicar que el componente dataTable sea dinámico, y hacer un update de ese componente desde cualquier otro, tendríamos solucionado la actualización.
- Se trata de una librería en crecimiento con una comunidad muy activa. Cuentan con un foro de soporte bastante acitvo en donde resuelven dudas y reciben feedback de los usuarios para mejorar la librería y corregir bugs.

Primefaces me ha parecido una opción a tener muy en cuenta a la hora de trabajar con JSF.

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

» **Registrate** y accede a esta y otras ventajas «

COMENTARIOS