

2009

# REUTILIZACIÓN Y DISEÑO DE PATRONES SOFTWARE

## PRÁCTICA FINAL: APLICACIÓN DE COMERCIO ELECTRÓNICO

Se pretende desarrollar una aplicación web de comercio electrónico, a modo de tienda virtual, con Servlets, JSP, Struts.

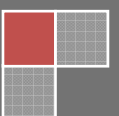
Luis Alberto Fernández Yélamos DNI: 03134085-J

correo: [luis.fernandez.y@gmail.com](mailto:luis.fernandez.y@gmail.com)

Fernando Serrano López DNI: 03129999-K

correo: [ferryv6@hotmail.com](mailto:ferryv6@hotmail.com)

19/05/2009



# ÍNDICE

|   |    |
|---|----|
| 1.INTRODUCCIÓN .....                                | 3  |
| 1.1.BASE DE DATOS .....                             | 5  |
| 2.PATRONES UTILIZADOS.....                          | 6  |
| 2.1.PATRONES DE CREACIÓN .....                      | 6  |
| 2.1.1.SINGLETON .....                               | 6  |
| 2.1.2.PROTOTYPE .....                               | 7  |
| 2.1.3.FACTORY METHOD.....                           | 9  |
| 2.2.PATRONES DE COMPORTAMIENTO .....                | 11 |
| 2.2.1.COMMAND.....                                  | 11 |
| 2.2.2.ITERATOR .....                                | 12 |
| 2.2.3.STRATEGY .....                                | 13 |
| 2.2.4.TEMPLATE METHOD.....                          | 15 |
| 2.3.PATRONES ESTRUCTURALES.....                     | 17 |
| 2.3.1.DECORATOR .....                               | 17 |
| 2.3.2.FACADE .....                                  | 19 |
| 2.3.3.MVC .....                                     | 20 |
| 2.4.PATRONES DE DISEÑO FUNDAMENTALES .....          | 22 |
| 3.DIAGRAMAS DE CLASES .....                         | 24 |
| 3.1.PAQUETE TIENDA.....                             | 24 |
| 3.2.PAQUETE ORDENACIÓN .....                        | 29 |
| 3.3.PAQUETE LOGIN .....                             | 30 |
| 3.4.PAQUETE DATOS.....                              | 33 |
| 4.DIAGRAMA DE COMPONENTES .....                     | 36 |
| 5.DIAGRAMAS DE CASOS DE USO .....                   | 37 |
| 5.1.CASO DE USO COMPRA .....                        | 37 |
| 5.2.CASO DE USO VISUALIZACIÓN DE PRODUCTOS .....    | 38 |
| 5.3.CASO DE USO MODIFICACIÓN DE DATOS .....         | 38 |
| 5.4.CASO DE USO MODIFICACIÓN ESTADO DE PEDIDOS..... | 38 |
| 5.5.CASO DE USO MODIFICACIÓN DE PRODUCTOS .....     | 39 |
| 5.6.CASO DE USO VISUALIZACIÓN DE CESTAS .....       | 39 |
| 5.7.CASO DE USO INTRODUCIR PRODUCTO .....           | 39 |
| 6.DIAGRAMAS DE SECUENCIA .....                      | 40 |
| 6.1.SECUENCIA COMPRA.....                           | 40 |
| 6.2. SECUENCIA VER PRODUCTO .....                   | 40 |
| 6.3.SECUENCIA INTRODUCIR PRODUCTO.....              | 41 |
| 6.4.SECUENCIA INTRODUCIR PRODUCTO.....              | 42 |
| 6.5. SECUENCIA MODIFICAR ESTADO PEDIDOS .....       | 42 |
| 6.6.SECUENCIA MODIFICACIÓN DE DATOS.....            | 43 |
| 6.7.SECUENCIA VER USUARIOS REGISTRADOS .....        | 43 |
| 6.8.SECUENCIA VER CESTAS .....                      | 44 |
| 7.INSTALACIÓN DE LA APLICACIÓN WEB.....             | 45 |

## 1. INTRODUCCIÓN

Se pretende diseñar una aplicación Web de comercio electrónico a modo de tienda virtual, con Servlets, JSP, Struts.

Dicha aplicación Web comprende una serie de funcionalidades que variarán en función del perfil del usuario que acceda a la aplicación. De este modo podemos asumir el papel de cliente o el de administrador del sistema.

Las funcionalidades del cliente son las siguientes:

- Acceder al catálogo de categorías y productos de la tienda. Dicho catálogo está almacenado en una base de datos de postgresql y constará de películas, videojuegos y música, cada uno con su propia tabla y atributos propios.
- Cesta de la compra. En dicha cesta, se irán acumulando los productos que el cliente vaya comprando. Una vez realizada la compra se procederá al checkout de la misma, lo que significa que no se pueden añadir más productos a la cesta.
- Sistema de seguimiento de pedidos. El cliente dispondrá de la información de los pedidos que ha realizado así como el estado de los mismos.
- Registro de clientes y modificación de datos de cliente. Se facilitará una interfaz para el registro de clientes, y otra para la modificación de los datos personales de los mismos.
- Búsqueda de productos. El sistema proporciona un sistema de búsqueda, que funcionará tanto por nombre de producto como por género de producto y categoría. Además se mostrarán los productos relacionados por género y precio al producto que se está visualizando.

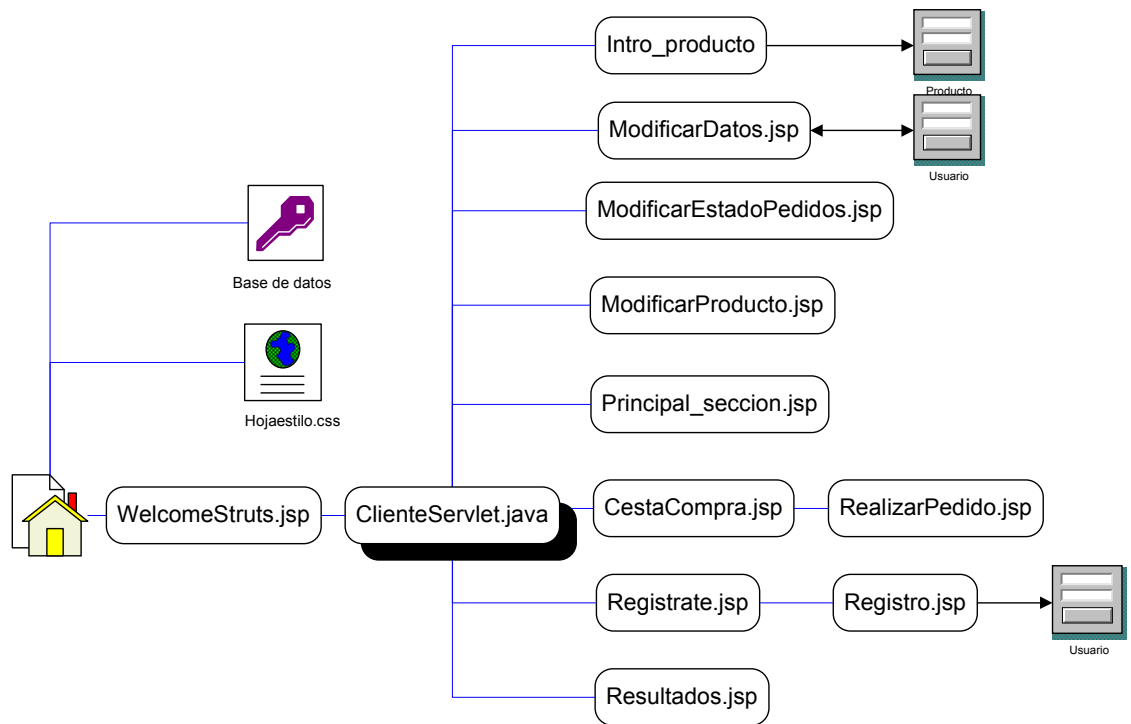
El administrador dispondrá de las mismas funcionalidades que el cliente, pero además dispondrá de una serie de funcionalidades específicas:

- Modificación del estado de los pedidos. El administrador modificara el estado de las cestas de la compra cuyo estado sea de checkout. Se podrá variar entre diferentes estados, como por ejemplo enviado o anulado.
- Modificación de los productos de la base de datos. Se dispondrá de una interfaz para la modificación de los atributos de los productos de la base de datos.
- Introducir productos en la base de datos. Se proporcionará una interfaz para introducir productos en la base de datos.
- Seguimiento de Usuarios y datos de usuarios. Se mostrará una estadística de los usuarios registrados en el sistema, así como el gasto acumulado de cada uno de ellos y las cestas de la compra relacionadas con cada uno de ellos.

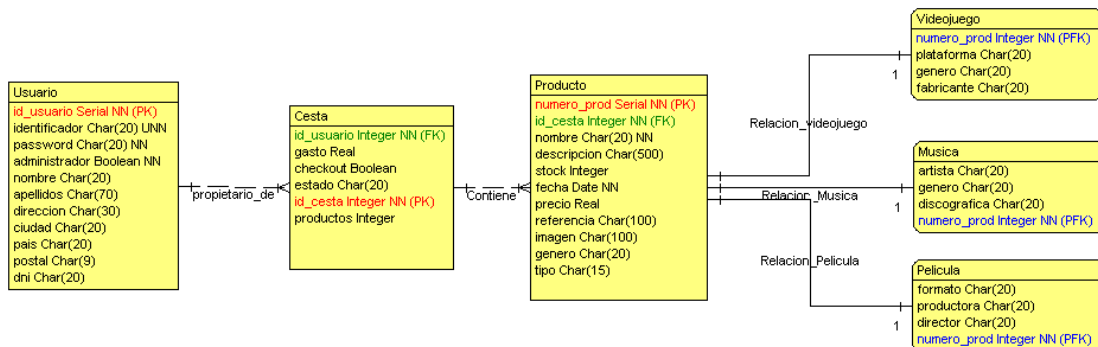
Además de lo anteriormente mencionado, el sistema solicitará la validación de los usuarios para poder realizar compras.

Como requisitos adicionales el sistema tendrá que validar correctamente sobre los estándares de la w3c.

El sistema será desarrollado utilizando el patrón modelo-vista-controlador de Struts.



## 1.1. BASE DE DATOS



La base de datos cuenta con 6 entidades:

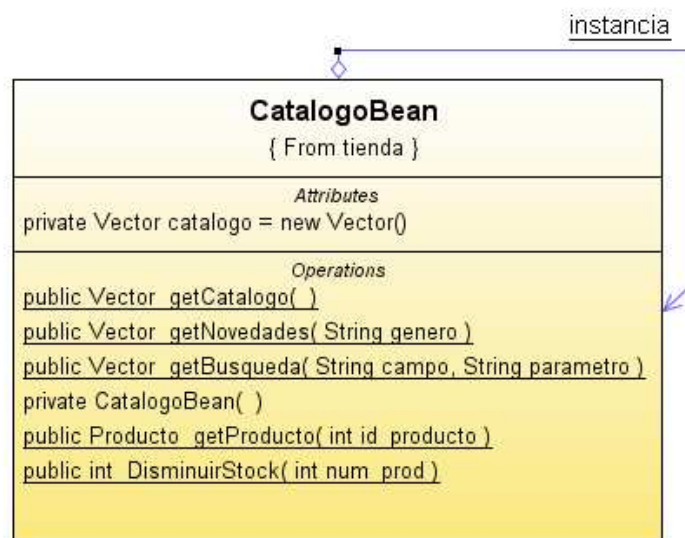
- Entidad Usuario: representa cada uno de los usuarios registrados para poder utilizar la aplicación web. Almacena información sobre los datos del usuario, como su nombre, apellidos, etc. Además es imprescindible que se establezca un valor para los campos identificador y contraseña, que serán los utilizados para logear a los usuarios. El booleano administrador, indica si el usuario es administrador del sistema.
- Entidad Cesta: representa cada una de las cestas que un usuario puede crear para realizar sus compras en la aplicación. Cada usuario puede ser propietario de ninguna o varias cestas. Se almacenará el gasto acumulado en ella, el estado de la cesta (ENVIADO, ANULADO, EN\_ENVIO, EN PROCESO), los productos almacenados en la misma( a través de un array de enteros, los cuales serán los identificadores de los productos), y un booleano que indica si la cesta está cerrada, esto es, si no se pueden añadir más productos.
- Entidad Producto: representa cada uno de los productos del catálogo de la tienda. Se almacena toda la información relevante del producto, como su nombre, descripción, imagen a mostrar (ruta relativa en la aplicación), etc. El campo tipo se utilizará para relacionar uno de los productos con una de las entidades Videojuego, Música, o Película.
- Entidad Videojuego: cada tupla de esta tabla está relacionada con un producto de la base de datos. Extiende la información de un producto determinado, cuyo campo tipo es igual a Videojuego.
- Entidad Música: cada tupla de esta tabla está relacionada con un producto de la base de datos. Extiende la información de un producto determinado, cuyo campo tipo es igual a Música.
- Entidad Película: cada tupla de esta tabla está relacionada con un producto de la base de datos. Extiende la información de un producto determinado, cuyo campo tipo es igual a Película.

## 2. PATRONES UTILIZADOS

### 2. PATRONES DE CREACIÓN

#### 2.1.1. SINGLETON

Objetivo: Garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a ella. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia. El resultado de aplicar este patrón es que solo puede existir al mismo tiempo una instancia de una clase, que será accedida a través de un punto de acceso.



Utilización: Dado que tenemos un catálogo muy extenso de productos en la tienda virtual, es interesante mantener un único catálogo donde se almacenen los diferentes productos, y que estos no tengan que ser leídos de la base de datos cada vez que se desee realizar un acceso a la base, y por lo tanto tengan que ser almacenados en una estructura nueva para cada acceso.

En la práctica, dichos productos se almacenan en una estructura de datos de tipo `java.util.Vector`.

```
private static CatalogoBean instancia=new CatalogoBean();

public static Vector getCatalogo()

{

    return(instancia.catalogo);

}
```

Como se puede observar, solo se puede crear una única instancia del objeto `CatalogoBean`. Para accesos posteriores se utilizará el método `getCatalogo()`, que devolverá la instancia de `CatalogoBean`, con nombre `instancia`.

### 2.1.2. PROTOTYPE

Objetivo: Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crea nuevos objetos copiando dicho prototipo.

Este patrón se usa en los casos en los que crear una instancia de una clase sea un proceso muy complejo y requiera mucho tiempo. Lo que hace es crear una instancia original, y cuando se necesite una nueva, en lugar de crearla, se copia esa original, y si es necesario se modifica.

El patrón se implementa a través del método Clone() de la clase Object, que realiza copias de objetos. Todas las clases heredan un método de la clase Object que retorna una copia de ese objeto, solamente si la instancia da permiso para ser clonada. Para ello es necesario implementar la interfaz cloneable.

|   |
|---|
| <b>Producto</b><br>{ From tienda }  |
| <i>Attributes</i>   |
| private int numero_prod = 0<br>private int id_cesta = 0<br>private String nombre = ""<br>private String descripcion = ""<br>private int stock = 0<br>private String fecha = ""<br>private float precio = 0<br>private String referencia = ""<br>private String imagen = ""<br>private String tipo = ""<br>private String genero = ""  |
| <i>Operations</i>   |
| public String getGenero( )<br>public void setGenero( String genero )<br>public int getId_cesta( )<br>public void setId_cesta( int id_cesta )<br>public String getFecha( )<br>public String toString( )<br>public Producto( int _numero_prod, int _id_cesta, String _nombre, String _descripcion, int _stock, String _fecha, float _precio, String _referencia, String _imagen, String _tipo )<br>public int getNumero_prod( )<br>public int getID_cesta( )<br>public String getNombre( )<br>public String getDescripcion( )<br>public int getStock( )<br>public String getReferencia( )<br>public float getPrecio( )<br>public String getImagen( )<br>public void setStock( int unidades )<br>public void setTipo( String _tipo )<br>public String getTipo( )<br>public Object Clone( ) |

Implementación: como se puede observar el objeto Producto implementa la interfaz Cloneable, y define un método Clone. Esto nos permitirá realizar una copia exacta y rápida del producto, sin tener que volver a invocar a cada uno de los métodos de un nuevo objeto producto, para establecer los valores de sus atributos.

```

public void guardarCompra(HttpServletRequest req)
{
    Vector prods=CatalogoBean.getCatalogo();
    String[] param=req.getParameterValues("numero_prod");
    if(param!=null)
    {
        for(int i=0;i<param.length;i++)
        {
            int id_producto=Integer.parseInt(param[i]);
            Producto p=null;
            for(int j=0;j<prods.size();j++)
            {
                p=(Producto)((Producto)prods.get(j)).Clone();
                if(p.getNumero_prod()==id_producto)
                productos.put(Integer.toString(p.getNumero_prod()),p);
            }
        }
    }
}

```

En el método guardarCompra de la clase CarroBean, se añaden los productos que se pasan como parámetros de la página. Para ello se obtienen los valores del parámetro numero\_prod, se busca en el catálogo el producto cuyo identificador coincida con uno de los numero\_prod extraídos. Posteriormente se clona dicho objeto y se añade al carro de la compra, representado por una tabla Hash denominada productos.



### 2.1.3. FACTORY METHOD

Objetivo: Define una interfaz para crear un objeto pero deja que sean las subclases quienes decidan que clase instanciar. Permite que una clase delegue en sus subclases la creación del objeto. Se utiliza este patrón cuando no se puede prever la clase de objetos que se debe instanciar, o una clase quiere que sean sus subclases quienes especifiquen los objetos que ésta crea.



Implementación: El método getOrdenacion devolverá una instancia de una de las tres clases de Estrategias de Ordenacion disponibles. La estructura de estas estrategias, así como su interfaz se definirá posteriormente, en la implementación del patrón Strategy.

El método GetOrdenacion es el siguiente:

```
public Estrategia getOrdenacion(String tipo)
{
    Estrategia estrategia=new EstrategiaNombre();

    if(tipo.equals(APELLIDOS))
        estrategia= new EstrategiaApellidos();
    else if(tipo.equals(NOMBRE))
        estrategia= new EstrategiaNombre();
    else if(tipo.equals(GASTO))
        estrategia= new EstrategiaGasto();

    return estrategia;
}
```

A modo de ejemplo, se muestra el uso de este patrón en la página que muestra los usuarios registrados en el sistema, y los ordena por el valor de uno de sus atributos.

```
String sortBy=(String)request.getParameter("sortBy");

Ordenacion.FactoriaDeOrdenacion fac=new Ordenacion.FactoriaDeOrdenacion();

if(sortby!=null)

    if (sortBy.contains("nombre")){

        Ordenacion.Estrategia estrategia=fac.getOrdenacion(fac.NOMBRE);

        registrados=estrategia.ordena(registrados);

    }

    else if (sortBy.contains("gasto")){

        Ordenacion.Estrategia estrategia=fac.getOrdenacion(fac.GASTO);

        registrados=estrategia.ordena(registrados);

    }

    else if (sortBy.contains("apellidos")){

        Ordenacion.Estrategia estrategia=fac.getOrdenacion(fac.APELLIDOS);

        registrados=estrategia.ordena(registrados);

    }

}
```

Como se puede observar será la clase FactoriaDeOrdenacion la que actúa como creador concreto, ya que redefine el método de fabricación para devolver una instancia de un ProductoConcreto. Estos productos concretos, serán instancias de las clases EstrategiaApellidos, EstrategiaNombre, o EstrategiaGasto.

## 2.2. COMPORTAMIENTO

### 2.2.1. COMMAND

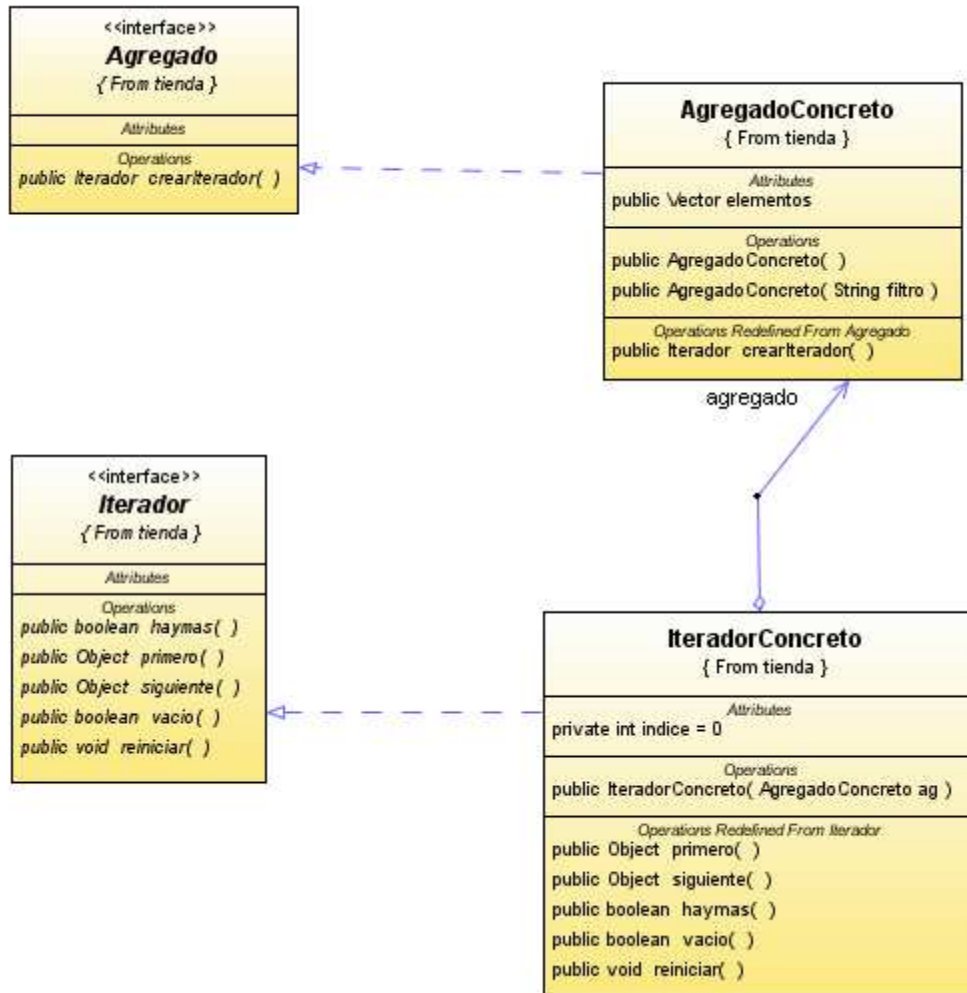
Objetivo: encapsular un comando en un objeto. Este objeto contiene el comportamiento y los datos necesarios para una acción específica. Permite parametrizar a los clientes con diferentes peticiones, hacer cola o llevar un registro de las peticiones. Se utiliza este patrón cuando se desea desacoplar la fuente de una petición del objeto que la cumple.

```
public class Conectar {  
    public static Connection ejecutar()  
    {  
        try{  
            Class.forName("org.postgresql.Driver");  
            Connection c;  
  
            c=DriverManager.getConnection("jdbc:postgresql://localhost:5432/DAW","postgres","postgres");  
  
            return c;  
        }catch(Exception ex){ex.printStackTrace();}  
        return null;  
    }  
}
```

La clase Conectar implementa la llamada al sistema postgresql para realizar la conexión con la base de datos. Esto facilita la modificación de dicha orden, ya que en caso de modificar los parámetros de conexión, por ejemplo, solo sería necesario reemplazar el código en la clase Conectar, ya que el resto de las clases no dependen de la implementación interna de esta.

### 2.2.2. ITERATOR

Objetivo: Proporcionar una forma coherente de acceder secuencialmente a los datos de una colección, independientemente del tipo de colección.

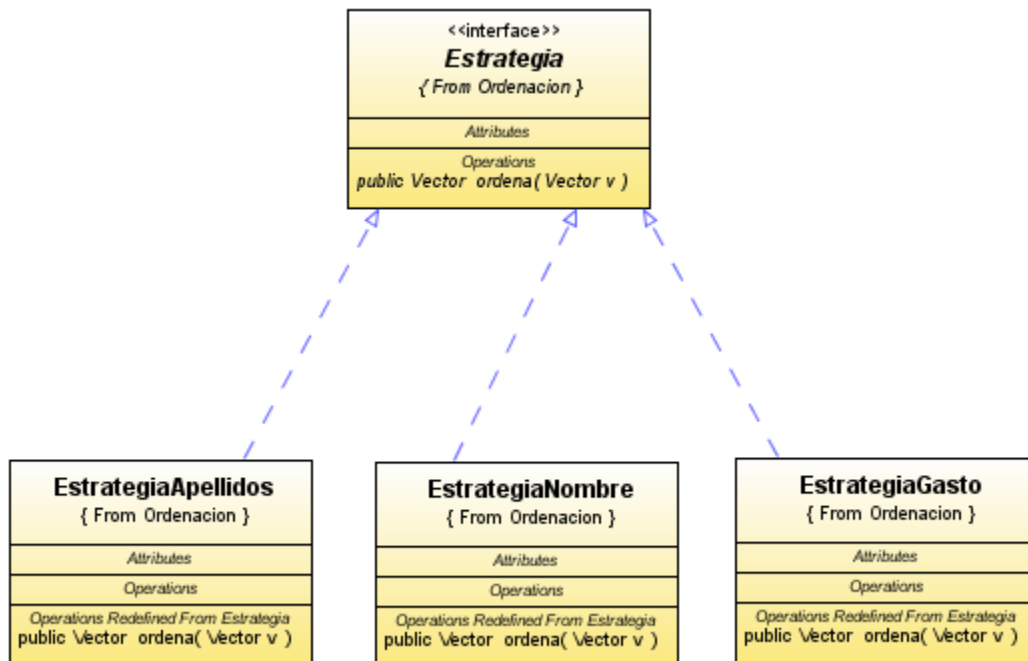


La clase `IteradorConcreto` implementa la interfaz `Iterador`. En ésta, se proporcionan los métodos `primero`, que devuelve el primer elemento del vector; `siguiente`, que devuelve el elemento siguiente del Vector e incrementa el índice de posición del mismo; `hayMas` indica si se ha alcanzado el final del Vector; `vacio`, indica si el Vector no contiene ningún elemento; y, por último, `reiniciar`, que vacía el Vector e inicializa el índice del Vector.

La clase `AgregadoConcreto` implementa la interfaz `Agregado` y se encarga de introducir en el Vector los objetos correspondientes a las cestas de compra que se encuentran almacenadas en la base de datos. Se puede observar que un segundo constructor toma como parámetro una cadena denominada `filtro`. Este constructor obtiene solamente las cestas que concuerdan con un estado determinado.

### 2.2.3. STRATEGY

Objetivo: Definir un grupo de clases que representan un conjunto de posibles comportamientos. Estos comportamientos pueden ser fácilmente intercambiados en una aplicación, modificando la funcionalidad en cualquier instante. Se utiliza cuando muchas clases relacionadas solo se diferencian en su comportamiento, o se necesitan diferentes variantes de un algoritmo.



Implementación: se proporcionan tres implementaciones de la interfaz Estrategia: EstrategiaApellidos, incluye la implementación del método abstracto ordena, que ordena los elementos del Vector por su atributo Apellidos; EstrategiaNombre, cuyo método abstracto ordena, realiza la ordenación de los elementos del Vector por su atributo Nombre; y EstrategiaGasto, que implementa el método abstracto ordena, a través de una ordenación basada en el atributo gasto, el cual indica el gasto total realizado en la tienda.

Se puede observar que estas clases solo difieren en la manera de ordenar los elementos del Vector, esto es, utilizan un algoritmo diferente para realizar la misma funcionalidad.

El método ordena para la clase EstrategiaNombre sería el siguiente:

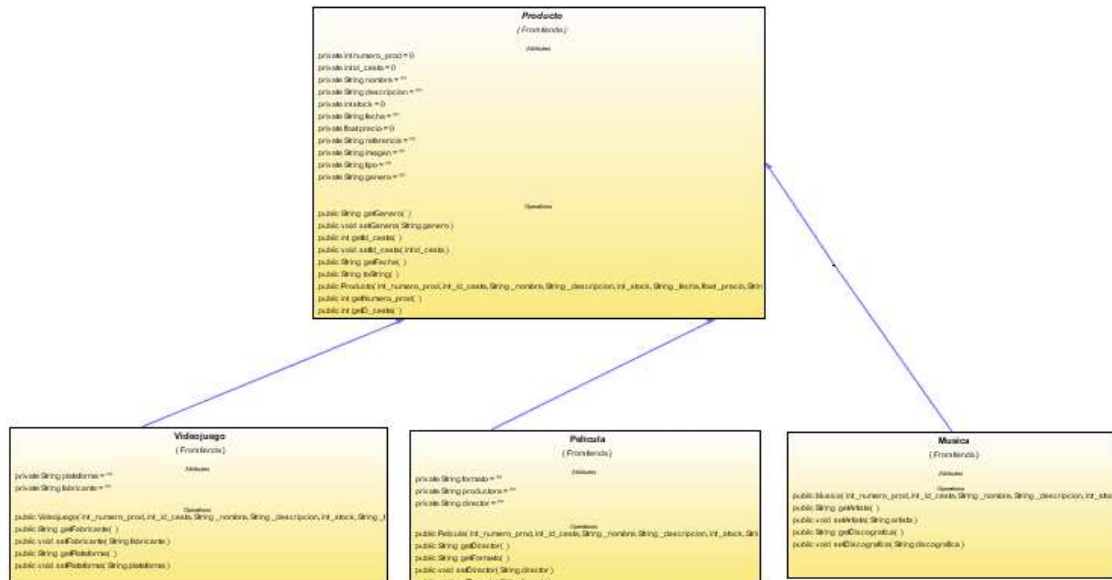
```
public Vector ordena(Vector v)
{
    Vector elementos=new Vector();
    Object[] vector=v.toArray();
    Comparator comparador=new Comparator()
    {
        public int compare(Object o1,Object o2)
        {
            login.Usuario usuario1=(login.Usuario)o1;
            login.Usuario usuario2=(login.Usuario)o2;

            return(usuario1.getNombre().compareTo(usuario2.getNombre()));

        }
    };
    Arrays.sort(vector,comparador);
    for(int i=0;i<vector.length;i++)
    {
        elementos.add(vector[i]);
    }
    return elementos;
}
```

## 2.2.4. TEMPLATE METHOD

Objetivo: Proporcionar un método que permita que las subclases redefinan partes del método sin reescribirlo. Se utiliza esta patrón cuando se desea centralizar partes de un método que se define en todos los subtipos de una clase, pero que siempre tienen una pequeña diferencia en cada subclase. Se utiliza este patrón cuando se desea evitar la duplicación de código entre clases de una jerarquía.



Implementación: se proporciona la clase abstracta Producto, que define una serie de atributos comunes a las tres subclases, y los métodos necesarios para establecer y recuperar dichos valores.

Las clases Videojuego, Pelicula y Música definen a su vez sus propios atributos (por ejemplo Plataforma y fabricante, en la clase videojuego), así como los métodos necesarios para modificarlos.

A través del parámetro tipo, de la clase Producto, sabremos cual de las tres subclases tenemos que instanciar. Un ejemplo de la página correspondiente a la modificación de productos sería el siguiente:

```
if(tipo.contains("videojuego"))
{
    tienda.Videojuego producto=(tienda.Videojuego)prod;
%>

<tr><td>

    Plataforma

</td>

<td>
```

```

        <input type="text" name="plataforma"
value="<%=producto.getPlataforma()%>"/>

    </td></tr>

    <tr><td>

        fabricante

    </td>

    <td >

        <input type="text" name="fabricante"
value="<%=producto.getFabricante()%>"/>

    </td></tr>

    <%>else if(tipo.contains("pelicula")){

        tienda.Pelicula producto=(tienda.Pelicula)prod;

        %>

    <tr><td >

        Formato

    </td>

    <td>

        <input type="text" name="formato" value="<%=producto.getFormato()%>"/>

    </td></tr><tr><td>

        Productora

    </td>

    <td>

        <input type="text" name="productora" value="<%=producto.getProductora()%>"/>

    </td></tr><tr><td>

        Director

    </td>

    <td>

        <input type="text" name="director" value="<%=producto.getDirector()%>"/>

    </td></tr>

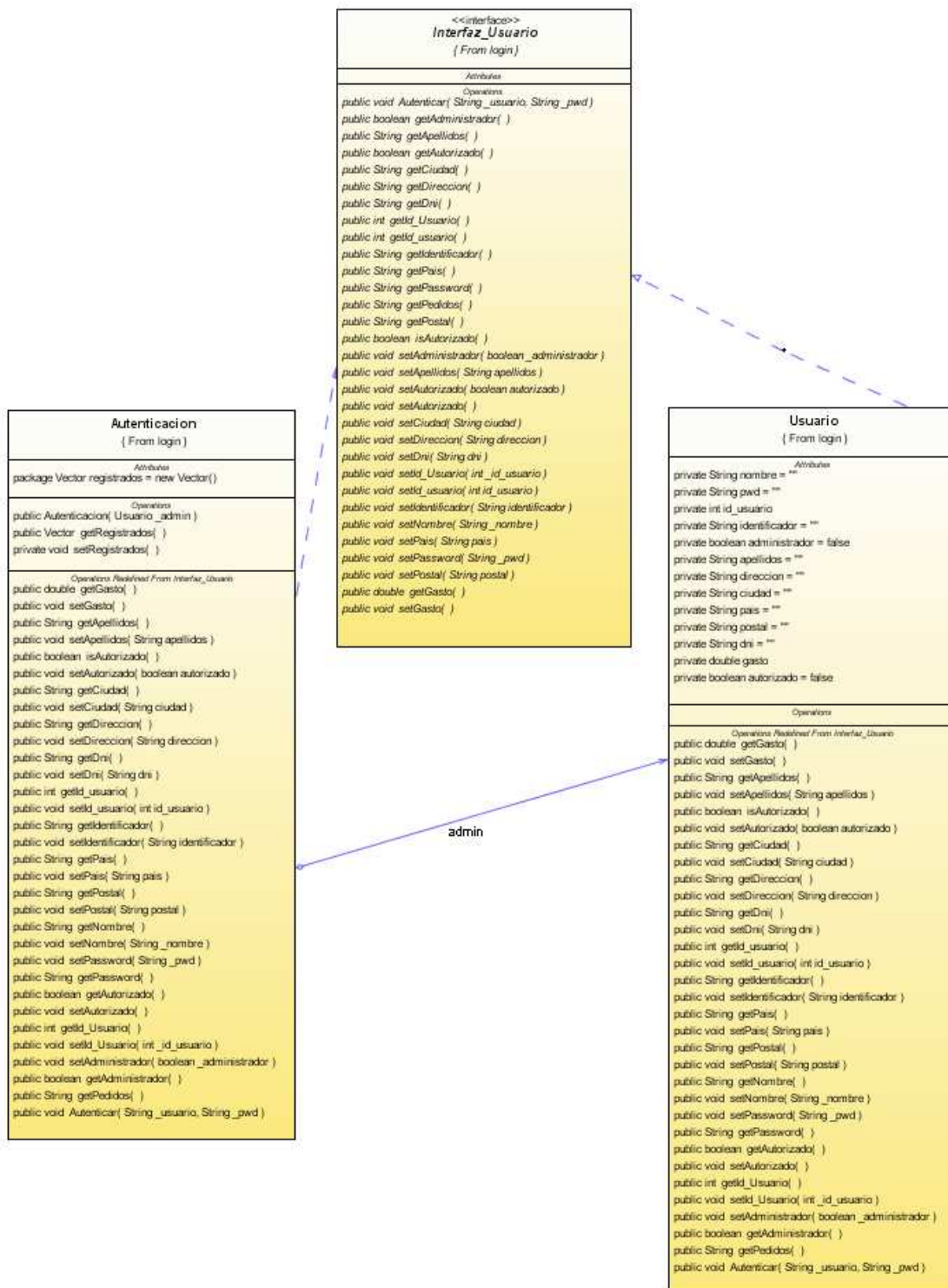
```



## 2.3. ESTRUCTURALES

### 2.3.1. DECORATOR

Objetivo: Añadir nuevas responsabilidades dinámicamente a un objeto sin modificar su apariencia externa o su función, es una alternativa a crear demasiadas subclases por herencia. Se utiliza este patrón cuando se desea añadir funcionalidad a una clase sin las restricciones que implica la utilización de la herencia, o cuando se desea añadir dicha funcionalidad de forma dinámica en tiempo de ejecución, de manera transparente a los usuarios.



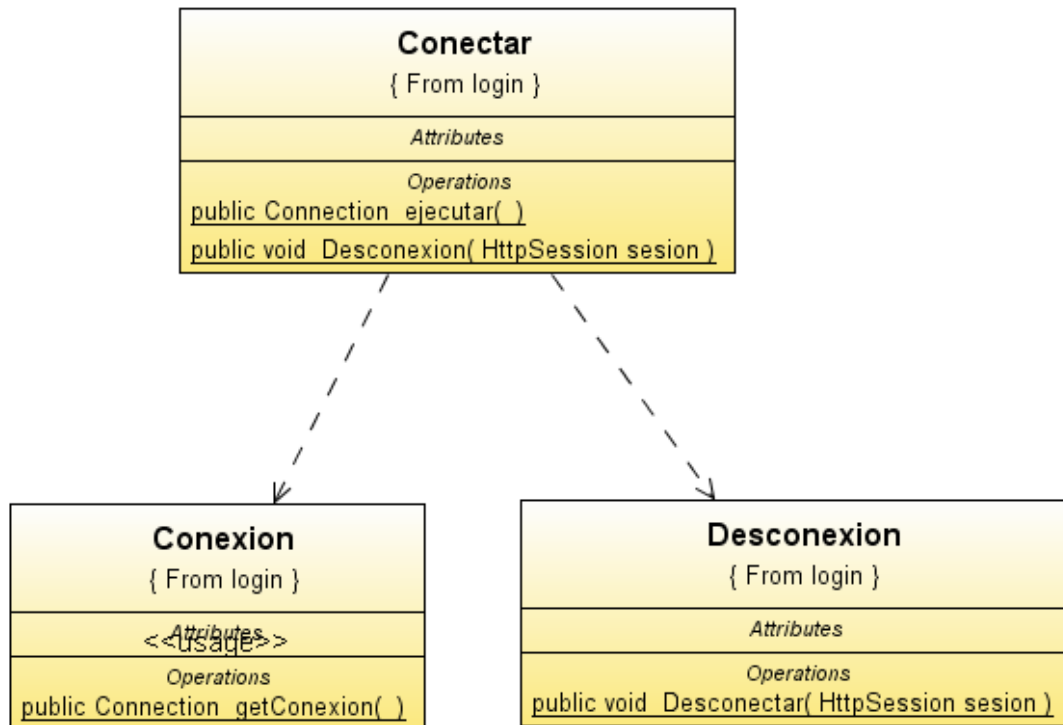
Implementacion: En el sistema existen dos tipos de usuarios: un usuario ordinario, el cual tiene la funcionalidad propia del rol de cliente; y un usuario administrador, al cual se le proporcionan una serie de funciones específicas, como introducir productos o mostrar una lista de usuarios conectados al sistema.

La clase Autenticacion corresponde a este rol de administrador. Dicha clase contiene un atributo de tipo Usuario que contiene los datos del administrador. Además implementa una serie de procedimientos que le otorgan funcionalidad añadida. Se define un nuevo atributo de tipo Vector, que guardará los usuarios registrados en la página web, y el método correspondiente para la creación de dicho vector.

A modo de establecer una comparación entre el diagrama mostrado y los que se encuentran en la documentación de la asignatura, la clase Interfaz\_Usuario actúa como Componente, la clase Usuario actúa como ComponenteConcreto, y la clase Autenticacion es el DecoradorConcreto.

### 2.3.2. FACADE

Objetivo: simplificar el acceso a un conjunto de subsistemas o un sistema complejo. Representa una única interfaz unificada, que envuelve el subsistema, y es responsable de colaborar con los componentes del subsistema. Este patrón ofrece un punto de acceso al resto de las clases, facilitando una interfaz sencilla para el acceso a subsistemas.



Implementación: la clase Conectar actúa como fachada, simplificando el acceso a las clases Conexión y Desconexion, que proporcionan la funcionalidad necesaria para establecer una conexión con la base de datos, y desconectarse del sistema (eliminado la sesión HttpSession del usuario).

Si fuese necesario realizar un cambio en alguna de las operaciones, simplemente se necesitaría cambiar el comportamiento de las clases Conexión o Desconexion. La interfaz accedida por los clientes permanecería inmutable.

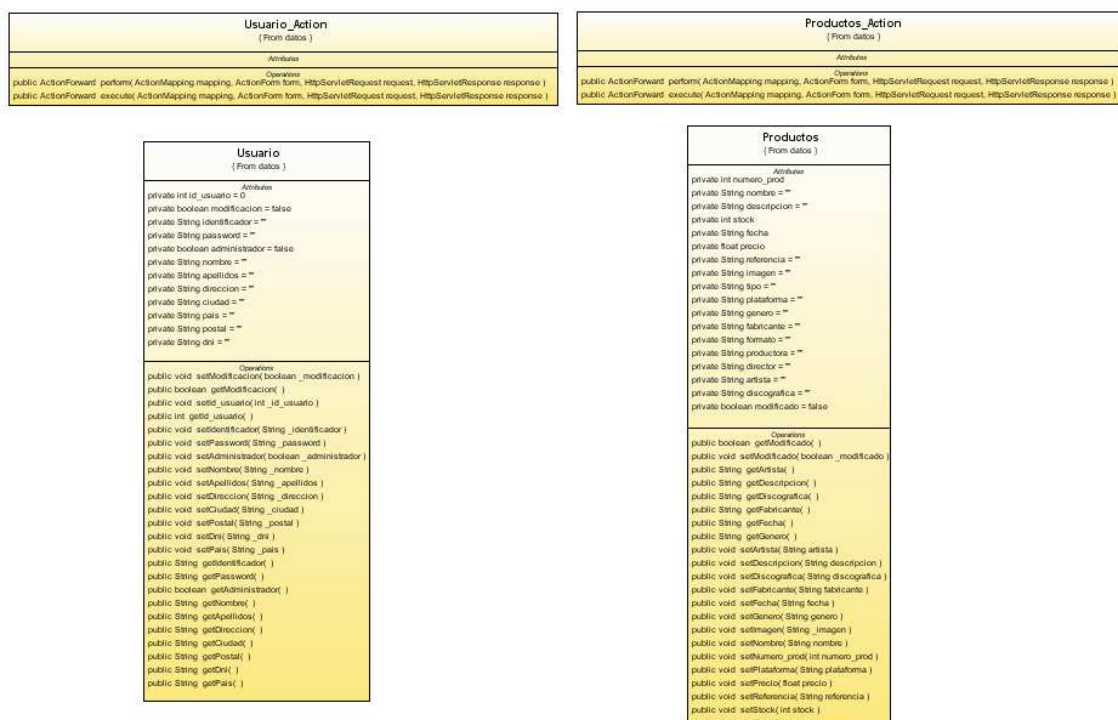
### 2.3.3. MODELO-VISTA-CONTROLADOR

Objetivo: el patrón MVC pretende la división de una solución en tres partes bien diferenciadas:

- Modelo: contiene el control de la funcionalidad de la aplicación. El modelo encapsula el estado de la aplicación sin saber nada de las categorías Vista y Controlador.
- Vista: proporciona la presentación del Modelo, representando el look o apariencia de la aplicación. La vista puede acceder a los métodos get() del Modelo para obtener información, pero no tiene acceso a los métodos set() para proporcionar información al Modelo, sino que las actualizaciones en el modelo deben realizarse a través del Controlador.
- Controlador: es quien reacciona a las acciones del usuario y el encargado de crear y asignar valores al Modelo para su funcionamiento.

Utilizar este patrón permite adoptar soluciones altamente escalables, y proporciona un mantenimiento mucho más sencillo que los modelos basados en una sola capa.

Implementación: para la implementación de este patrón se ha dividido la aplicación en los citados tres niveles. El nivel del modelo estará compuesto por los beans y clases que representan los diferentes elementos de la aplicación de compra virtual; la vista estará representada por las páginas JSP que componen la aplicación; y por último, el controlador está constituido por los servlets que controlarán el flujo de la aplicación, y la introducción y modificación de datos en la base de datos.



En esta imagen se observan las clases Usuario y Productos que extienden de ActionForm y representan los formularios enviados a los Servlet para modificar o añadir usuarios y productos respectivamente. Las clases Usuario\_Action y Productos\_Action son los servlet mencionados anteriormente.

Cabe destacar la clase `ClienteServlet`, la cual realiza un control del flujo de la aplicación, redirigiendo las peticiones del usuario en función de los parámetros que recibe en las sucesivas peticiones http.

| <b>ClienteServlet</b>   |
|---|
| <i>Attributes</i>   |
| <i>Operations</i>   |
| <pre>protected void processRequest( HttpServletRequest request, HttpServletResponse response ) protected void doGet( HttpServletRequest request, HttpServletResponse response ) protected void doPost( HttpServletRequest request, HttpServletResponse response ) public String getServletInfo( )</pre> |

Un ejemplo de cómo redirige el tráfico se puede observar en el siguiente fragmento de su código:

```
if(pagina==null || pagina.contains(tienda.CarroBean.COMPRAR))
{
    if(cesta.isVacia())

reqDisp=getServletConfig().getServletContext().getRequestDispatcher("/CestaCompra.jsp");
        else

reqDisp=getServletConfig().getServletContext().getRequestDispatcher("/RealizarPedido.jsp");
    }else
    {
        if(pagina.equals(tienda.CarroBean.SELECCIONAR))

        { if(log.getAutorizado())
            {

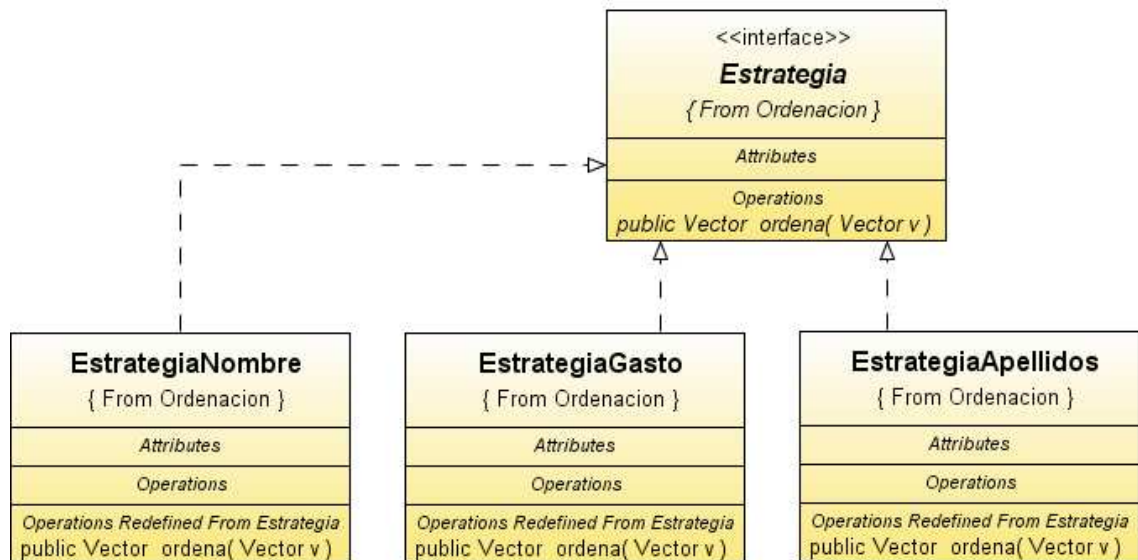
reqDisp=getServletConfig().getServletContext().getRequestDispatcher("/welcomeStruts.jsp");
                }else
                {

reqDisp=getServletConfig().getServletContext().getRequestDispatcher("/Registrate.jsp");
                }
            }
        }
    }
```

## 2.4. PATRONES DE DISEÑO FUNDAMENTALES

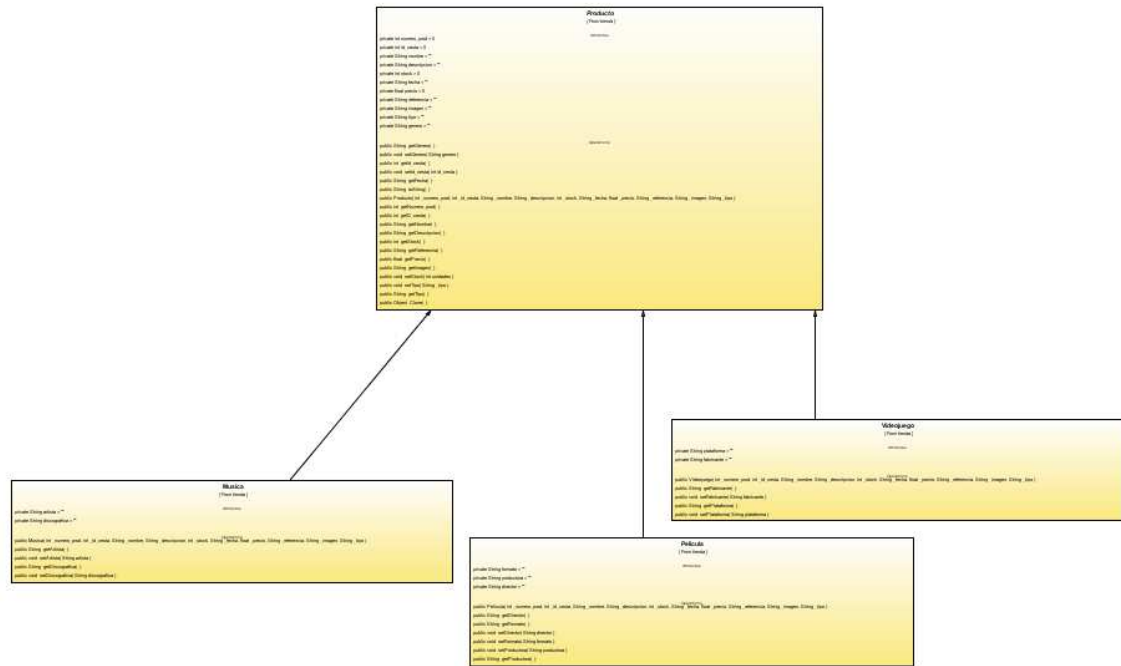
El patrón Interface mantiene una clase que usa datos y servicios provistos por otras clases independientes, para proveer un acceso uniforme. Usando indirección, esta clase interfaz provee a sus clases herederas acceso uniforme a métodos específicos, sin que deben saber a clase concreta pertenecen.

Este patrón es usado por ejemplo, en el patrón Strategy.



El patrón Abstract Superclass garantiza la coherencia de comportamiento de las clases conceptualmente relacionadas dándoles una clase padre abstracta. Organiza el comportamiento común de las clases en una superclase abstracta.

Este patrón es usado por ejemplo, en el patrón Factory Method, en el cual producto está representado por una clase abstracta, la cual es redefinida por las clases Música, Película y Videojuego.



El patrón inmutable se utiliza cuando múltiples objetos comparten el acceso a un mismo objeto y es necesaria una coordinación para evitar problemas. Para ello ningún método, a excepción del constructor, debe modificar los valores de las variables de la instancia de la clase.

Este objetivo se consigue mediante el uso del patrón Singleton en la clase que representa el catálogo de la tienda virtual, en el cual, una vez es creada una instancia no será modificada. Los cambios se verán reflejados cuando se genere una nueva instancia a partir de los datos de la base de datos.

### 3. DIAGRAMAS DE CLASES

#### 3.1. PAQUETE TIENDA

##### CLASE PRODUCTO

| <b>Producto</b><br>{ From tienda }   |
|--|
| <i>Atributos</i>   |
| <pre>private int numero_prod = 0 private int id_cesta = 0 private String nombre = "" private String descripcion = "" private int stock = 0 private String fecha = "" private float precio = 0 private String referencia = "" private String imagen = "" private String tipo = "" private String genero = ""</pre>  |
| <i>Operations</i>  |
| <pre>public String getGenero( ) public void setGenero( String genero ) public int getId_cesta( ) public void setId_cesta( int id_cesta ) public String getFecha( ) public String toString( ) public Producto( int _numero_prod, int _id_cesta, String _nombre, String _descripcion, int _stock, String _fecha, float _precio, String _referencia, String _imagen, String _tipo ) public int getNumero_prod( ) public int getID_cesta( ) public String getNombre( ) public String getDescripcion( ) public int getStock( ) public String getReferencia( ) public float getPrecio( ) public String getImagen( ) public void setStock( int unidades ) public void setTipo( String _tipo ) public String getTipo( ) public Object Clone( )</pre> |

Esta clase implementa cada uno de los productos genéricos que se van a introducir en el catálogo y en una cesta de la compra de un determinado cliente.

Cuenta con los atributos correspondientes a los campos de la tabla Productos de la base de datos. Para cada uno de estos atributos dispone de los métodos Set y Get correspondientes.

Esta clase implementa la interfaz cloneable, lo que permite que cuando se vaya a copiar un determinado producto desde el catálogo, a una determinada cesta, se realice mediante clonación. Esto permite eliminar la sobrecarga que incluye instanciar un nuevo objeto de la clase estableciendo el valor de todos sus atributos. Dicho objetivo se consigue a través del método Clone definido en el objeto.



## CLASE PELICULA

| <b>Pelicula</b><br>{ From tienda }  |
|---|
| <i>Atributos</i>  |
| <pre>private String formato = "" private String productora = "" private String director = ""</pre>  |
| <i>Operations</i>   |
| <pre>public Pelicula( int _numero_prod, int _id_cesta, String _nombre, String _descripcion, int _stock, String _fecha, float _precio, String _referencia, String _imagen, String _tipo ) public String getDirector( ) public String getFormato( ) public void setDirector( String director ) public void setFormato( String formato ) public void setProductora( String productora ) public String getProductora( )</pre> |

La clase película extiende de la clase producto, lo que permite a ésta disponer de los mismos atributos y métodos que la clase producto y, además, implementar una serie de atributos y métodos propios que la especializarán, como son los atributos formato, productora y director, y sus correspondientes métodos Set y Get.

## CLASE VIDEOJUEGO

| <b>Videojuego</b><br>{ From tienda }   |
|--|
| <i>Atributos</i>   |
| <pre>private String plataforma = "" private String fabricante = ""</pre>   |
| <i>Operations</i>  |
| <pre>public Videojuego( int _numero_prod, int _id_cesta, String _nombre, String _descripcion, int _stock, String _fecha, float _precio, String _referencia, String _imagen, String _tipo ) public String getFabricante( ) public void setFabricante( String fabricante ) public String getPlataforma( ) public void setPlataforma( String plataforma )</pre> |

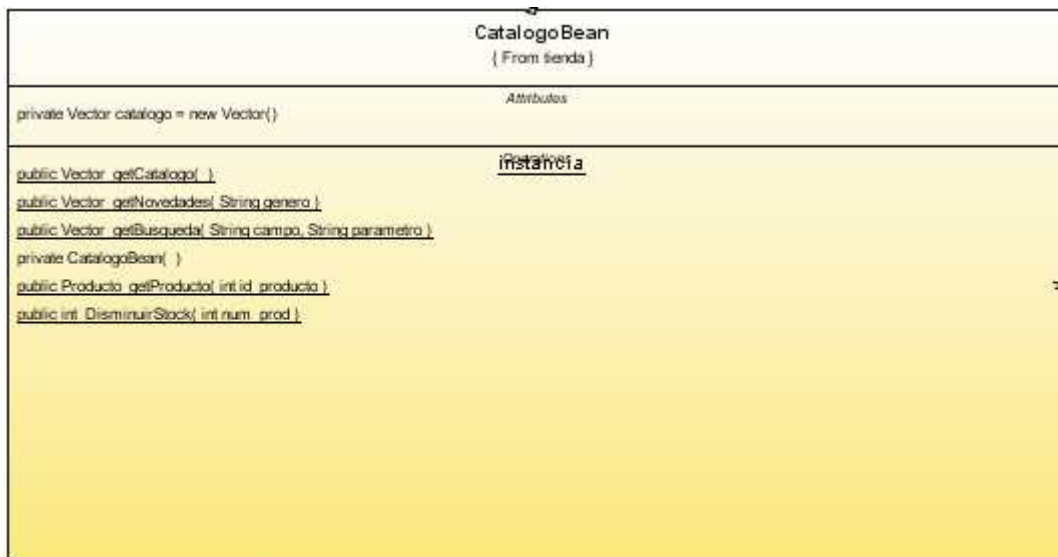
Al igual que la clase Película, la clase Videojuego extiende de la clase producto, implementando atributos y métodos propios, como son plataforma y fabricante, y sus correspondientes métodos Set y Get.

## CLASE MÚSICA

| <b>Musica</b><br>{ From tienda }  |
|---|
| <i>Atributos</i>  |
| <pre>private String artista = "" private String discografica = ""</pre>   |
| <i>Operations</i>   |
| <pre>public Musica( int _numero_prod, int _id_cesta, String _nombre, String _descripcion, int _stock, String _fecha, float _precio, String _referencia, String _imagen, String _tipo ) public String getArtista( ) public void setArtista( String artista ) public String getDiscografica( ) public void setDiscografica( String discografica )</pre> |

Del mismo modo que las dos clases anteriores, la clase Música extiende de la clase producto, añadiendo los atributos artista y discográfica, así como sus métodos Set y Get.

## CLASE CATALOGOBEAN



La clase CatalogoBean implementa la funcionalidad de un catálogo de productos. Para almacenar dichos productos se utiliza una estructura Vector.

Para que no resulte necesario instanciar un nuevo Vector cada vez que accedamos, se ha utilizado el patrón de diseño Singleton. Para ello se almacena una instancia del objeto dentro de la clase, que será la que se devolverá a las llamadas del cliente a través de la función `getCatalogo()`; en caso de no haberse instanciado ningún objeto se crea, extrayendo todos los productos de la base de datos.

Además de esta función, se han implementado otras funcionalidades:

- Obtener el catálogo ordenado por su campo fecha, de uno de los tres tipos disponibles de producto. `getNovedades()`.
- Obtener productos cuyo valor en un determinado campo coincida con uno especificado. `getBusqueda()`.
- Obtener un producto concreto, a través de su identificador de producto. `getProducto()`.
- Disminuir el stock de un producto especificado por su identificador de producto. `DisminuirStock()`.

## CLASE CARROBEAN



La clase CarroBean representa el objeto al que se irán añadiendo los productos seleccionados por el cliente para su compra.

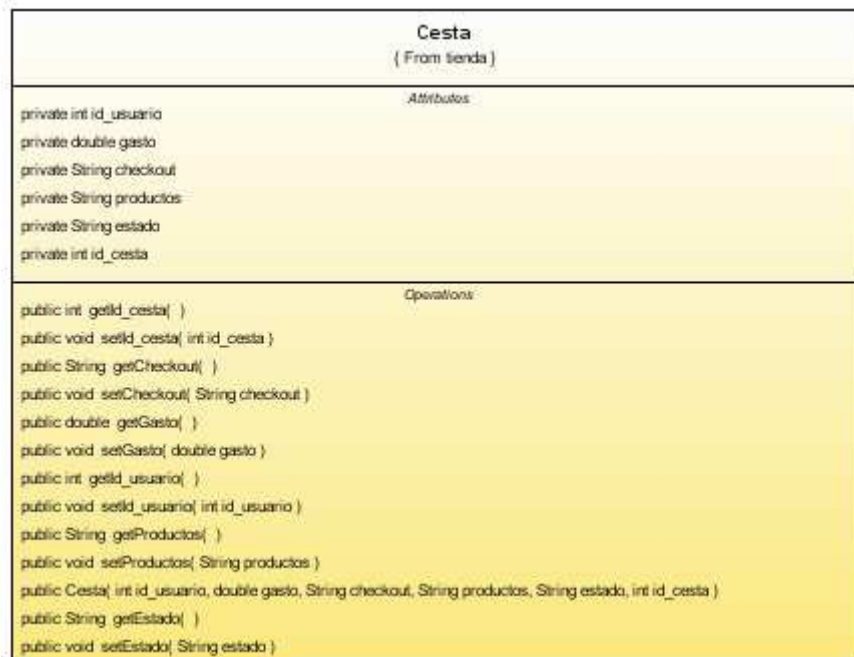
Para almacenar dichos productos se utiliza una tablaHash, en la cual se utilizará como índice el número de producto que se introduce, lo que permitirá realizar más rápidamente determinadas operaciones con la cesta, como eliminar un producto de la misma.

Se han definido una serie de constantes de tipo String que serán utilizadas por el Servlet ClienteServlet para controlar el flujo de páginas que irá visitando el cliente en función de sus acciones.

Las funcionalidades proporcionadas por esta clase son:

- Obtener el total de gasto de la cesta. Mediante el método getTotal().
- Obtener una cadena con los productos incluidos en la cesta. Mediante el método getProductos().
- Insertar los productos seleccionados por el usuario en la cesta, especificados en la petición de la página Web (esto se realiza de esta manera para una futura ampliación de la web que permita introducir más de un producto simultáneamente). Mediante el método guardarCompra().
- Finalizar la compra, introduciendo la cesta cerrada a la base de datos. Esta acción tendrá como consecuencia que no se puedan introducir más elementos en la cesta. Se guardará la cesta con el identificador de usuarios indicado como parámetro. Mediante el método finalizarCompra().
- Borrar el contenido de la cesta. Mediante el método Clear().
- Eliminar un determinado elemento de la cesta. Mediante el método Eliminar().
- Comprobar si la cesta está vacía. Esta funcionalidad es necesaria para imposibilitar cerrar una cesta que no contiene ningún producto. Mediante el método isVacia().

## CLASE CESTA



La clase cesta representa las cestas de la compra que se encuentran almacenadas en la base de datos. Cuando un usuario con rol de administrador obtiene el catálogo de cestas del sistema, ya sea para modificarlas o para observarlas simplemente, cada una de estas cestas es representada por una instancia de esta clase.

Cuenta con un atributo por cada campo de la tabla cesta de la base de datos, así como los métodos set y get para cada uno de ellos.

## CLASE ITERADOR

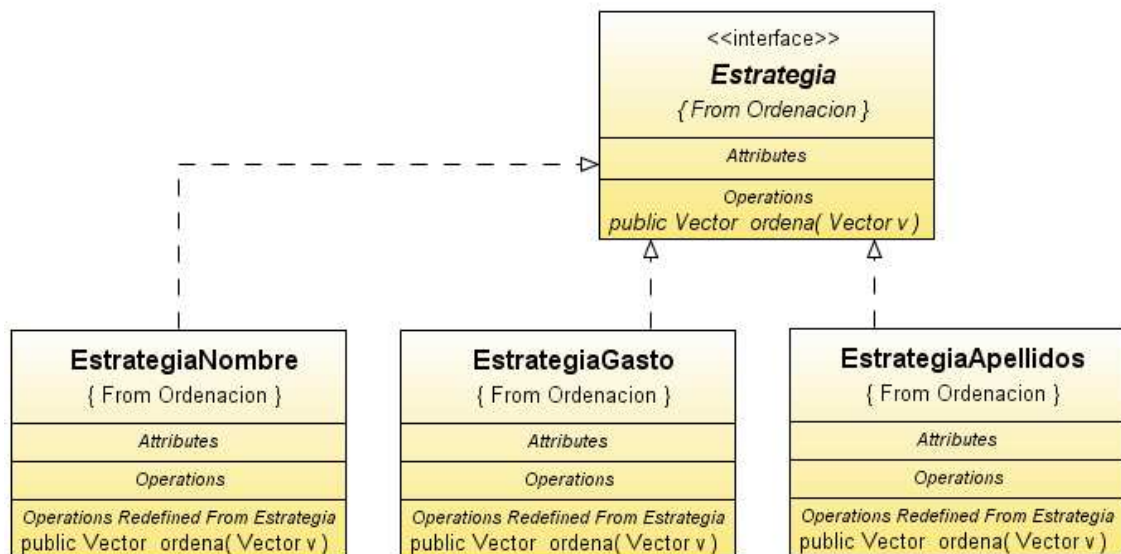
Se ha implementado el patrón de diseño iterador para acceder a cada una de las cestas almacenadas en la base de datos. El diseño de este patrón se encuentra en el apéndice de la memoria, titulada patrones de diseño.

### 3.2. PAQUETE ORDENACIÓN

Las clases contenidas en este paquete tienen como finalidad proporcionar una ordenación de los diferentes usuarios registrados en el sistema, a modo de facilitar la visualización de los datos estadísticos de la tienda.

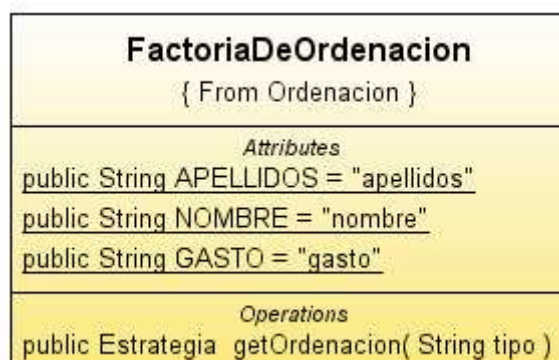
Las clases implementadas en este paquete son las resultante de la aplicación del patrón de diseño Strategy.

#### CLASES ESTRATEGIANOMBRE, ESTRATEGIAGASTO, ESTRATEGIAAPELLIDOS

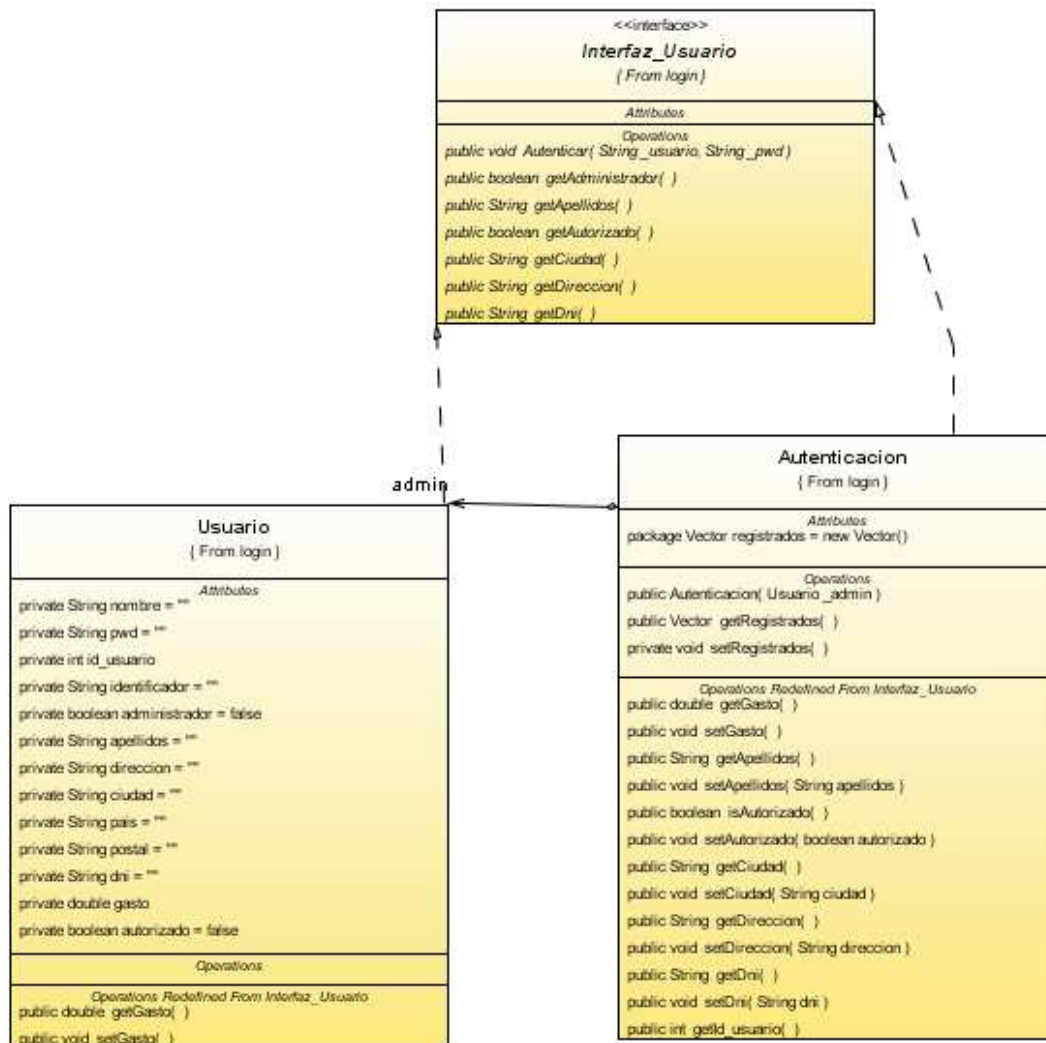


Estas tres clases implementan la interfaz Estrategia, lo que permite redefinir la operación Ordena en función del objeto instanciado. El método ordena de la clase EstrategiaNombre ordena el Vector v de usuarios, en función de su atributo nombre; el método ordena de la clase EstrategiaGasto los ordena en función de su campo gasto; y por último, el método ordena de la clase EstrategiaApellidos los ordena en función de su campo Apellidos.

La selección de una clase u otra se realiza a través de la clase FactoriadeOrdenacion, que tomará una cadena indicando el campo por el que ordenar los usuarios, y devolverá una instancia de una de las clases anteriores.



### 3.3. PAQUETE LOGIN



El paquete login implementa los dos roles que se pueden tomar cuando se utiliza la aplicación Web, el de usuario estándar y el de administrador del sistema.

La opción seleccionada consiste en la utilización de un patrón Decorator, mediante el cual si una determinada instancia de la clase Usuario tiene derechos de administrador, se instanciará un objeto de la clase Autenticacion, que tomará este objeto Usuario como parámetro, y ampliará la funcionalidad de este.

## CLASE USUARIO

| Usuario<br>( From login )  |
|--|
| <i>Attributes</i><br>private String nombre = ""<br>private String pwd = ""<br>private int id_usuario<br>private String identificador = ""<br>private boolean administrador = false<br>private String apellidos = ""<br>private String direccion = ""<br>private String ciudad = ""<br>private String pais = ""<br>private String postal = ""<br>private String dni = ""<br>private double gasto<br>private boolean autorizado = false  |
| <i>Operations</i><br><i>Operations Redefined From Interface, Usuario:</i><br>public double getGasto( )<br>public void setGasto( )<br>public String getApellidos( )<br>public void setApellidos( String apellidos )<br>public boolean isAutorizado( )<br>public void setAutorizado( boolean autorizado )<br>public String getCiudad( )<br>public void setCiudad( String ciudad )<br>public String getDireccion( )<br>public void setDireccion( String direccion )<br>public String getDni( )<br>public void setDni( String dni )<br>public int getId_usuario( )<br>public void setId_usuario( int id_usuario )<br>public String getIdentificador( )<br>public void setIdentificador( String identificador )<br>public String getPais( )<br>public void setPais( String pais )<br>public String getPostal( )<br>public void setPostal( String postal )<br>public String getNombre( )<br>public void setNombre( String nombre )<br>public void setPassword( String pwd )<br>public String getPassword( )<br>public boolean getAutorizado( )<br>public void setAutorizado( )<br>public int getId_Usuario( )<br>public void setId_Usuario( int id_usuario )<br>public void setAdministrador( boolean administrador )<br>public boolean getAdministrador( )<br>public String getPedidos( )<br>public void Autenticar( String_usuario, String_pwd ) |

Esta clase cuenta con los atributos que contiene la tabla Usuario de la base de datos. Además dispone de los métodos Set y Get para cada uno de estos atributos.

El método Autenticar() toma como parámetros un nombre de usuario y una contraseña. Tras comprobar que se encuentra un usuario en la base de datos, cuyo identificador y contraseña concuerda con los valores recibidos, realiza una consulta SQL, para recuperar el valor de los diferentes atributos del objeto.

El método setGasto() realiza una consulta SQL para obtener el total de gasto de las cestas de la compra las cuales ha cerrado el usuario.

El método getPedidos() devuelve una cadena que muestra las cestas cerradas por el usuario, así como los productos que incluyen y el gasto total de la misma.



## CLASE AUTENTICACIÓN

| Autenticación<br>( From login )   |
|---|
| <i>Attributes</i><br>package Vector registrados = new Vector();   |
| <i>Operations</i><br>public Autenticación( Usuario_admin )<br>public Vector getRegistrados( )<br>private void setRegistrados( )   |
| <i>Operations Inherited From Interfaz_Usuario</i><br>public double getGasto( )<br>public void setGasto( )<br>public String getApellidos( )<br>public void setApellidos( String apellidos )<br>public boolean isAutorizado( )<br>public void setAutorizado( boolean autorizada )<br>public String getCiudad( )<br>public void setCiudad( String ciudad )<br>public String getDireccion( )<br>public void setDireccion( String direccion )<br>public String getDni( )<br>public void setDni( String dni )<br>public int getid_usuario( )<br>public void setid_usuario( int id_usuario )<br>public String getIdentificador( )<br>public void setIdentificador( String identificador )<br>public String getPais( )<br>public void setPais( String pais )<br>public String getPostal( )<br>public void setPostal( String postal )<br>public String getNombre( )<br>public void setNombre( String _nombre )<br>public void setPassword( String _pwd )<br>public String getPassword( )<br>public boolean getAutorizado( )<br>public void setAutorizado( )<br>public int getid_Usuario( )<br>public void setid_Usuario( int id_usuario )<br>public void setAdministrador( boolean _administrador )<br>public boolean getAdministrador( )<br>public String getPedidos( )<br>public void Autenticar( String _usuario, String _pwd ) |

La clase autenticación contiene un atributo de tipo Usuario, que incluye los datos del Administrador. Además tiene como atributo un Vector en el que almacenará los distintos usuarios registrados en el sistema; así como sus métodos Set y Get. Este Vector es utilizado para aplicar la funcionalidad correspondiente a la visualización de los usuarios registrados en el sistema, la cual solamente puede ser accedida por un administrador del sistema.



### 3.4. PAQUETE DATOS

Las clases definidas en este paquete corresponden a los formularios utilizados para la introducción de un nuevo usuario o nuevo producto, o la modificación de un usuario o producto.

#### CLASE USUARIO

| Usuario<br>( From datos )  |
|--|
| <i>Attributes</i><br>private int id_usuario = 0<br>private boolean modificacion = false<br>private String identificador = ""<br>private String password = ""<br>private boolean administrador = false<br>private String nombre = ""<br>private String apellidos = ""<br>private String direccion = ""<br>private String ciudad = ""<br>private String pais = ""<br>private String postal = ""<br>private String dni = ""   |
| <i>Operations</i><br>public void setModificacion( boolean _modificacion )<br>public boolean getModificacion( )<br>public void setId_usuario( int _id_usuario )<br>public int getId_usuario( )<br>public void setIdentificador( String _identificador )<br>public void setPassword( String _password )<br>public void setAdministrador( boolean _administrador )<br>public void setNombre( String _nombre )<br>public void setApellidos( String _apellidos )<br>public void setDireccion( String _direccion )<br>public void setCiudad( String _ciudad )<br>public void setPostal( String _postal )<br>public void setDni( String _dni )<br>public void setPais( String _pais )<br>public String getIdentificador( )<br>public String getPassword( )<br>public boolean getAdministrador( )<br>public String getNombre( )<br>public String getApellidos( )<br>public String getDireccion( )<br>public String getCiudad( )<br>public String getPostal( )<br>public String getDni( )<br>public String getPais( ) |

La clase usuario extiende de la clase ActionForm. Corresponde al formulario enviado para la modificación o el añadido de un nuevo usuario al sistema.

## CLASE USUARIO\_ACTION

|   |
|---|
| <b>Usuario_Action</b><br>{ From datos }   |
| <i>Attributes</i>   |
| <i>Operations</i><br>public ActionForward perform( ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response )<br>public ActionForward execute( ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response ) |

La clase Usuario\_Action extiende de la clase Action. Esta clase forma parte del flujo de la aplicación, recibirá el formulario instanciado por la case Usuario y a partir de su método execute introducirá o modificara un usuario en la base de datos del sistema.

## CLASE PRODUCTOS

|   |
|---|
| <b>Productos</b><br>{ From datos }  |
| <i>Attributes</i><br>private int numero_prod<br>private String nombre = ""<br>private String descripcion = ""<br>private int stock<br>private String fecha<br>private float precio<br>private String referencia = ""<br>private String imagen = ""<br>private String tipo = ""<br>private String plataforma = ""<br>private String genero = ""<br>private String fabricante = ""<br>private String formato = ""<br>private String productora = ""<br>private String director = ""<br>private String artista = ""<br>private String discografica = ""<br>private boolean modificado = false  |
| <i>Operations</i><br>public boolean getModificado( )<br>public void setModificado( boolean _modificado )<br>public String getArtista( )<br>public String getDescripcion( )<br>public String getDiscografica( )<br>public String getFabricante( )<br>public String getFecha( )<br>public String getGenero( )<br>public void setArtista( String artista )<br>public void setDescripcion( String descripcion )<br>public void setDiscografica( String discografica )<br>public void setFabricante( String fabricante )<br>public void setFecha( String fecha )<br>public void setGenero( String genero )<br>public void setImagen( String _imagen )<br>public void setNombre( String nombre )<br>public void setNumero_prod( int numero_prod )<br>public void setPlataforma( String plataforma )<br>public void setPrecio( float precio )<br>public void setReferencia( String referencia )<br>public void setStock( int stock )<br>public void setTipo( String tipo ) |

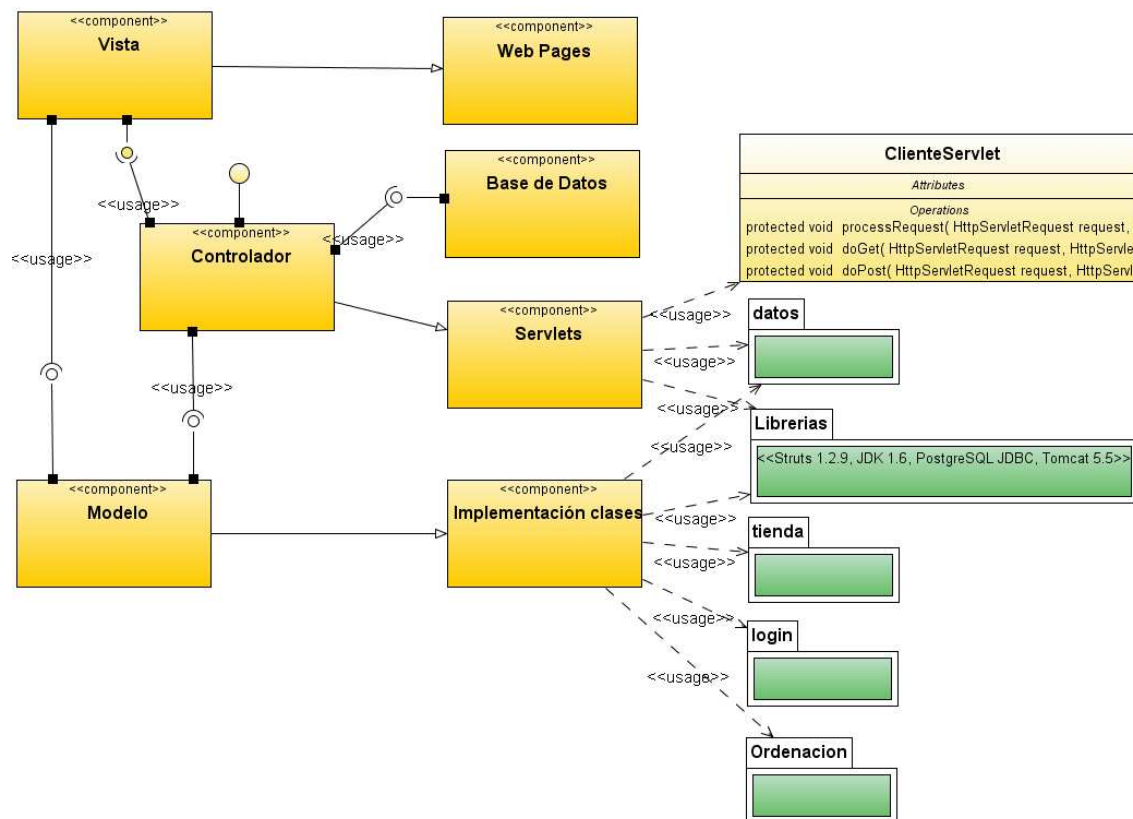
La clase productos extiende de la clase ActionForm. Corresponde al formulario enviado para la modificación o el añadido de un nuevo producto al sistema.

## CLASE PRODUCTOS\_ACTION

|  |
|--|
| <b>Productos_Action</b><br>{ From datos }  |
| Attributes   |
| Operations<br>public ActionForward perform( ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response )<br>public ActionForward execute( ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response ) |

La clase Productos\_Action extiende de la clase Action. Esta clase forma parte del flujo de la aplicación, recibirá el formulario instanciado por la case Productos y a partir de su método execute introducirá o modificara un producto en la base de datos del sistema.

## 4. DIAGRAMA DE COMPONENTES



El modelo desarrollado está basado en el patrón de diseño estructural MVC. Por ello dispondrá de tres componentes:

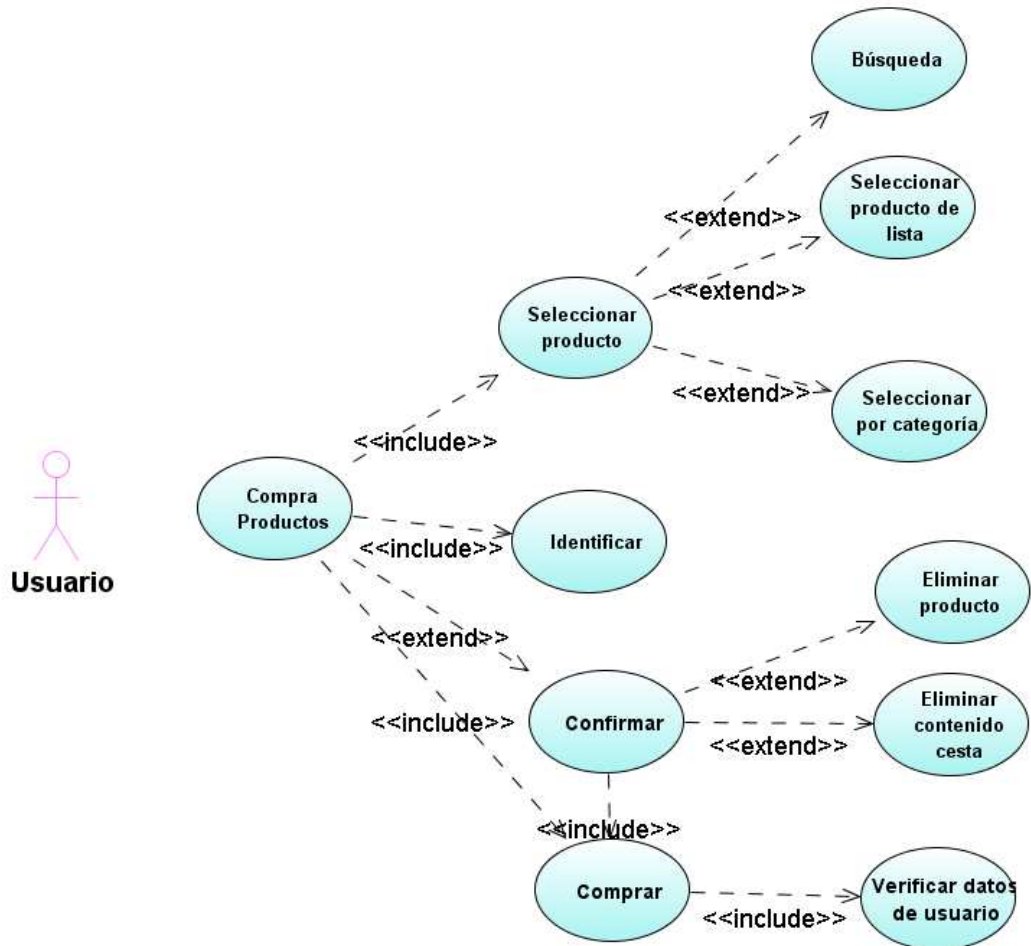
- **Modelo:** contiene el control de la funcionalidad de la aplicación. El modelo encapsula el estado de la aplicación sin saber nada de las categorías Vista y Controlador.
- **Vista:** proporciona la presentación del Modelo, representando el look o apariencia de la aplicación. La vista puede acceder a los métodos `get()` del Modelo para obtener información, pero no tiene acceso a los métodos `set()` para proporcionar información al Modelo, sino que las actualizaciones en el modelo deben realizarse a través del Controlador.
- **Controlador:** es quien reacciona a las acciones del usuario y el encargado de crear y asignar valores al Modelo para su funcionamiento.

Utilizar este patrón permite adoptar soluciones altamente escalables, y proporciona un mantenimiento mucho más sencillo que los modelos basados en una sola capa.

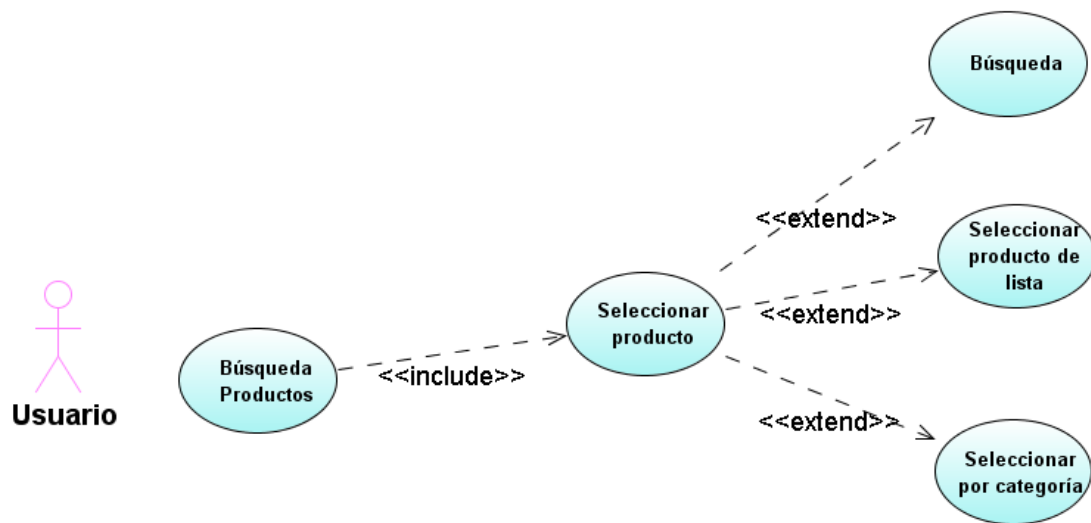
**Implementación:** para la implementación de este patrón se ha dividido la aplicación en los citados tres niveles. El nivel del modelo estará compuesto por los beans y clases que representan los diferentes elementos de la aplicación de compra virtual; la vista estará representada por las páginas JSP que componen la aplicación; y por último, el controlador está constituido por los servlets que controlarán el flujo de la aplicación, y la introducción y modificación de datos en la base de datos.

## 5. DIAGRAMAS DE CASOS DE USO

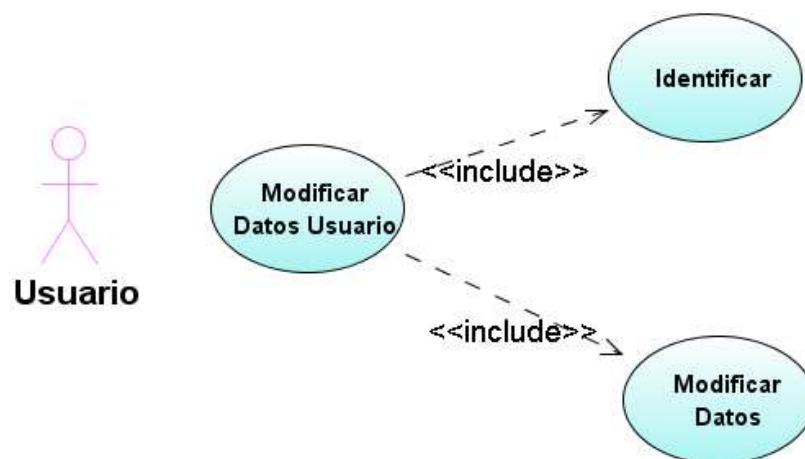
### 5.1. CASO DE USO COMPRA



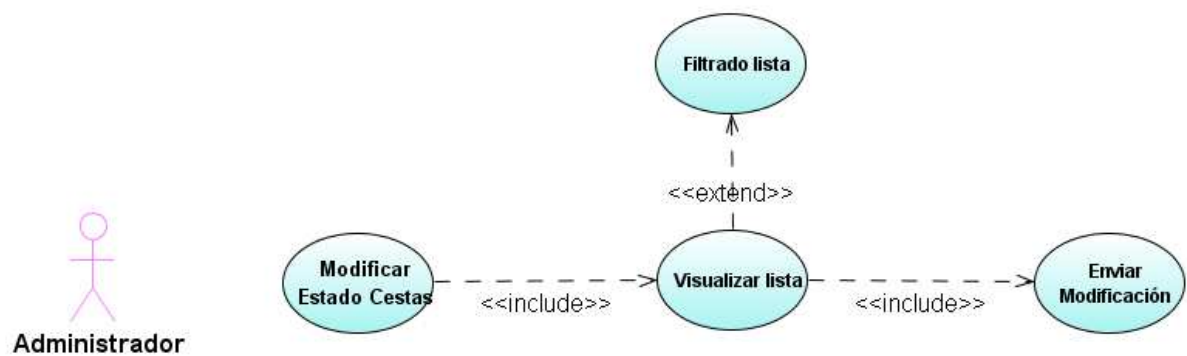
## 5.2. CASO DE USO VISUALIZACIÓN DE PRODUCTOS



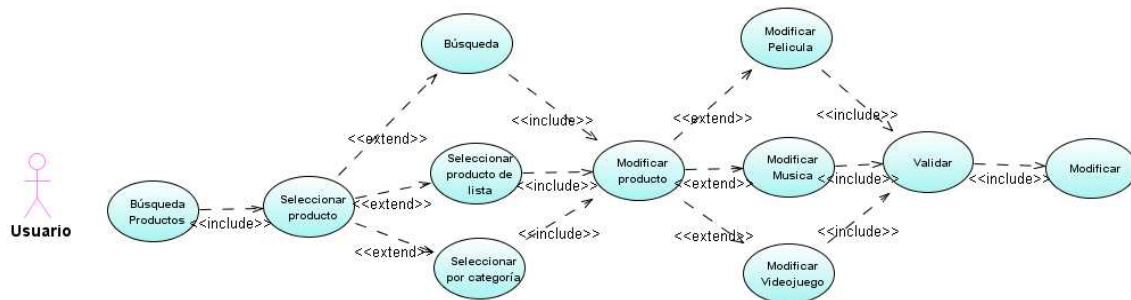
## 5.3. CASO DE USO MODIFICACIÓN DE DATOS



## 5.4. CASO DE USO MODIFICACIÓN DE ESTADO DE PEDIDOS



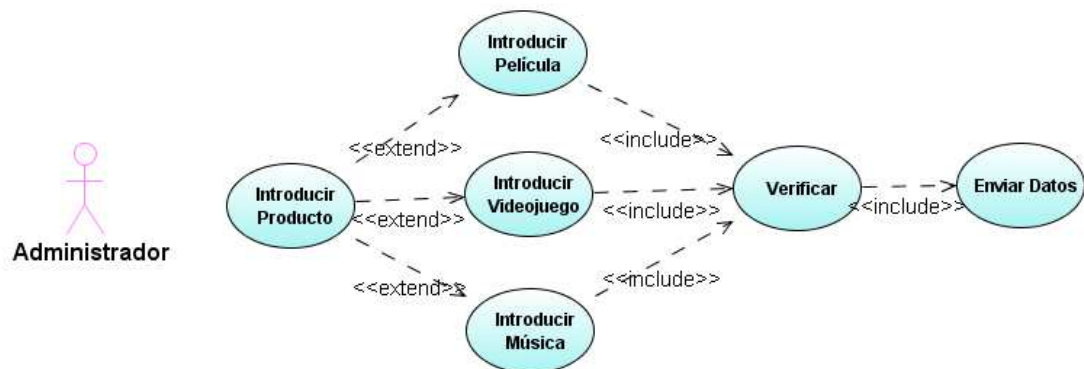
## 5.5. CASO DE USO MODIFICACIÓN DE PRODUCTO



## 5.6. CASO DE USO VISUALIZACIÓN DE CESTAS

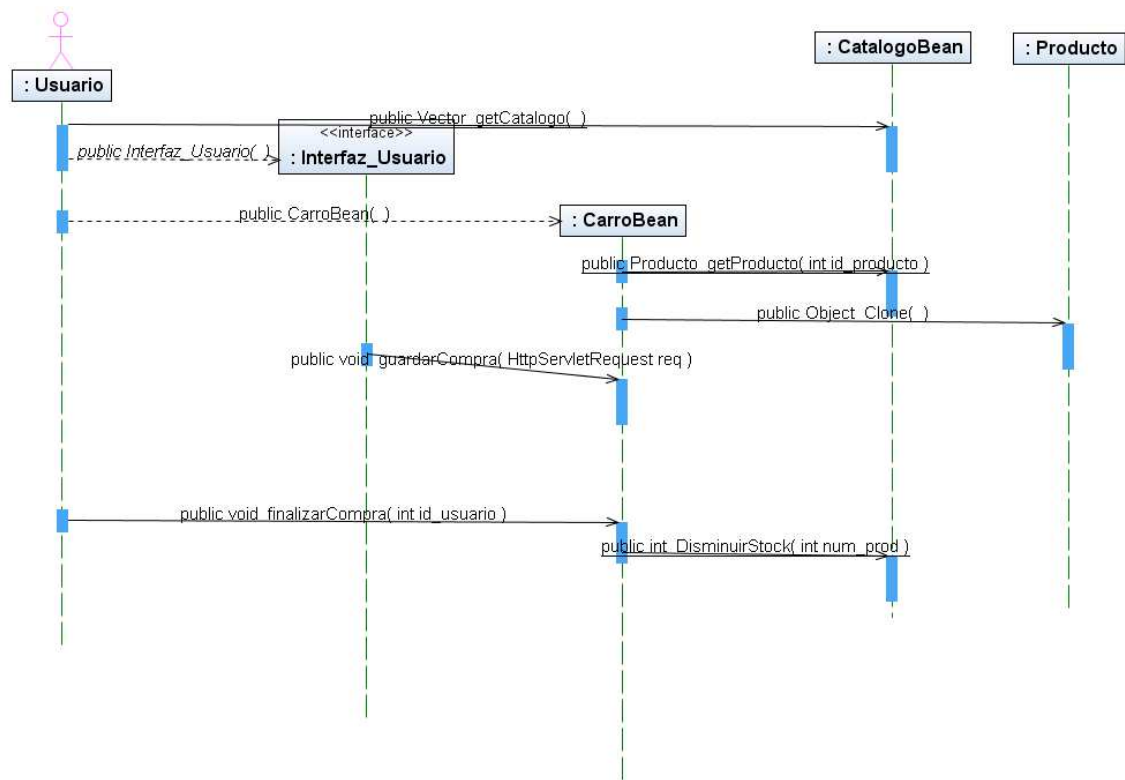


## 5.7. CASO DE USO INTRODUCIR PRODUCTO

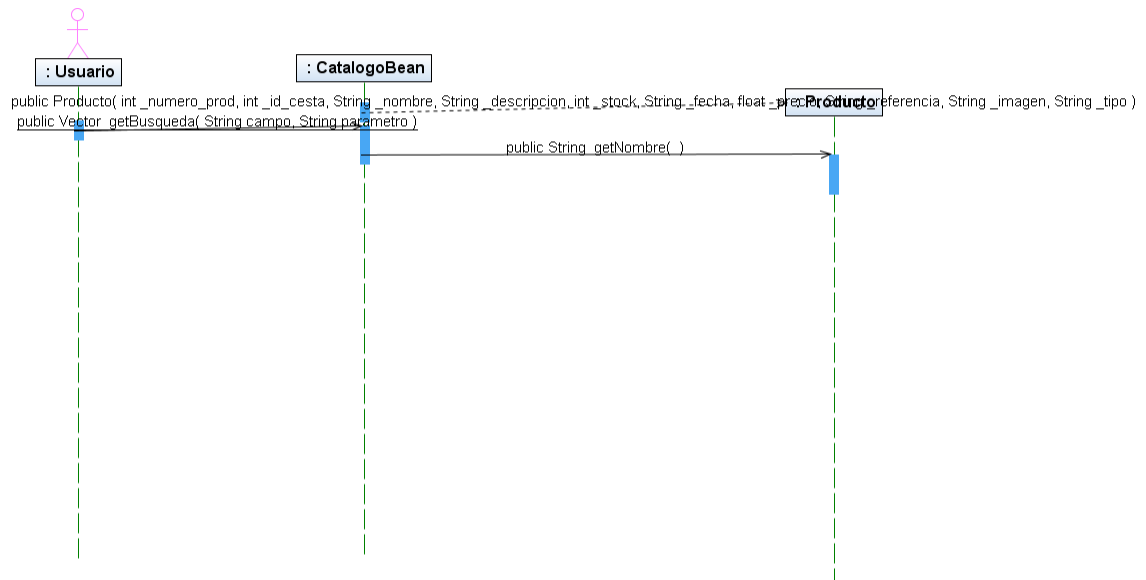


## 6. DIAGRAMAS DE SECUENCIA

### 6.1. SECUENCIA COMPRA

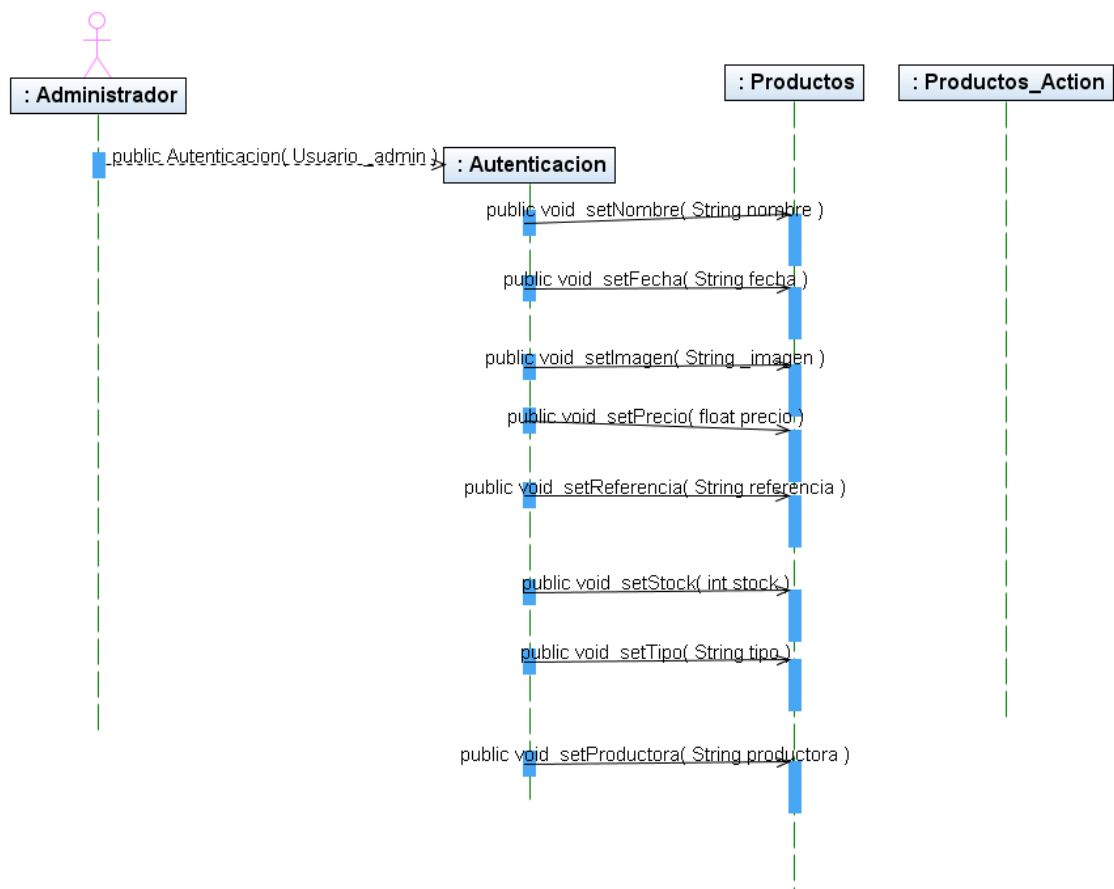


### 6.2. SECUENCIA VER PRODUCTO

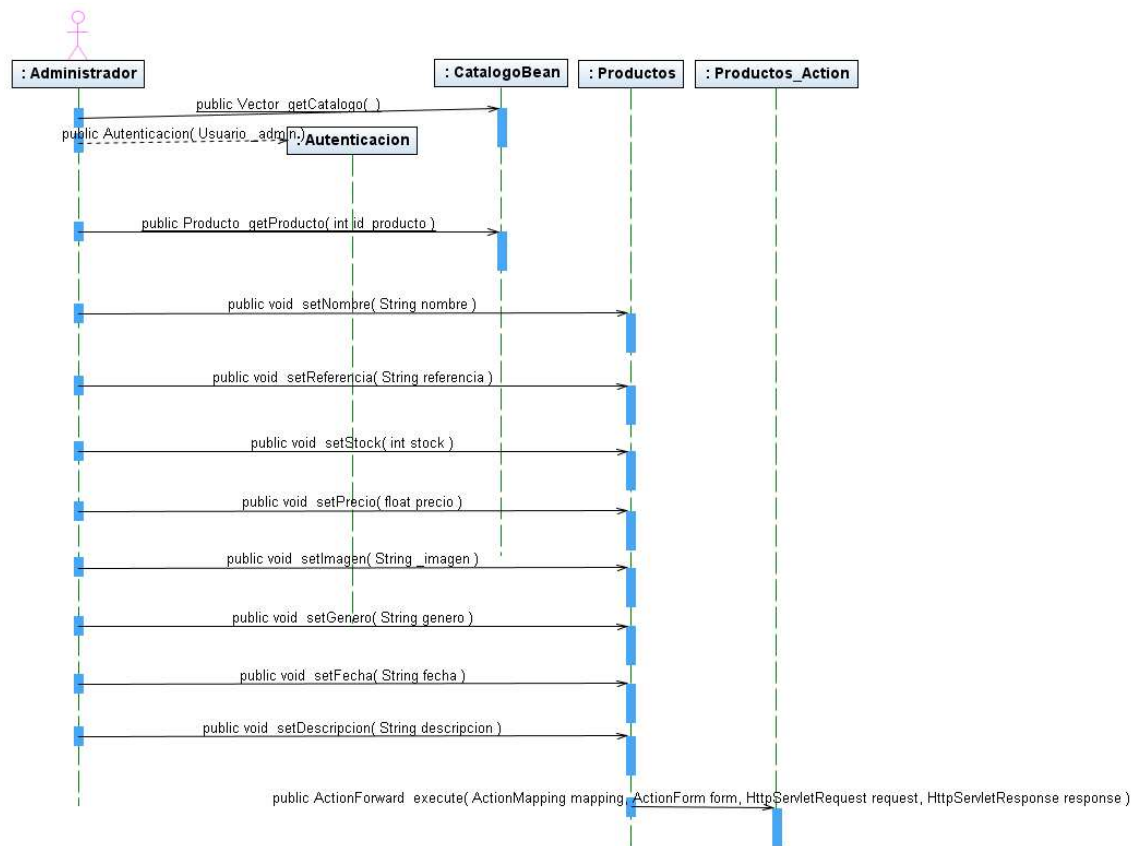




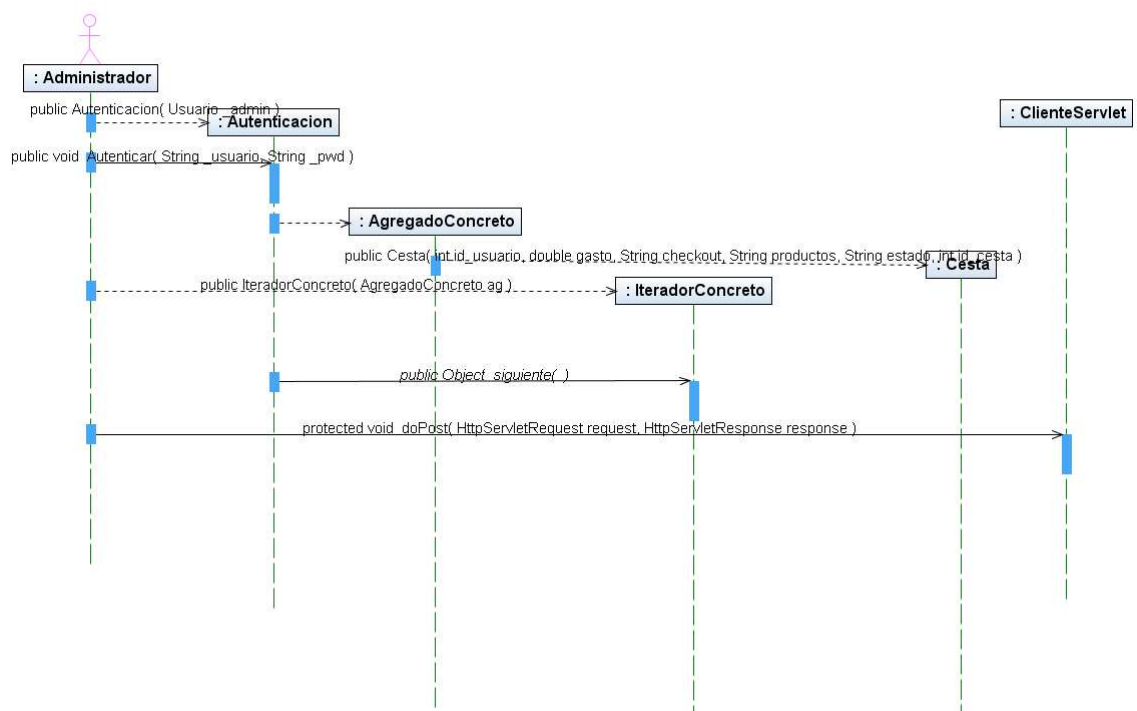
### 6.3. SECUENCIA INTRODUCIR PRODUCTO



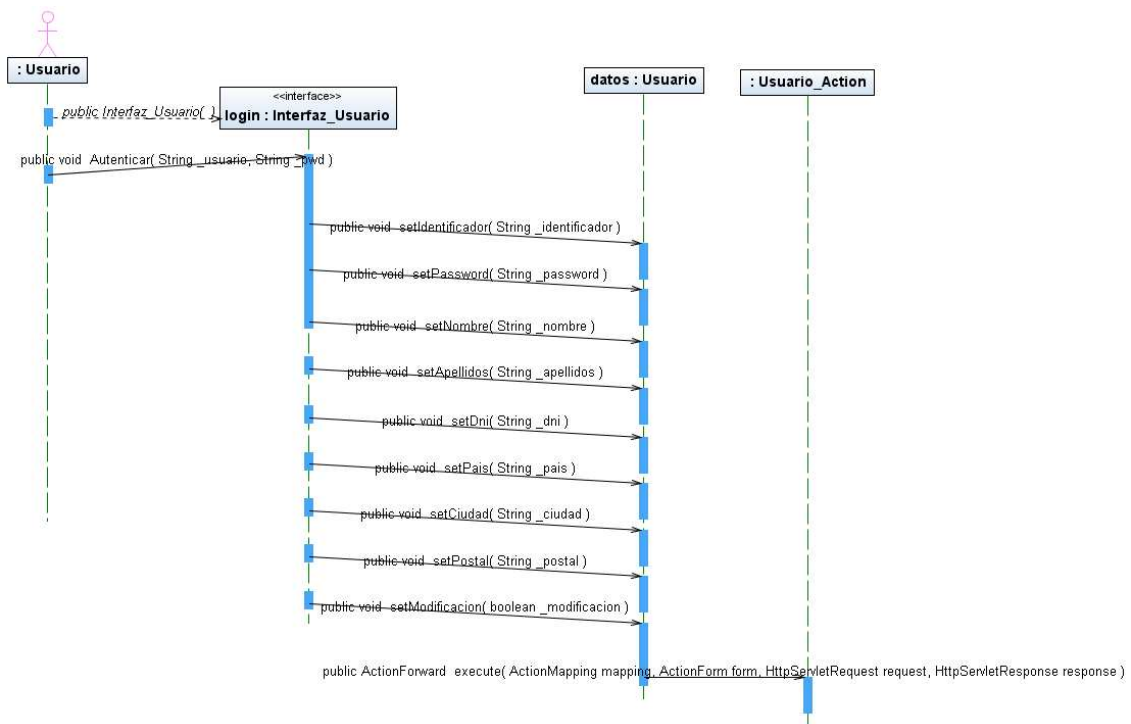
## 6.4. SECUENCIA INTRODUCIR PRODUCTO2



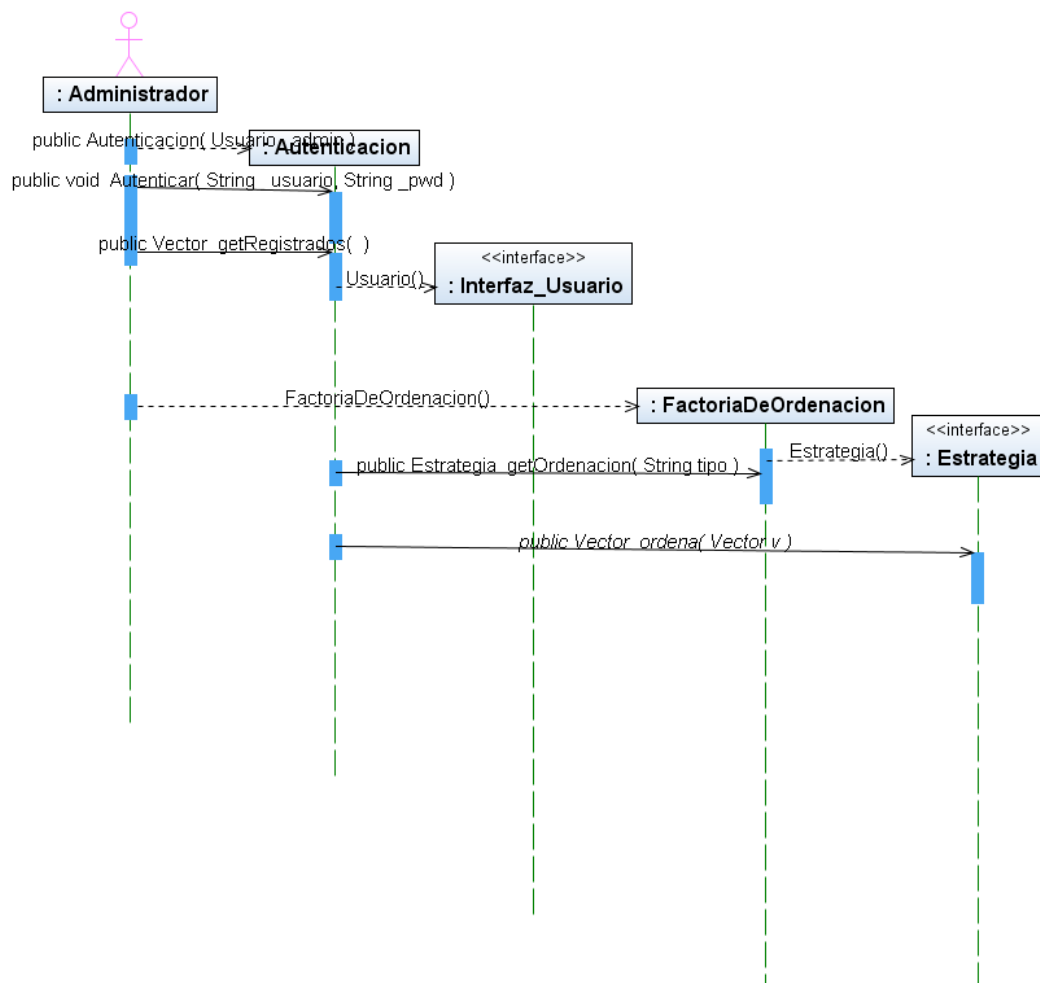
## 6.5. SECUENCIA MODIFICAR ESTADO PEDIDOS



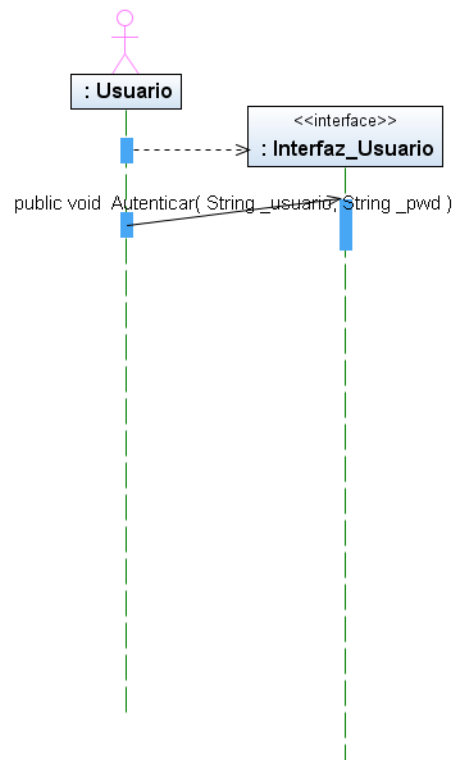
## 6.6. SECUENCIA MODIFICACIÓN DE DATOS



## 6.7. SECUENCIA VER USUARIOS REGISTRADOS



## 6.8. SECUENCIA VER CESTAS

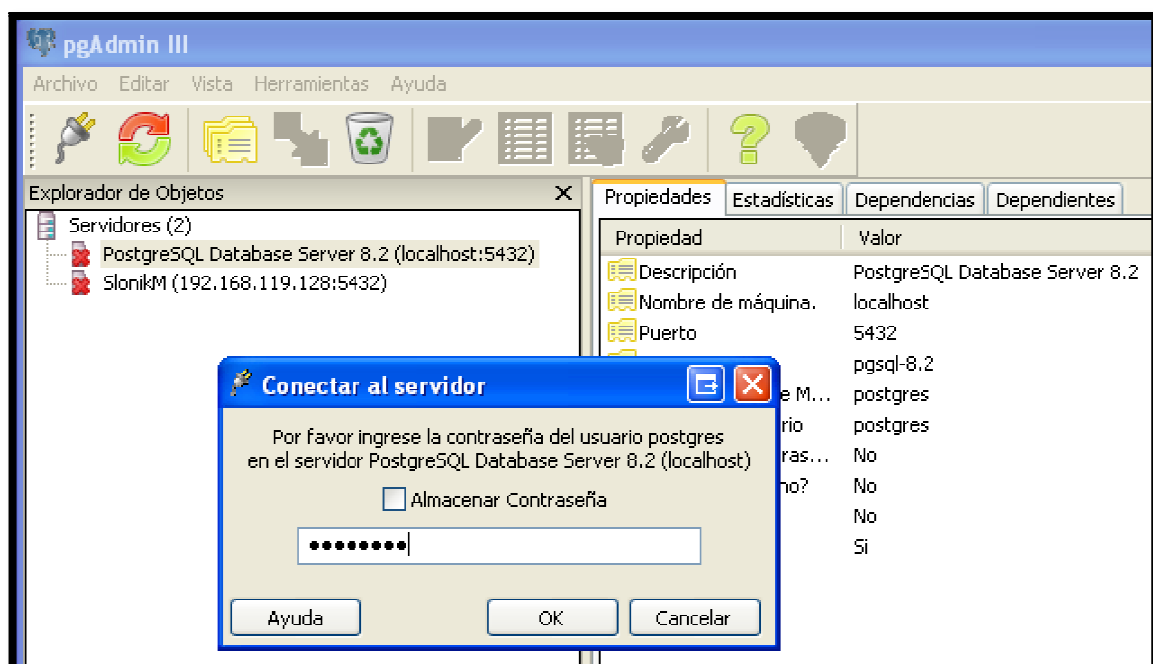


## 7. INSTALACIÓN DE LA APLICACIÓN WEB

### BASE DE DATOS

Esta aplicación Web utiliza una base de datos PostgreSQL, para almacenar tanto los usuarios registrados en el sistema, como el catálogo de productos disponibles, así como las cestas de la compra registradas para cada usuario.

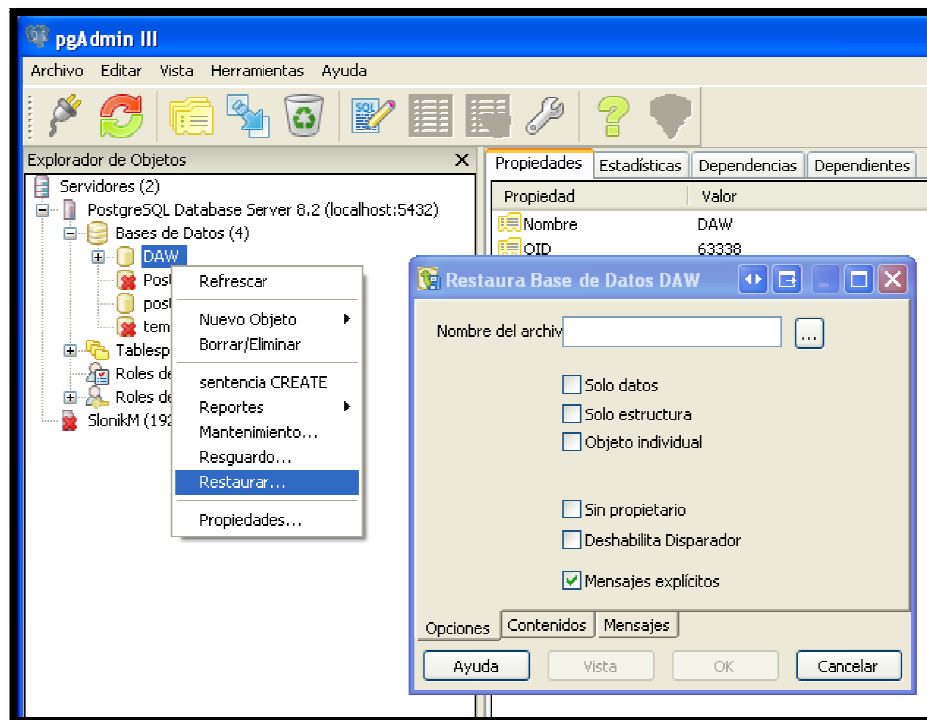
El usuario utilizado para el servidor postgres es “postgres” sin las comillas, y la contraseña “postgres” sin las comillas. Es importante que estos campos usuario y contraseña tengan estos mismos valores, ya que son los que utilizará la aplicación Web para conectar con la base de datos.



A continuación se creará una base de datos de nombre DAW, y con una codificación de archivos UTF8. Es importante establecer estos parámetros con los valores indicados.

Junto con el código de la práctica se suministra un backup de la base de datos, que además de los esquemas contiene valores de tuplas para productos, cestas y usuarios ya introducidos.

Para realizar una restauración de la base de datos, pulsamos con el botón derecho sobre la base de datos recientemente creada y pulsamos de nuevo sobre la orden Restaurar. A continuación seleccionamos el archivo de resguardo y pulsamos OK.



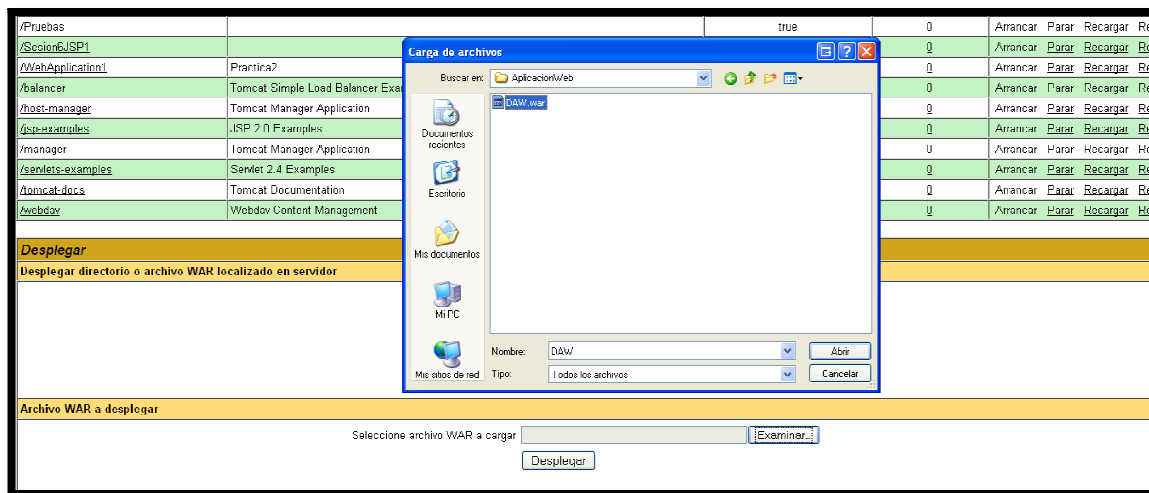
## SERVIDOR APACHE

Esta aplicación funciona sobre el entorno JAVA del servidor Apache Tomcat en su versión 5.5. Dicho software puede obtenerse de manera gratuita de la página web [www.apache.org](http://www.apache.org).

Una vez está instalado correctamente el software, se puede comprobar visitando la página <http://localhost:8080>, accedemos a las opciones de administración (pestaña Tomcat Manager). La contraseña y nombre de usuario por defecto es “admin”, sin las comillas para ambos.

Una vez dentro de esta consola de configuración, se nos muestra una lista de las aplicaciones Web instaladas, o que han sido ejecutadas en el servidor.

Entre los archivos de la práctica se ha suministrado un archivo .war, el cual deberemos cargar en el servidor apache. Para ello disponemos de una opción denominada “Archivo WAR a desplegar”.



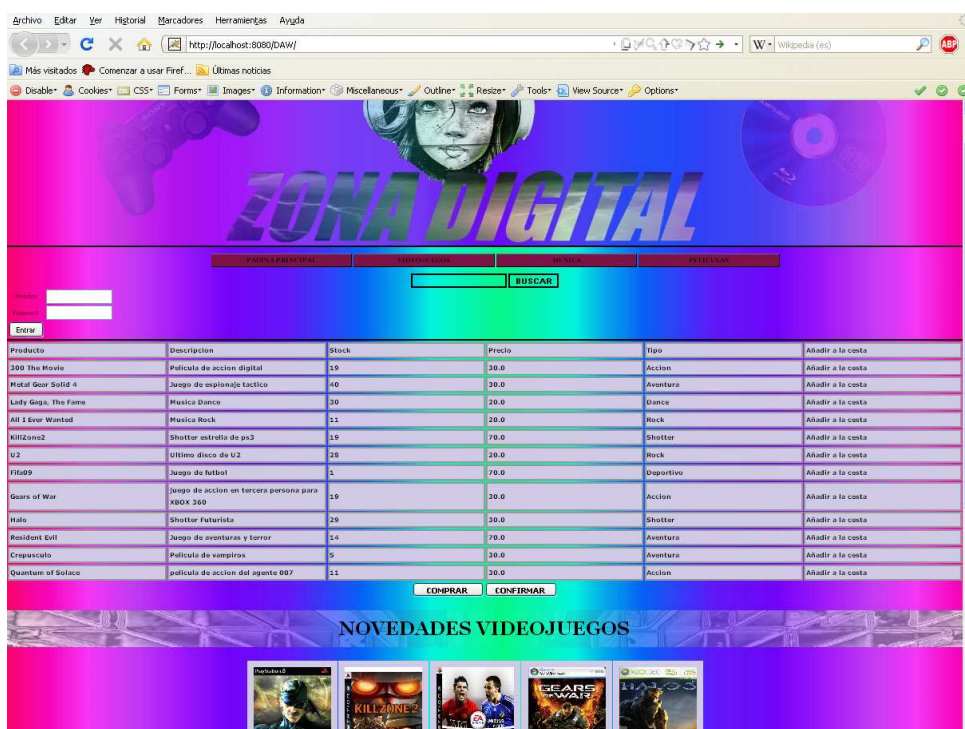
A continuación, seleccionamos el archivo DAW.war suministrado con la práctica, y pulsamos el botón desplegar.

Podemos observar, que aparece una referencia a la aplicación Web en la tabla de aplicaciones de la misma página. Bastará con que pulsemos sobre la referencia para que se ejecute la aplicación Web.

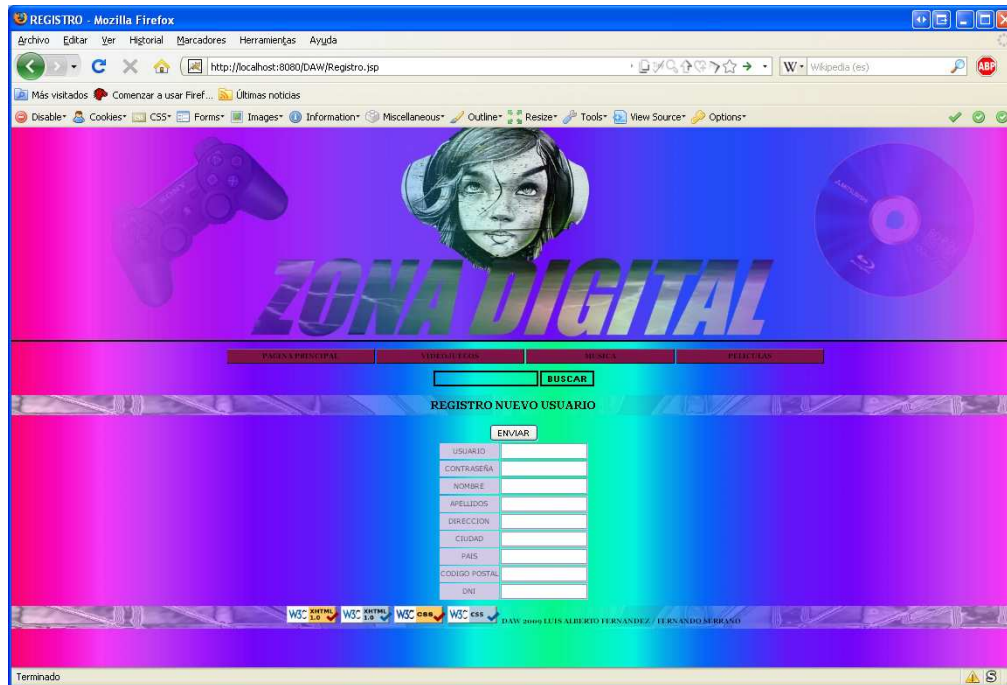
## APLICACIÓN WEB

Ahora se va a proceder a explicar el uso de la aplicación Web desarrollada.

La primera página JSP que aparece será la de Bienvenida, en ella se muestra una lista de los productos más recientemente incluidos en la base de datos, así como la opción de logearse y comenzar a comprar productos.



Si nos logeamos correctamente podremos acceder a una serie de funcionalidades añadidas. Esto será necesario si queremos empezar a comprar productos, ya que en caso de no estar correctamente identificados, la aplicación solicitará que creamos un nuevo usuario para la aplicación, mostrándonos un enlace a la página de registro.



Si nos logeamos como un usuario administrador (usuario: Luis, contraseña: Luis), aparecerá un nuevo submenú en el que se incluyen las funcionalidades exclusivas del administrador, como son las opciones de introducir películas, videojuegos, o música; la opción de modificar el estado de los pedidos, o la de observar la lista de los usuarios registrados y una lista de estadísticas de los usuarios.





Si nos logeamos como un usuario ordinario tan solo aparecerán las opciones de ir a la página principal, ver la lista de videojuegos, música y películas; modificar los datos de usuario; y cerrar la sesión. Estas funcionalidades también están disponibles para el administrador del sistema.

Además se mostrará una lista de las cestas vinculadas al cliente.

**ZONA DIGITAL**

PÁGINA PRINCIPAL | VER VIDEOJUEGOS | MÚSICA | PÉLICULAS | MODIFICAR USUARIO | CERRAR SESIÓN

Buscar

**Bienvenido Irene**

CESTA Nº 1: 10 ITEMS (10.000)

PRODUCTOS (10 ITEMS) EN PROCESO

CESTA Nº 2: 10 ITEMS (10.000)

PRODUCTOS (10 ITEMS) EN PROCESO

CESTA Nº 3: 10 ITEMS (10.000)

PRODUCTOS (10 ITEMS) EN PROCESO

| Producto            | Descripcion                                      | Stock | Precio | Tipo      | Añadir a la cesta |
|---------------------|--|-------|--------|-----------|-------------------|
| 300 The Movie       | Película de accion digital                       | 19    | 30.0   | Accion    | Añadir a la cesta |
| Metal Gear Solid 4  | Juego de espionaje tactico                       | 40    | 30.0   | Aventura  | Añadir a la cesta |
| Lady Gaga, The Fame | Musica Dance                                     | 30    | 20.0   | Dance     | Añadir a la cesta |
| All I Ever Wanted   | Musica Rock                                      | 11    | 20.0   | Rock      | Añadir a la cesta |
| KillZone2           | Shooter estrella de ps3                          | 19    | 70.0   | Shooter   | Añadir a la cesta |
| U2                  | Ultimo disco de U2                               | 28    | 20.0   | Rock      | Añadir a la cesta |
| Fifa09              | Juego de futbol                                  | 1     | 70.0   | Deportivo | Añadir a la cesta |
| Gears of War        | Juego de accion en tercera persona para XBOX 360 | 19    | 30.0   | Accion    | Añadir a la cesta |
| Halo                | Shooter Futurista                                | 29    | 30.0   | Shooter   | Añadir a la cesta |
| Resident Evil       | Juego de aventuras y terror                      | 14    | 70.0   | Aventura  | Añadir a la cesta |
| Crepusculo          | Película de vampiros                             | 5     | 30.0   | Aventura  | Añadir a la cesta |
| Quantum of Solace   | película de accion del agente 007                | 11    | 30.0   | Accion    | Añadir a la cesta |

COMPRAR | CONFIRMAR

**NOVEDADES VIDEOJUEGOS**

## COMPRAR UN PRODUCTO

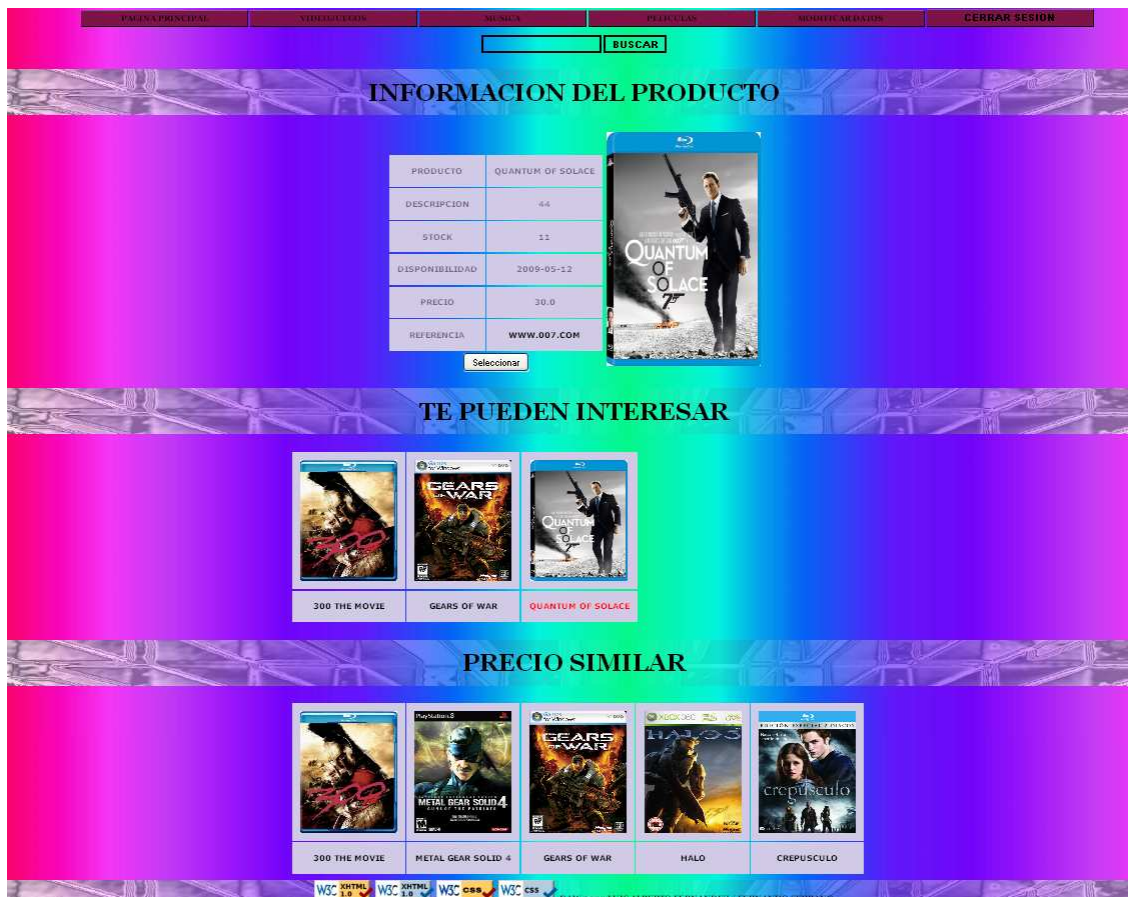
Una vez estamos autenticados en el sistema, podemos proceder a comprar un producto.

Cuando un usuario se autentica, automáticamente se crea una cesta de la compra para el mismo, de esta manera solo tiene que ir añadiendo productos a la misma.

En primer lugar seleccionamos un producto, bien en la lista de productos, bien en la sección de novedades, o bien en las páginas correspondientes a videojuegos, música y películas, accesibles desde el menú.

Una vez hemos seleccionado el producto, se muestra una nueva página con los datos del producto, así como una lista de productos relacionados por género y precio similar al producto seleccionado.

Para introducir el producto en la cesta pulsamos el botón seleccionar. Si el usuario es administrador del sistema, se mostrará además un segundo botón para modificar la información del producto en la base de datos.



Una vez hecho esto, la aplicación nos redirige a la página principal. Ahora podemos seguir comprando, eliminar productos seleccionados de la cesta de la compra, o finalizar la compra.

Si queremos finalizar la compra, en la página principal se nos muestran dos opciones bajo la lista de productos:

- **COMPRAR:** realiza una compra inmediata, realizando el CHECKOUT de la lista.
- **CONFIRMAR:** se realiza un paso intermedio de confirmación de los productos introducidos en la cesta de la compra.

Tanto en uno como en otro se solicita al cliente que verifique los datos de envío del producto.



Una vez realizada la compra la aplicación nos redirigirá a la página principal.

Si se desean eliminar productos de la base de datos, en la página principal pulsamos sobre el botón confirmar. En la nueva página que aparece, dispondremos de un botón junto a cada producto para su eliminación individual, y un botón para la eliminación de todo el contenido de la cesta de la compra.



## MODIFICAR DATOS DE USUARIO

La función de modificar los datos personales está disponible para todos los usuarios. Para acceder a ella se debe pulsar sobre el botón Modificar Datos del menú.

A continuación se muestra un formulario con los datos del usuario, sobre lo cuales podrá realizar las modificaciones que considere oportunas. Una vez realizados los cambios se pulsará el botón Enviar.

## INTRODUCIR PRODUCTOS

Si el usuario se autentica como administrador del sistema dispondrá de las opciones de introducir productos. Para ello seleccionamos del menú cualquiera de las opciones Introducir Música, Introducir Videojuego, o Introducir Película.

Aparecerá un formulario Web en el que se deben rellenar los campos con los que posteriormente se introducirá el nuevo producto en la base de datos.

Nótese que la ruta de las imágenes es relativa a la carpeta Imágenes del proyecto Web, por lo tanto solo se deberá especificar el nombre del archivo, por ejemplo MetalGear.jpg.


Tanto si la fecha introducida por el usuario tiene un formato inválido, como si no se introduce fecha, se introducirá por defecto la fecha actual del sistema.



## CONSULTAR USUARIOS REGISTRADOS

Otra de las funcionalidades añadidas que posee el administrador del sistema, es obtener una lista de los usuarios registrados en el sistema, así como datos estadísticos de los mismos, como cestas vinculadas a los mismos, gasto acumulado, etc.

En esta página se proporciona un sistema de ordenación por nombre, apellidos y gasto acumulado en la aplicación. Para realizar una ordenación bastará con pulsar sobre la cabecera de la celda correspondiente al campo por el cual queremos realizar la ordenación.



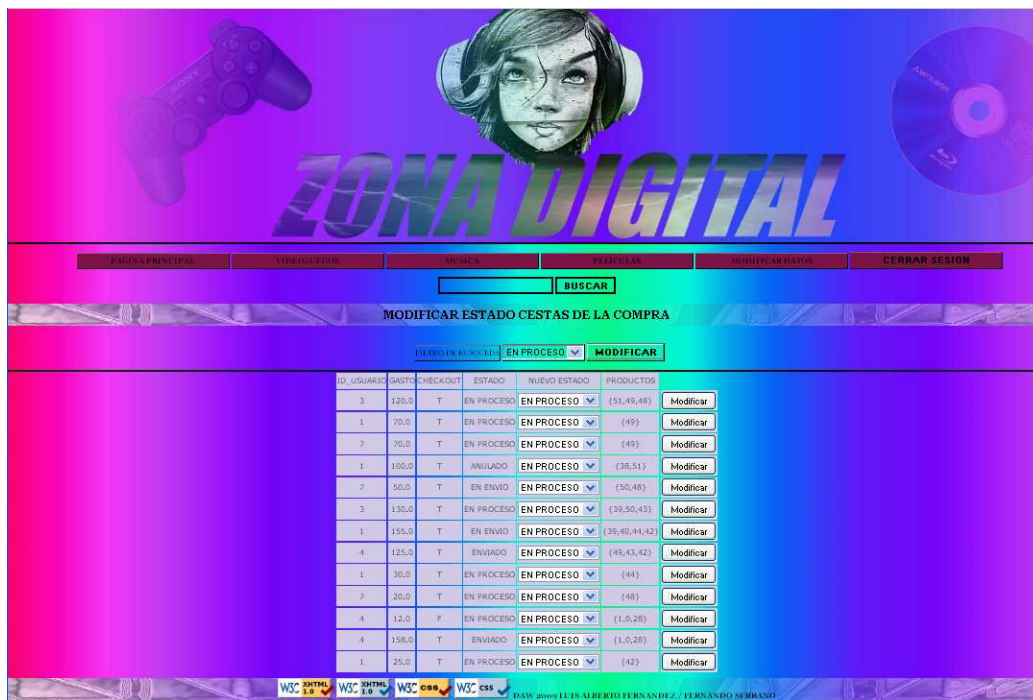
| INICIAR SESION   | VIDEOS/JUEGOS | MUSICA   | PEDIDOS   | MODIFICAR DATOS | CERRAR SESION |
|--|---------------|----------|---|-----------------|---------------|
| <input type="text"/> <input type="button" value="BUSCAR"/> |               |          |   |                 |               |
| NOMBRE   | APELLIDOS     | DNI      | PEDIDOS   | CODIGO POSTAL   | TOTAL GASTADO |
| Carlos   | Cepelini      | 31348765 | CESTA Nº: 6, GASTO: 120 Euros<br>PRODUCTOS(51,49,48), ESTADO: EN PROCESO<br>CESTA Nº: 11, GASTO: 130 Euros<br>PRODUCTOS(39,50,43), ESTADO: EN PROCESO   | 19005           | 250,0         |
| Fernando   | Serrano       | 31299999 | CESTA Nº: 5, GASTO: 125 Euros<br>PRODUCTOS(49,43,42), ESTADO: ENVIADO<br>CESTA Nº: 0, GASTO: 12 Euros<br>PRODUCTOS(1,0,28), ESTADO: EN PROCESO<br>CESTA Nº: 3, GASTO: 158 Euros<br>PRODUCTOS(1,0,28), ESTADO: ENVIADO | 19005           | 295,0         |
|  |               |          | CESTA Nº: 4, GASTO: 10,5 Euros  |                 |               |

## MODIFICAR ESTADO DE PEDIDOS

Para permitir una correcta gestión de los pedidos de la tienda, el administrador de la misma dispondrá de una interfaz desde la cual podrá modificar el estado en el que se encuentran los diferentes pedidos.

Sobre la tabla donde se muestran los diferentes pedidos podemos seleccionar un filtro para que solo se muestren los pedidos en un determinado estado. A continuación pulsamos en botón modificar y se realizara el filtrado.

Una vez hemos cambiado el estado de un pedido pulsamos el botón modificar que se encuentra a la derecha de dicho pedido.



## FUNCION DE BUSQUEDA

La aplicación proporciona un sistema de búsqueda que realiza una comprobación utilizando la cadena introducida sobre multitud de atributos de los productos del catálogo. Si por ejemplo introducimos la cadena “a”, no se limitará a mostrarnos solamente los productos cuyo nombre contiene la letra a, sino también los productos pertenecientes al género de aventura, o acción, así como coincidencias en multitud de campos.

