Aspectos a mejorar en el código de la práctica 4

# Codificación de la capa de persistencia

## Implementación muy ineficiente del método getPaginadorArticulos()

Algunos de vosotros habéis implementado el método getPaginadorArticulos llamando al método getArticulos() de la clase paw.bd.GestorBD que yo os proporcionaba, para recuperar la lista de todos los artículos y luego usar el tamaño de esa lista para construir el paginador.

public static Paginador getPaginadorArticulos**(**int tamanioPagina**)** **throws** ExcepcionDeAplicacion **{**

paw**.**bd**.**GestorBD s **=** **new** paw**.**bd**.**GestorBD**();**

int registros **=** s**.**getArticulos**().**size**();**

**return** **new** Paginador**(**registros**,** tamanioPagina**);**

**}**

Es una solución muy ineficiente porque, lo que se podría hacer con una consulta muy simple ("select count(\*) from articulo"), lo estáis resolviendo con una consulta que recupera los datos de todos los artículos de la BD (447) y construye un montón de objetos (447). La solución es cómoda para el programador, pero muy ineficiente para la aplicación (“lo que hará que vuestro jefe se vuelva a disgustar mucho”).

## Otra implementación ineficiente del método getPaginadorArticulos()

Esta es un poco menos ineficiente que la anterior, pero también es criticable. Algunos de vosotros habéis implementado el método getPaginadorArticulos haciendo una consulta "select codigo from articulo" y contando luego el número de códigos obtenidos.

int numArticulos **=** 0**;**

Statement stmt **=** con**.**createStatement**();**

ResultSet rs **=** stmt**.**executeQuery**(**"SELECT CODIGO FROM ARTICULO"**);**

**while (**rs**.**next**())** **{**

numArticulos**++;**

**}**

Es una solución ineficiente porque el conteo de los artículos lo hace de forma mucho más eficiente el SGBD, usando count(\*): "select count(\*) from articulo". El SGBD cuenta mucho más rápido que Java. Esta optimizado para ello.

int numArticulos **=** 0**;**

Statement stmt **=** con**.**createStatement**();**

ResultSet rs **=** stmt**.**executeQuery**(**"SELECT count(\*) FROM ARTICULO"**);**

**if (**rs**.**next**())** **{**

numArticulos **=** rs.getInt(1)**;**

**}**

# Programación en general

## Conversión de cadenas a números

No se puede presuponer nada en informática. Por ejemplo, no se puede presuponer que el parámetro de configuración del tamaño de la página en catalogo.jsp vaya a estar bien escrito (con “forma” de número), ni que en la petición a catalogo.jsp vayamos a recibir un parámetro de página que sea un número.

Por ello, las conversiones de texto a número que se hacen en la página catalogo.jsp deben estar dentro de un bloque try-catch, procurando que, si ocurriese algún error, las variables numéricas tomasen algún valor seguro:

int tamanioPagina **=** 15**;**

**try** **{**

tamanioPagina **=** Integer**.**parseInt**(this.**getInitParameter**(**"tamnioPagina"**));**

**}** **catch** **(**Exception e**)** **{**

**}**

...

int pagina **=** 1**;**

**try** **{**

pagina **=** Integer**.**parseInt**(**request**.**getParameter**(**"pag"**));**

**}** **catch** **(**NumberFormatException ex**)** **{**

**}**

## Uso de métodos static: no hay que construir objetos

Os pedía que hicieseis una clase (sol.GestorBD) con dos métodos static

public static List**<**Articulo**>** getArticulos**(**int pagina**,** int tamanioPagina**)** **throws** ExcepcionDeAplicacion

public static Paginador getPaginadorArticulos**(**int tamanioPagina**)** **throws** ExcepcionDeAplicacion

Para usar esos métodos no hay que crear ningún objeto de la clase.

sol**.**GestorBD gbd2 **=** **new** sol**.**GestorBD**(); 🡨 NO**

gbd2**.**getPaginadorArticulos**(...**

Se usan directamente sobre el nombre de la clase, como cualquier otra cosa que sea static.

sol**.**GestorBD**.**getPaginadorArticulos**(... 🡨 SI**

## No repetir código innecesariamente

Mirad este trozo de código:

**<%**

GestorBD gbd = **new** GestorBD();

**if** (request.getParameter("cart") == null || request.getParameter("cart").**equals**("")) {



response.sendRedirect("index.html");



} **else** {



**if** (gbd.getArticulo(request.getParameter("cart")) == null) {



response.sendRedirect("error.jsp?msg=" + URLEncoder.encode("El artículo no existe", "UTF-8"));



} **else** {

Articulo a = gbd.getArticulo(request.getParameter("cart"));

**%>**



O este otro

**<%**

Paginador paginador = GestorBD2.getPaginadorArticulos(Integer.parseInt(config.getInitParameter("tamanioPagina")));



**if** (request.getParameter("p") == null || request.getParameter("p").trim().length() == 0 || Integer.parseInt(request.getParameter("p")) < 1) {



response.sendRedirect("catalogo.jsp?p=1");



**return**;

}



**if** (Integer.parseInt(request.getParameter("p")) > paginador.getNumPaginas()) {



response.sendRedirect("catalogo.jsp?p=" + paginador.getNumPaginas());



**return**;

}



int pag = Integer.parseInt(request.getParameter("p"));



List<Articulo> articulos = GestorBD.getArticulos(pag, Integer.parseInt(config.getInitParameter("tamanioPagina")));



...



**%>**

Además de otros “fallos” que ya he comentado en otros apartados, he subrayado las veces que se repiten las instrucciones Integer.parseInt, config.getInitParameter y request.getParameter. Estas repeticiones acarrean estos problemas:

* Ineficiencia. Aunque sea poco el tiempo de ejecución asociado a la lectura de un parámetro de la queyString, o a la conversión de una cadena en entero, no nos podemos permitir el lujo de repetir tantas veces la ejecución de estos métodos. Es tiempo perdido innecesariamente. Especialmente grave es la repetición de gbd.getArticulo, ya que implica repetir una consulta a la BD.
* El código es más difícil de leer
* El código es más difícil de escribir

Ejecutad una única vez esas operaciones, usad variables para almacenar el resultado y utilizad esas variables en el programa. Más claro, y más eficiente, es algo así:

**<%**

int tamPag = Integer.parseInt(config.getInitParameter("tamanioPagina"));

Paginador paginador = GestorBD2.getPaginadorArticulos(tamPag);

**String** ps = request.getParameter("p");

int pag = 1;

**if** (ps == null || ps.trim().length() == 0) {

response.sendRedirect("catalogo.jsp?p=1");

**return**;

} **else** {

**try** {

pag = Integer.parseInt(ps);

} **catch** (NumberFormaException ex) {}

}

**if** (pag < 1) {

response.sendRedirect("catalogo.jsp?p=1");

**return**;

}

**if** (pag > paginador.getNumPaginas()) {

response.sendRedirect("catalogo.jsp?p=" + paginador.getNumPaginas());

**return**;

}

List<Articulo> articulos = GestorBD.getArticulos(pag, tamPag);

...

**%>**

# JSP / HTML

## Contenidos de los ficheros auxiliares de cabecera y el pie

Estos ficheros deben contener solamente lo siguiente:

**Cabecera**

<div class=**"logo"**><a href=**"index.html"**><img src=**"img/LogoElectrosa200.png"** border=**"0"**></a></div>

<div class=**"sombra"**>

**...**

</div>

<div class=**"barra\_menus"**>

**...**

</div>

**Pie**:

<div class=**"pie"**>

**...**

</div>

**MUY IMPORTANTE:**

No deben contener ni <!DOCTYPE html>, <html> ni <head> ni <body>.

Pensad qué pasaría si contuviesen esas etiquetas (o alguna o algunas de ellas). Es decir, si la cabecera fuese, por ejemplo:

**<!DOCTYPE html>**

<html>

<body>

<div class=**"logo"**><a href=**"index.html"**><img src=**"img/LogoElectrosa200.png"** border=**"0"**></a></div>

<div class=**"sombra"**>

**...**

</div>

<div class=**"barra\_menus"**>

**...**

</div>

</body>

</html>

Al incluir este fichero en un JSP, por ejemplo como este:

**<%@**page contentType="text/html" pageEncoding="UTF-8"**%>**

**<!DOCTYPE html>**

<html>

<head>

<meta http-equiv=**"Content-Type"** content=**"text/html; charset=UTF-8"**>

<title>**JSP Page**</title>

</head>

<body>

**<%@** include file='cabecera.html'**%>**

<div class=**"sombra"**>

<div class=**"nucleo"**>**...** </div>

**...**

</body>

</html>

El resultado del procesamiento sería:

**<!DOCTYPE html>**

<html>

<head>

<meta http-equiv=**"Content-Type"** content=**"text/html; charset=UTF-8"**>

<title>**JSP Page**</title>

</head>

<body>

**<!DOCTYPE html>**

<html>

<body>

<div class=**"logo"**><a href=**"index.html"**><img src=**"img/LogoElectrosa200.png"** border=**"0"**></a></div>

<div class=**"sombra"**>

**...**

</div>

<div class=**"barra\_menus"**>

**...**

</div>

</body>

</html>

<div class=**"sombra"**>

<div class=**"nucleo"**>**...** </div>

**...**

</body>

</html>

Fijaos en que, en medio del cuerpo de la página resultante aparece un <!DOCTYPE html> y un <html> y un <body>, lo cual es obviamente erróneo.

## Usar cabecera y pie en todas las páginas JSP

Bastantes de vosotros habéis usado las directivas de inclusión de cabecera y pie sólo en la página de catalogo.jsp. Hay que usarlas en todas las páginas JSP de la aplicación.

## Usar <jsp:useBean de forma consistente

Aunque no lo habéis hecho todos, está bien que hayáis usado jsp:useBean para crear un objeto GestorBD en el scope application para evitar crear objetos en scriptlets. Es decir:

EVITAR:

**<%**

GestorBD gbd = **new** GestorBD();

**%>**

USAR:

<jsp:useBean id="gbd" class="paw.bd.GestorBD" scope="application"></jsp:useBean>

Pero sirve de poco hacerlo sólo en una única página. Lo que tenéis que hacer es usar jsp:useBean siempre. Así estaréis optimizando memoria porque creáis un único objeto GestorBD para toda la aplicación. Como los objetos de esta clase no tienen estado, no hay peligro en hacerlo. Para que la optimización sea máxima, el scope usado en jsp:useBean debe ser application.

## Redirección ante errores de paginación

Algunos de vosotros habéis programado a medias la validación de las peticiones de cambio de página en catalogo.jsp. Habéis comprobado que el parámetro de la página pueda convertirse a un entero, y que éste esté entre 1 y el número total de páginas, y habéis corregido el valor en caso contrario, mostrando la información correspondiente a la página corregida. Pero eso no es suficiente ya que, ante una petición de página “-10” mostrarías información correcta de la página 1, pero la barra de direcciones indicaría que la página es la -10.

Lo que hay que enviar una respuesta de redirección al navegador para que vuelva a hacer la petición con el número de página corregido. Esta es la única manera de hacer que la dirección mostrada en la barra de direcciones del navegador corresponda con la información visualizada en la página.

## Comprobar el mensaje en la página de error

Casi nadie ha comprobado que el valor del parámetro con el mensaje de error no sea vacío, asignándole un mensaje por defecto si eso ocurre. Esto sería lo correcto:

**<%**

**String** msg = request.getParameter("msg");

**if** (UtilesString.isVacia(msg))

msg = "Error de aplicación";

**%>**

<div class=**"error"**>**<%=** msg**%>**</div>