Ejercicios de JSP y Javabeans

Índice

1. Conversor JSP (0,5 puntos)

Crear una aplicación llamada "conversor" que sirva para convertir de pesetas en euros y viceversa. La aplicación será una única página JSP a la que:

- Si se le llama sin que exista un parametro HTTP `numero`, mostrará el formulario para introducir la cantidad de euros
- Si se la llama con un parámetro HTTP `numero` tomará el valor del parámetro, realizará la conversión euros/pesetas y la mostrará en pantalla.

Una vez realizada la parte básica, añadirle tratamiento de errores, de manera que si se produce una excepción en la página `conversor.jsp` se salte a la página de error, `error.jsp` (que también debéis crear). En caso de que la cantidad que representa el parámetro `numero` sea negativa lanzar una excepcion con el mensaje de error deseado (`throw new Exception("La cantidad no es correcta")`). Imprimir dicho mensaje desde la página de error.

2. Contador de visitas (0,5 puntos)

Crear un fragmento de código JSP que sirva como contador de visitas. El objetivo es poder incluir este código en otras páginas JSP, por lo que no es necesario (ni conveniente) que sea una página web completa, sino solo el texto del contador. Este debe aparecer con texto HTML en negrita similar al siguiente

Esta página ha sido visitada X veces

Valorar si sería más conveniente emplear la directiva `include` o la acción del mismo nombre. El código debe poder incluirse en distintas páginas de forma que cada una tenga un contador propio.

3. Contador avanzado (0,5 puntos)

Una vez se tenga funcionando el contador básico, mejorarlo haciendo que el número de visitas se conserve, aunque el servidor se apague o se descargue el JSP de la memoria. Para ello, tener en cuenta que:

- El número de visitas se puede guardar en un fichero con el nombre de la página. Para simplificar, se puede suponer que en la página principal se define una variable "nombrePag" de tipo String con el nombre del fichero. No obstante, este se podría obtener a partir del método `getRequestURI()` del objeto implícito `request` (este método devuelve la URI completa de la página actual, de la que habrá que extraer el nombre del fichero).

Nota

> Sólo tenemos acceso al objeto `request` una vez realizada una petición, por lo que no podremos utilizarlo desde el método `jspInit`. En su lugar podríamos utilizar, por ejemplo, el nombre de la clase actual.

- Habrá que sobreescribir los métodos `public void jspInit()` y `public void jspDestroy()`, que se ejecutan, respectivamente, cuando la página se carga por primera vez y cuando se destruye por falta de memoria o porque se apaga el servidor. Al llamar a `jspDestroy` se debe guardar el número de visitas en el fichero, y cargarlo cuando se realice el `jspInit`.

4. Chat con JSPs (1 punto)

En la sesión anterior implementamos un chat con *servlets*. Hemos visto que los JSP son más adecuados para la presentación. Por lo tanto, vamos a modificar el chat para que la lista de mensajes sea generada por un JSP, en lugar de un *servlet*. Se pide:

a) Crearemos un directorio `jsp` que contendrá aquellos JSP que utilizan los servlets de la aplicación para presentar la información producida. Estos JSP serán:

`listaMensajes.`	Mostrará los mensajes publicados en el chat en forma de documento HTML. Debe esperar recibir un atributo `org.especialistajee.cw.chat.mensajes` en el ámbito de la petición (`request`), con la lista de mensajes publicados en el chat.
`error.jsp`	Mostrará un mensaje de error y permitirá volver a la página de *login*. Espera recibir un atributo `org.especialistajee.cw.chat.error` en el ámbito de la petición (`request`), con el mensaje de error que deberá mostrar.

b) Implementar el JSP `listaMensajes.jsp`. Modificar el *servlet* `ListaMensajesServlet` para que en lugar de generar el mismo el HTML, redirija mediante `forward` la petición al JSP que acabamos de crear. En el *servlet* se deberá obtener la lista de mensajes y le deberá pasar este objeto al JSP como atributo de la petición, como se ha descrito anteriormente.

c) Implementar el JSP `error.jsp`. Modificar el *servlet* `EnviaMensajeServlet` para que en lugar redirigir a `error.html` en caso de error, lo haga a `error.jsp`, pasando como atributo de la petición una cadena con el mensaje de error que deberá mostrar esta página.

5. Bean cronómetro (0,5 puntos)

Crear un *bean* denominado **`org.especialistajee.cw.beans.CronoBean`** que sirva como "cronómetro". Cuando el *bean* se inicialice, debe guardar internamente el momento de su creación. El *bean* tendrá una propiedad `segundos`, que devolverá el número de segundos transcurridos desde su creación. Para calcular tiempos, se puede usar el método `System.currentTimeMillis()`, que devuelve el número de milisegundos transcurridos

entre el instante actual y el 1/1/1970.

a) Probar el bean en una página llamada **cronoini.jsp** que lo inicialice (mostrando un mensaje HTML que indique que se está inicializando) y muestre el número de segundos transcurridos.

b) Hacer que el *bean* sólo se inicialice si no estuviese inicializado ya. Si ya estuviese inicializado simplemente se deberá mostrar el número de segundos transcurridos.

c) Acceder al *bean* desde otra página **crono2.jsp** que muestre el mismo temporizador (¿qué ámbito debe tener el *bean* para funcionar correctamente?).