MASTER EN MODELIZACIÓN MATEMÁTICA, ESTADÍSTICA Y COMPUTACIÓN
2009-2010

Curso: Bases de datos y programación orientada a objetos
Parte 2 -

Nota previa.

Este texto explica como "montar" **un ejemplo concreto** muy sencillo de Aplicación Web de acceso a una base de datos. Para hacerlo funcionar correctamente contamos con que el servidor de bases de datos MySQL está activo en la misma máquina, y la base de datos con la que se trabaja es la vista en la primera parte del curso. A pesar de ser **un ejemplo concreto**, se "tocan" todos los recursos necesarios (HTML, JSP, CSS,…) o cuando menos se mencionan indicando su funcionalidad si quedan muy lejos del ámbito del curso (DTD, EJB,…). La pretensión es que el alumno tenga una visión tanto concreta como global de las técnicas necesarias de modo que sea capaz de "extender" el ejemplo tanto como precise con la ayuda de la ingente cantidad de información disponible en Internet sobre esta temática (documentos "oficiales", cursos, tutoriales, foros con soluciones a dificultades a todos los niveles, etc.)

Acceso a base de datos desde la web

Para permitir el acceso a una base de datos con un navegador web como capa de usuario (es decir utilizando el lenguaje HTML para componer el interfaz y el protocolo de comunicación HTTP para comunicarlo con el software que nos permitirá el acceso) necesitaremos de un Servidor de Aplicaciones o cuando menos de un "Servidor Web dinámico". Nos quedaremos con el caso más sencillo, el Servidor Web, y en el caso que nos ocupa –programando en Java-, es razonable pensar en el servidor Tomcat, que además de ser la implementación de referencia[1] es gratuito. Para instalar Tomcat basta con descargarlo de la web (se localiza con Google de inmediato) y ejecutar el instalador. También puede obtenerse asociado a Netbeans en una de sus distribuciones para descarga (la denominada simplemente "Java"). Algunos sistemas Linux lo incluyen como servicio estándar que podremos activar.

Una vez que dispongamos de nuestro servidor, lo que haremos será escribir las páginas que el usuario, desde el navegador, va a utilizar. En principio pueden ser una mezcla de ficheros "HTML" y "JSP", aunque por simplificar podemos pensar en todos como JSP, ya que estos son en realidad páginas con zonas de código java que se ejecuta y terminan por generar "HTML". De este modo una página "HTML" es equivalente a una "JSP" sin partes ejecutables (con algún cambio puntual de sintaxis).

La programación que incluiremos en nuestras "JSP" es la misma que se ha visto para el acceso a bases de datos independientemente de la web.

Para entender todo el mecanismo de interacción con la base de datos desde el navegador necesitaremos entender, además de las "JSP", los formularios de "HTML", y un poco de "Javascript" (programación que ejecuta el mismo navegador) para controlar que la cumplimentación de los mismos sea correcta.

[1] Que sea la implementación de referencia significa lo siguiente: cualquier fabricante que quiera desarrollar su propio servidor "compatible con Java" (certificado como tal) podrá incluir todas las capacidades que quiera, pero para cualquier característica que presente ya Tomcat habrá de ocuparse de que sea idéntica a la de éste servidor.

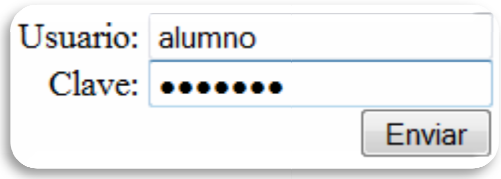
Formularios en html

Ya sabemos que un fichero HTML tiene una estructura general marcada por las etiquetas <html></html> dentro de la cual hay dos zonas, la cabecera (head) y el cuerpo (body), y que es dentro de este último donde se sitúan los elementos visibles de las páginas. Uno de estos elementos es el "formulario". Un formulario se parentiza con las marcas <form></form>, y dentro de él se sitúan los diferentes campos que el usuario puede rellenar (además de otros elementos como textos fijos, imágenes, etc.). Junto a la etiqueta "form" se especifican ciertos atributos que indican cómo y a dónde se enviará la información introducida en el formulario.

Un ejemplo sencillo de formulario es el siguiente:

```
<form action="http://miservidor.com/acceso.jsp">
    <p>
    <label>Usuario: <INPUT type="text" name="usr"></label><br>
    <label>Clave: <INPUT type="password" name="clv"></label><br>
    <input type="submit" value="Enviar">
    </p>
 </form>
```

El formulario incluye un párrafo con tres líneas (hay dos "breaks"
) encontrando en las dos primeras dos campos de entrada de tipo texto y en la tercera una entrada de tipo botón_de_envío.



Si rellenamos los campos con los valores "alumno" y "mate001", y pulsamos "enviar", el navegador se encargará de conectar con la siguiente URL:

Como se ve, está compuesta por el valor del atributo "action" dentro de la etiqueta "FORM", el símbolo "?" y los pares nombre=valor de cada campo de entrada de datos separados por el símbolo "&".

Evidentemente, de inmediato nos damos cuenta de que "algo no puede ser así": la clave "secreta" resulta perfectamente visible. La explicación a esta observación tiene dos partes como veremos más adelante, pero antes de ello hay que decir que el método de conexión que acabamos de ver es perfectamente válido para aquellos casos en que no es preciso ocultar ninguna información, e incluso más, es adecuada cuando se pretende que la información enviada sea muy visible y pueda permitirse su alteración manualmente en la URL.

Volviendo a lo de las "dos partes": siempre podemos evitar que los pares nombre-valor sean visibles utilizando otro método de conexión para lo cual incluiremos en el elemento "form" el atributo method="post" (no poner nada es equivalente a method="get"); de este modo los datos llegan al servidor internamente dentro del mensaje de petición de la nueva página y no incluidos en la URL; pero aún así, si quisiéramos que la comunicación fuese absolutamente "secreta" (con sólo poner method="post" no evitamos que alguien con los conocimientos y medios necesarios pueda interceptar el mensaje y leerlo) será necesario utilizar encriptación. Esto pasa por utilizar el protocolo "https", cosa que está en manos de la configuración del servidor. De este modo, unos accesos absolutamente secretos se darán en caso de poder utilizar "https" y hacerlo así: <form action="https://miservidor.com/acceso.jsp" method="post">.

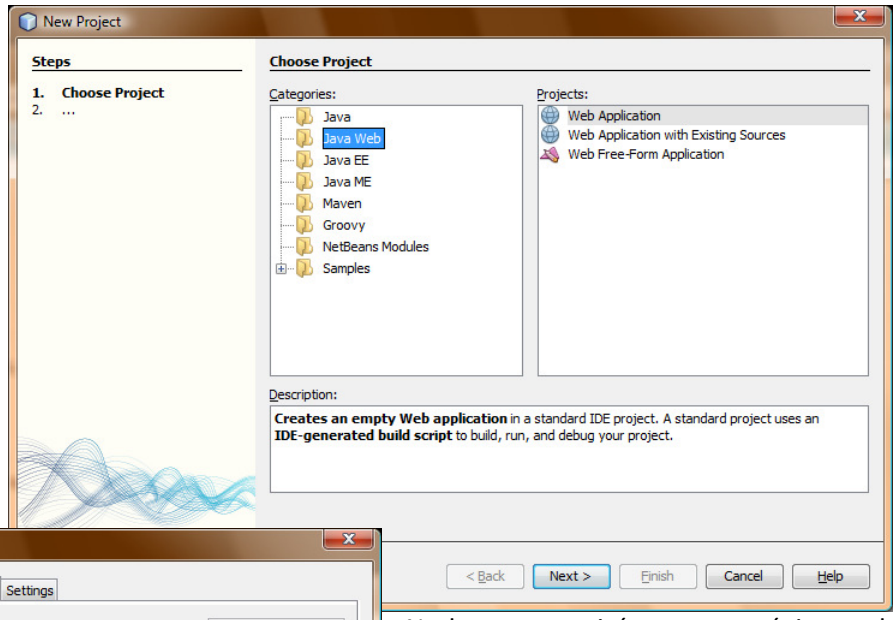
Dejando de lado estas consideraciones y volviendo a nuestro formulario, lo que hemos hecho puede ser perfectamente nuestra página inicial de acceso a la base de datos. Podremos añadirle cuanto queramos para que incluya la información adicional y el aspecto que creamos conveniente, pero la funcionalidad estrictamente necesaria ya está conseguida.

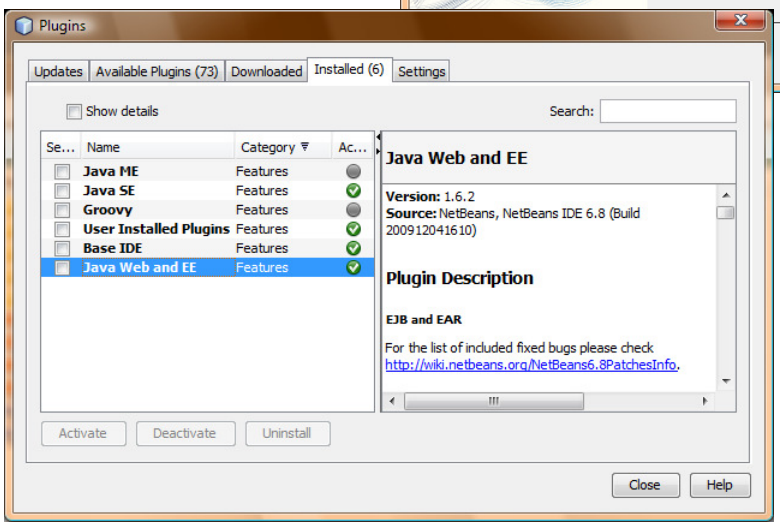
Nuestra primera página

Las variantes posibles a la hora de plantear el desarrollo que vamos a hacer son muchísimas, pero intentaremos seguir el camino más simple (no el más correcto habitualmente) de modo que lleguemos a obtener un servicio sencillo sin necesidad de excesivas complicaciones. Con ayuda de la ingente información disponible online sobre estas tecnologías, sin duda el alumno podrá explorar variantes si le resulta de utilidad.

La primera simplificación que haremos será "encargar" a la primera página web, no solo de pedir la información de usuario y clave, sino de recogerla y comprobarla. Por tanto no hablamos ya de una página en HTML sino de una JSP que además del código HTML visto incluirá algo de programación. Cuando reciba los datos y compruebe que no son correctos volverá a enviar el mismo formulario hasta que el acceso sea posible y pase a enlazar con otra página

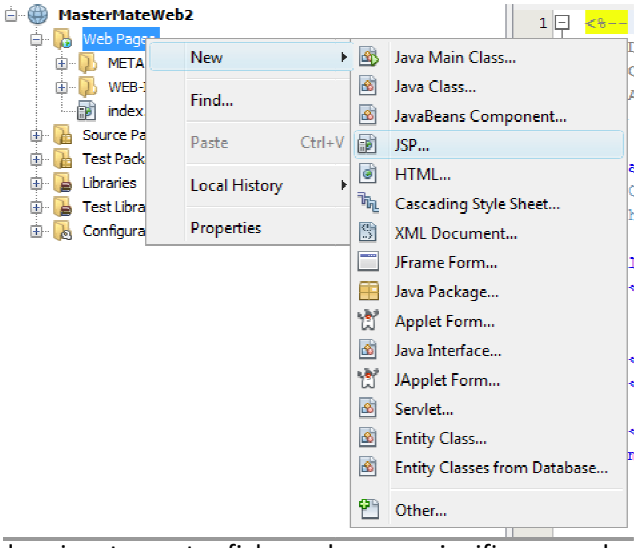
Comenzaremos generando un nuevo proyecto del tipo "Java Web" (menú "File->new Project") (figura de la derecha), lo que será posible si hemos instalado el plugin para proyectos web (menú "tools->plugins") (figura de abajo). Pondremos al nombre que queramos al proyecto y aceptaremos todo hasta finalizar.



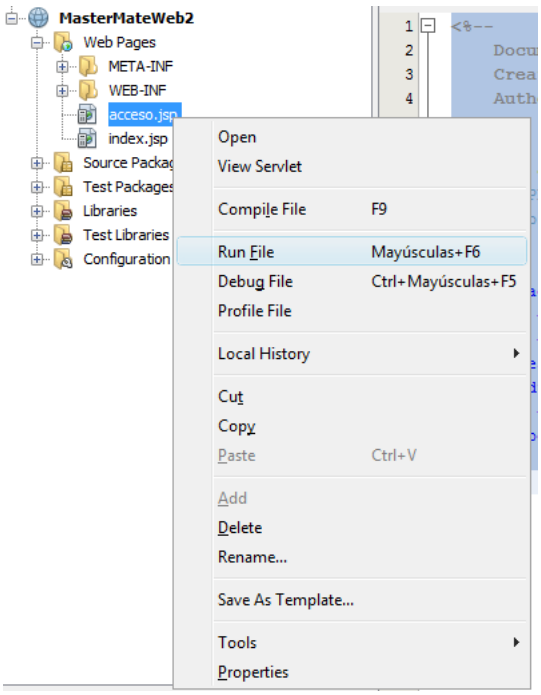


Netbeans nos sitúa en una página web que compone "de oficio" bajo el nombre "index.jsp" y que se encuentra en la carpeta del proyecto denominada "Web Pages". "index.jsp" es el fichero que el servidor enviará a un navegador cliente cuando no le especifique una página concreta. La dejaremos para más adelante y haremos nuestra nueva página "acceso.jsp" pulsando el botón derecho sobre la mencionada carpeta

"Web Pages" y llegando a seleccionar "JSP" (si no aparce bajaremos hasta "Other" y buscaremos "JSP" en el cuadro de diálogo que se abrirá) (ver figura).



La nueva página tiene un contenido inicial compuesto por un comentario JSP parentizado con "<%--" y "--%>", una directiva de página JSP "<%@page...%>" de la que hablaremos más adelante, y el restante código propio de una página HTML estática. Esta comienza con una línea que le indicará al navegador cómo interpretarla puesto que hay un par de variantes. Hablaremos de ello también más adelante. A continuación viene el contenido propio de definición de la página, con un título en la cabecera y una frase en el cuerpo. Podemos hacer la prueba de ejecutar este fichero lo que significa que lo veremos en un navegador servido por el Tomcat que controla Netbeans. Una vez que veamos la página en el navegador podemos pedirle que nos muestre el código fuente para comprobar lo que ha recibido y relacionarlo con nuestro fichero "acceso.jsp".



Lo que haremos ahora es convertir esta página en nuestra página de acceso. Para ello pondremos un título más adecuado (p.ej. "Acceso a Base de Datos") tanto en la cabecera como en el cuerpo, y añadiremos el formulario que hemos visto anteriormente usando "post" como método de envío de parámetros y poniendo como acción únicamente "#", que es una autoreferencia de la página a sí misma ().

Si salvamos esto y recargamos la página en el navegador veremos los cambios (el aspecto no es muy bueno, pero esa es otra cuestión que pasa por utilizar adecuadamente los estilos (CSS)).

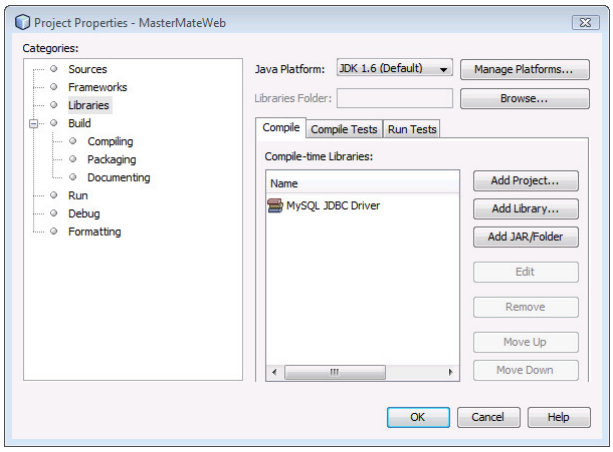
```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Acceso a Base de Datos</title>
    </head>
    <body>
        <h1>Acceso a Base de Datos</h1>
        <form action="#" method="post">
            <p>
                <label>Usuario: <INPUT type="text" name="usr"></label><br/>
                <label>Clave: <INPUT type="password" name="clv"></label><br/>
                <input type="submit" value="Enviar">
            </p>
        </form>

    </body>
</html>
```

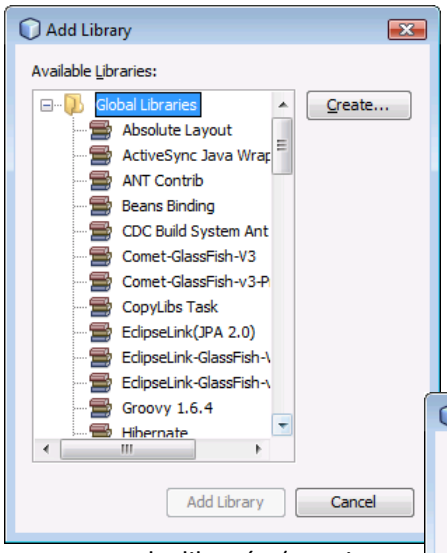
Comencemos ahora a incluir el código Java que procesará los valores de usuario y clave cuando lleguen. Lo primero será hacernos con esos valores. Para ello insertamos una zona

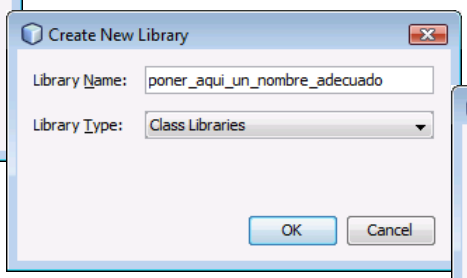
para Java parentizando con <% y %>, y en su interior requeriremos los valores de los parámetros “usr” y “clv” al objeto del lado del servidor encargado de recoger los datos de la petición (request). Situaremos este código antes del HTML porque en caso de recibir valores correctos pasaremos a otra página sin presentar el formulario.

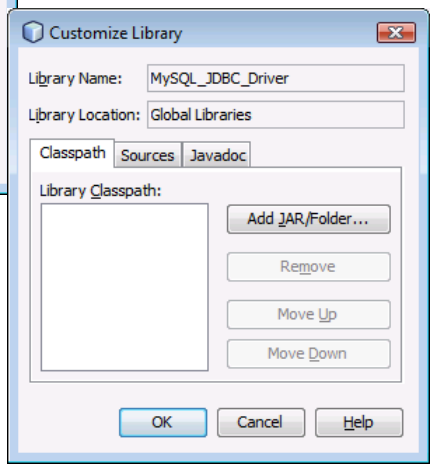
```
<%
String usuario=(String)request.getParameter("usr");
String clave =(String)request.getParameter("clv");
%>
```

En caso de no recibir los parámetros, las referencias “usuario” y/o “clave” tendrán el valor “null”, de manera que lo que haremos a continuación será intentar acceder a la base de datos sólo en caso de que ambas sean no nulas. Antes de ello, una vez que hemos llegado al momento de acceder a la base de datos, nos vamos a asegurar de disponer de la biblioteca Java que nos permite conectar con MySQL. En el icono raíz del proyecto, con el menú del botón derecho, seleccionamos “propiedades” y vamos a “libraries” (figura de la derecha). Debemos encontrarnos con “MySQL JDBC Driver”. De no ser así, pulsaremos “Add Library” y en diálogo que aparece (figura de la izquierda) la buscaremos porque posiblemente se encuentre ya entre las que Netbeans ha descargado por algún motivo. De no ser así, tendremos que descargar la librería del sitio web de MySQL y añadirla. Para ello pulsaremos “create”, que nos llevará a un nuevo diálogo que nos permite asignar un nombre a nuestro gusto para la librería (conviene mantener el original) y a continuación pasamos al otro dialogo en donde subir el “jar” y, en caso de que dispongamos de ello, los fuentes y la documentación (figuras subsiguientes).









Una vez asegurado que disponemos de la librería seguiremos adelante. En este segmento de código hay bastantes cuestiones que comentar. Comenzaremos por presentarlo e intentaremos comprenderlo a continuación:

```
12  <%
13  String usuario=(String)request.getParameter("usr");
14  String clave =(String)request.getParameter("clv");
15  if (usuario!=null && clave!=null) {
16      try {
17          Class.forName("com.mysql.jdbc.Driver").newInstance();
18          java.sql.Connection con=java.sql.DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/fabricacion",usuario, clave);
19          session.setAttribute("conexion", con);
20          %><jsp:forward page="/opciones.html"/><%
21          } catch (java.sql.SQLException ex){
22              %>se ha producido un error: <%=ex.getMessage()%><%
23          }
24      }
25  %>
```

Las dos primeras líneas (13,14) son la obtención de los parámetros que ya veíamos anteriormente, y a continuación lo que hacemos es lo que comentábamos: si tenemos el par usuario-clave, "intentar" conectar con la base de datos y avanzar hacia otra página que ya nos permita trabajar con la base de datos. Si no tenemos el par usuario-clave o algo falla en el intento, la ejecución continuará con el envío del código HTML que se encontrará a continuación.

El "intento" de conexión comienza con la carga del driver de acceso a MySql (línea 17). Esto en realidad sólo hay que realizarlo una vez y no está situado en el mejor lugar al ser esta una página a la que se accederá muchas veces, pero se ha puesto aquí por simplificar. Realmente no tiene mayor importancia puesto que sólo la primera llamada a la carga del driver tiene efecto; posteriores llamadas quedan inmediatamente sin efecto al percatarse el sistema de que el driver ya está presente. En la siguiente línea se lleva a cabo el intento de conexión a una base de datos conocida con el usuario y la clave recibidas (tanto la base de datos como el servidor podrían haberse determinado también por parámetros en caso de querer dar acceso a otras bases de datos). Esta conexión puede fallar (par usuario clave inadecuado, base de datos errónea, etc) y en tal caso se producirá una excepción que se recoge en la línea 21, de la que luego hablaremos. Si todo va bien el objeto "con" es nuestra conexión, que utilizaremos para todas las interacciones con la base de datos. Este objeto deberá ser conocido por todas las páginas web por las que transitemos a lo largo de la sesión de trabajo con la base de datos. Para permitirlo existe el objeto "session" que admite el almacenamiento de pares (nombre_de_objeto,objeto) con la denominación de "atributos"[2]. En la línea 19 se guarda este "atributo" (la conexión) para toda la sesión y que se utilizará en otras páginas. Una vez hecho esto podemos pasar a la primera página de trabajo con la base de datos: la hemos llamado "opciones.html" ya que no es otra cosa que una página estática con un menú de opciones a realizar frente a la base de datos. Esto se lleva a cabo del siguiente modo (línea 20): se sale por un momento de la zona de código Java cerrándola con %> para abrirla poco después con <%, y en ese espacio se coloca una directiva JSP que permite hacer el desvío ("jsp:forward"); cuando se llega a ejecutar esta directiva, se abandona la página presente y se ejecuta la indicada en su parámetro "page".

Nos queda por comentar qué sucede en el caso de producirse un error en el intento de conexión. En realidad no se hace otra cosa que indicarlo y permitir que siga la ejecución con el código HTML que se encuentra debajo. Para mostrar el mensaje de error se vuelve a abrir un espacio en la zona de código Java con el cierre %> y la apertura posterior con <%, y en ese espacio se sitúa un texto fijo ("se ha producido un error: ") encadenado con uno dependiente del error producido que es el "mensaje" que llega en el objeto recibido como parámetro en el catch (ex). Para mostrar como HTML el valor de una variable Java (de cualquier expresión en general) la parentizamos con <%= y %>.

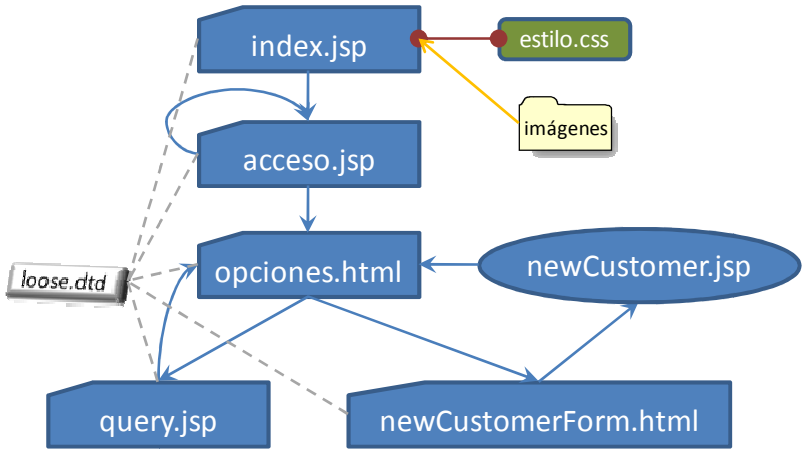
En definitiva nuestra página de acceso queda como sigue[3]:

[2] El servidor web mantiene la sesión durante un tiempo limitado (establecido en un fichero de configuración de la Aplicación Web –se localiza en "WEB-INF/web.xml"-). Este tiempo de mantenimiento se reinicia cada vez que se interacciona con el sistema. Si entre una acción y otra pasa más tiempo del establecido, el acceso al objeto "session" dará un error, circunstancia que contemplaremos para hacer que en tal caso se vuela a la página de acceso con objeto de permitir el inicio de una nueva sesión.

[3] Las dos primeas líneas no se han comentado pero hablaremos de ellas más adelante.

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<%
String usuario=(String)request.getParameter("usr");
String clave =(String)request.getParameter("clv");
if (usuario!=null && clave!=null) {
    try {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        java.sql.Connection con=java.sql.DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/fabricacion",usuario, clave);
        session.setAttribute("conexion", con);
        %><jsp:forward page="/opciones.html"/><%
        } catch (java.sql.SQLException ex){
            %>se ha producido un error: <%=ex.getMessage()%><%
        }
    }
%>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Acceso a Base de Datos</title>
    </head>
    <body>
        <h1>Acceso a Base de Datos</h1>
        <form action="#" method="post">
            <p>
                <label>Usuario: <INPUT type="text" name="usr"></label><br/>
                <label>Clave: <INPUT type="password" name="clv"></label><br/>
                <input type="submit" value="Enviar">
            </p>
        </form>

    </body>
</html>
```

Una vez puestos en marcha y visto cómo hacer una JSP vamos a plantear en qué consistirá el resto de elementos que compondrán nuestro ejemplo.



Por delante de la página “acceso.jsp” pondremos un contenido en index.jsp con el único objetivo de tener una presentación en la página que el servidor envía por defecto (podría ser “index.html” puesto que no llevará nada de Java, pero como Netbeans lo ha generado por sí mismo, no nos molestaremos en cambiarlo). Aprovecharemos esto para ver cómo utilizar una página de estilo (CSS) en conjunción con una HTML (deberíamos hacerlo así con todas las páginas, pero no lo hacemos por no alargar el ejemplo. La página de estilo sería común para todas ellas, definiendo de este modo un estilo homogéneo para toda la Aplicación Web). Además será la única página en que cuidaremos un poco el aspecto por ello contendrá algunas imágenes que se encontrarán en una subcarpeta.

Una vez que consigamos entrar, iremos a una página que nos dará cuatro opciones de actuación sobre la base de datos a modo de ejemplos. Tres de ellas serán de tipo “query” realizadas con variantes en el código HTML y que en todo caso serán ejecutadas, y presentados sus resultados, por la página “query.jsp”. Esta presentación tendrá un enlace que permitirá volver a la página de opciones. La cuarta opción será un “update” sobre la base de datos. Para ello nos llevará a un formulario que permitirá cumplimentar los datos para añadir un nuevo “cliente” a la tabla de clientes y al pulsar el botón de aceptación ejecutará otra JSP (newCustomer.jsp).

Sobre esta última JSP hay que comentar lo siguiente: en realidad sólo ejecutará código para dar de alta el nuevo usuario, y terminará redireccionando automáticamente a la página de opciones, lo que significa que no se encarga de nada relacionado con la presentación y no debería ser en realidad una “página”, es decir no debería ser una JSP, sino un programa en sentido estricto. De nuevo lo dejamos así por no complicar el ejemplo, pero tengamos en cuenta que ya se trata de algo que no pertenecería a nuestra capa de presentación. Una manera correcta de realizar esto consistiría en utilizar un “bean”, que es un tipo de objeto Java que se relaciona fácilmente con las páginas web (la especificación JSP tiene soporte específico para ello). En caso de desarrollar un sistema de acceso a base de datos real, es decir, mucho más complejo, necesitaríamos gran cantidad de “beans”, y la solución adecuada sería utilizar “EJBs” (Enterprise Java Beans) en un entorno de Servidor de Aplicaciones que facilita la tarea. Pero eso es para un curso avanzado…

Por último, observando el dibujo vemos que falta explicar el elemento “loose.dtd” ligado con líneas discontinuas a todas las páginas con presentación. Tiene que ver con la línea que Netbeans nos pone al comienzo de cada fichero: el “DOCTYPE”.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"   "http://www.w3.org/TR/html4/loose.dtd">
```

Esta es una indicación al navegador para especificarle a qué norma debe ajustarse a la hora de interpretar el código HTML. Existen diversas versiones de HTML: las mejoras que se van introduciendo con el tiempo, así como variantes en función de mantenimiento de compatibilidades. Esta norma a que debe ajustarse el navegador está definida de un modo formal por lo que se denomina “DTD” (Document Type Definition), y Netbeans está indicando por defecto el uso de “loose.dtd” de HTML4 “transicional” (observar la URL) como la especificación más razonable a priori, ya que es el estándar mejor soportado por los navegadores (version4) y que guarda compatibilidad con anteriores (Transicional). Sería bueno poder ajustarnos a normativas posteriores y más estrictas, pero cabria la posibilidad de encontrarnos con navegadores que no las trataran correctamente. Actualmente sería:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

La página de opciones

A continuación mostramos el aspecto de la página de “opciones” y su código para comentarlo posteriormente.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Acceso a Base de Datos</title>
 </head>
 <body>
  <h1>Acceso a Base de Datos</h1>
   <ul>
    <li>
     <a href="query.jsp?e=Listar%20proveedores&amp;q=SELECT%20*%20FROM%20proveedores">
      Listar proveedores
     </a> (demo de "get" directo)</li>
    <li><a href="#"
          onclick="javascript:f1.e.value='Abonados';f1.q.value='SELECT * FROM abonados  order by apellido';f1.submit();">
        Listar abonados
       </a> (demo de uso de FORM mediante Javascript)</li>
    <li>
     <a href="#"
        onclick="javascript:f1.e.value='Abonados';f1.q.value='SELECT abonados.*, ciudades.provincia FROM abonados INNER JOIN ciudades
      Listado de usuarios en ciudades conocidas con provincia
     </a>(una orden más compleja)</li>                                  ON abonados.ciudad = ciudades.ciudad order by apellido';f1.submit();">
    <li>
     <a href="newCustomerForm.html">
      Nuevo usuario
     </a>(demo de update -las anteriores son querys-)
    </li>
   </ul>

   <form name="f1" action="query.jsp" method="post" style="display:none">
           <input type="text" name="q">
           <input type="text" name="e">
       </form>
    </body>
</html>
```

Nuestra página de acceso contiene una lista sin numerar (unordered list) y un formulario oculto (observar el atributo style="display:none"). La lista tiene las siguientes opciones:

- La primera no es otra cosa que un enlace a la página que procesa queries (query.jsp) con los parámetros necesarios incluidos en la URL (un "get").Los dos parámetros son "e" (de "encabezamiento" que permitirá poner un titulo en la página de respuesta), y "q" (la "query" en SQL) Una particularidad que puede observarse es que, por dificultades de codificación, las URLs no pueden incluir caracteres blancos –entre otros- y por tanto se indican con su código hexadecimal correspondiente (%20).
- Las dos opciones siguientes se diferencian únicamente en la complejidad de la sentencia SQL, pero en lo que vemos ahora son idénticas. Ambas utilizan el método "post" de envío de parámetros, para lo que es necesario disponer de un formulario oculto que se rellenar automáticamente y se envía. Utilizamos un hiperenlace ("anchor" <a>) para que el navegador lo muestre normalmente, pero el recurso referenciado está anulado[4] y se ejecuta código Javascript en caso de que se pulse en el enlace (atributo "onclick"). Este código ejecuta tres acciones: en el elemento llamado "f1" (el formulario –ver el atributo name="f1"-) cumplimenta el atributo "valor" del campo de texto de entrada llamando "e"; hace lo mismo con la query en SQL para el campo "q"[5]; y termina llamando al método de envío ("submit()") sobre el formulario. De este modo llegan a la página especificada en el formulario (action="query.jp") los parámetros según el método especificado (method="post").

[4] En realidad es una URL válida ya que "#" es el prefijo para etiquetas asociadas a puntos internos de documentos. Si no lleva nada por delante se refiere una etiqueta del mismo documento mostrado, y al no llevar etiqueta asociada, se refiere al documento en sí mismo.

[5] Obsérvese que ahora no es necesario sustituir blancos por %20, ya que estamos cumplimentando normalmente un formulario, y será este el que se encargue de codificar datos adecuadamente cuando se proceda al envío.

- La última opción no es una “query”, sino que se pretende hacer un “update”, de modo que el enlace es un enlace normal a una página web que contendrá el formulario a rellenar para aportar los nuevos datos.

La página de QUERY y presentación de resultados

Esta página recoge los datos que le llegan de la de acceso, consulta a la base de datos, recoge los resultados y los presenta mediante código HTML (puede verse una imagen resultado en la página siguiente).

Antes de nada comentaremos las dos líneas iniciales.

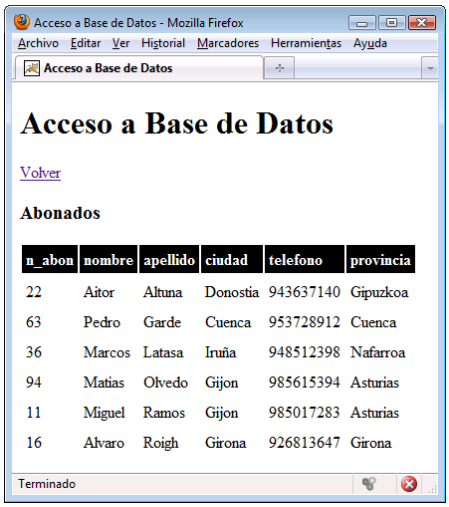
Ambas son “directivas de página” JSP (<%@page ...%>) la primera indica que el resultado de la ejecución será un texto en HTML (el servidor se lo comunica al navegador antes del envío para diferenciarlo de otros contenidos –imágenes, vídeos, etc-), y que además se usará la codificación de texto UNICODE en formato UTF-8[6].

La segunda línea indica los “imports” que serán necesarios para escribir nuestro código Java si queremos evitar escribir los nombres completos de las clases (p.ej. “ResultSet” por “java.sql.Resulset”). Cuando el servidor convierta la página JSP en un fuente Java que luego compilará y ejecutará, añadirá las líneas de “include” correspondientes a los paquetes que aquí le indicamos.

```
1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <%@page import="java.sql.*, java.io.*"%>
3  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4
5  <%
6  Connection con;
7     if (null==(con=(Connection)session.getAttribute("conexion"))){
8      %><jsp:forward page="acceso.jsp"/><%
9      }
10 %>
11
12 <html>
13  <head>
14   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
15   <title>Acceso a Base de Datos</title>
16  </head>
17  <body>
18   <h1>Acceso a Base de Datos</h1>
19   <p><a href="opciones.html">Volver</a></p>
20   <h3><%=request.getParameter("e")%></h3>
21   <%
22    try { // "try" de confeccion de una respuesta ----------------------------------------
23     ResultSet rs = con.createStatement().executeQuery(request.getParameter("q"));
24     ResultSetMetaData rsmd = rs.getMetaData();
25   %>
26   <table border="0" cellpadding="4">
27    <thead style="color:white;background-color: black; font-weight:bolder;"><tr>
28    <%
29     int nCol = rsmd.getColumnCount();
30     for (int col = 0; col < nCol; col++) {
31      out.println("<td>" + rsmd.getColumnLabel(col + 1) + "</td>");
32      } // fin del "for" para todas las columnas de la cabecera
33   %>
34    </tr></thead>
35    <%
36     rs.last();
37     int nRow = rs.getRow();
38     for (int row = 0; row < nRow; row++) {
39      rs.absolute(row + 1);
40      out.print("<tr>");
41      for (int col = 0; col < nCol; col++)
42       out.print("<td>" + rs.getString(col + 1) + "</td>");
43      out.println("</tr>");
44      } // fin del "for" para todas las filas
45   %>
46   </table>
47   <%
48     } catch (Exception ex) { // "catch" de error en respuesta (fin del "try") ------------   ----
49     out.println(ex.getMessage());
50   %>
51   <pre style="border:thick solid">
52   <% ex.printStackTrace(new PrintWriter(out)); %>
53   </pre>
54   <% } // fin del boque "catch" ------------------------------------------------------------  -----
55   %>
56 </body>
57 </html>
```

[6] UNICODE es una codificación de caracteres que da soporte a todos los idiomas del mundo y otros conjuntos de símbolos de uso toda clase de codificación de documentos. En este sentido es la solución a las limitaciones e incompatibilidades de las diversas codificaciones ASCII, si bien estas no han desaparecido y es preciso indicar en cada caso qué se usa. Puede verse esta diversidad de codificaciones en el menú de un navegador seleccionando en el menú “ver→ codificación”

Vayamos ahora al contenido. Comenzamos por recuperar el objeto de clase “Connection” que guardamos en la sesión al acceder a la aplicación web. Si ha pasado un tiempo excesivo la sesión habrá caducado y el resultado del intento de recuperar la conexión será “null”, de modo que en tal caso redireccionamos directamente a la página de acceso para permitir iniciar una nueva sesión. Si hemos recuperado el objeto seguimos adelante comenzando a construir la página HTML de respuesta. Llegados al punto donde pondremos el encabezamiento, insertamos el resultado de pedir el parámetro “e” al objeto “request” (<%=request.getParameter("e")%>). Una vez hecho esto estamos en el punto de “montar la matriz respuesta. En una zona de código (<% … %>) hacemos la consulta a la base de datos como la hacemos normalmente en una aplicación independientemente de que sea web o no, solo que aquí la query la obtendremos del parámetro “q” que nos proporciona el objeto “request”. De este modo nos haremos con el “ResulSet” y posteriormente con el “ResulSetMetaData”. Estas operaciones pueden fallar (p.ej. porque no nos haya llegado el parámetro “q” –ver figura, donde se ha escrito una URL que provoca el error), de modo que se incluirán dentro de un bloque “try-catch”. En la parte “try” se presenta la matriz resultado, y en la “catch” se muestra el mensaje de error. Vayamos a esto último en primer lugar: al recoger la excepción “ex” se muestra su mensaje (“ex.getMessage()”) haciendo un “println(.)” sobre otro de los objetos disponibles en las JSP, el “out”. El objeto “out” es un Writer (en concreto es un “javax.servlet.jsp.JspWriter” –la biblioteca “java.servlet” la carga el servidor-). A continuación se pone el elemento “<pre></pre>” de HTML para definir una zona en donde la impresión de texto se realiza “preformateada”, es decir, no se altera (p.ej. no se eliminan espacios múltiples, ni pasos de línea,…). Se indica además un formato (estilo) con un grueso recuadro. Dentro de este espacio se muestra la traza de error (“ex.printStackTrace(.)”), pero en lugar de usar el método que imprime por la salida estándar (el que no lleva parámetro) usamos uno que toma como parámetro un “PrintWriter”, y para proporcionarle nuestro objeto “out”, que es un “Writer” (“JspWriter”), y que se asocia a la página

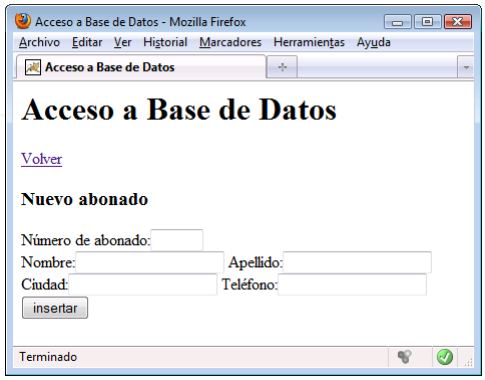


web, lo “envolvemos” adecuadamente (“new PrintWriter(out)”).

La parte de “camino de éxito” del “try-catch” es al que construye la matriz de respuesta en la página. Para ello comienza con el elemento tabla (<table>…</table>) y a continuación con la estructura de la fila de cabecera de la misma (<thead><tr>…</tr></thead>). A esta cabecera le define un estilo –fondo negro y letra blanca y gruesa-. La construcción de las celdas que componen la cabecera se hace mediante código Java: toma el número de columnas de los metadatos, y mediante un ciclo “for” escribe, a través del objeto “out” las cabeceras de cada columna (“rsmd.getColumnLabel(col + 1)”) dentro del elemento celda (<td>…</td>). Una vez cerrada la cabecera, otro segmento de código Java recorre con un doble ciclo “for” las filas y columnas de datos construyendo cada fila (<tr>…</tr>) y cada celda (<td>…</td>) de un modo similar a lo visto para la cabecera.

La página de formulario para nuevo usuario

Esta página es la más simple de todas. Se limita un formulario con los campos necesarios para dar de alta a un nuevo usuario, que conecta para su procesamiento con la página “newCustomer.jsp”.




```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Acceso a Base de Datos</title>
    </head>
    <body>
        <h1>Acceso a Base de Datos</h1>
        <p><a href="opciones.html">Volver</a></p>

        <h3>Nuevo abonado</h3>
        <form action="newCustomer.jsp" method="POST">
            Número de abonado:<input type="text" name="n_a" value="" size="4" /><br/>
            Nombre:<input type="text" name="nom" value="" size="20" />
            Apellido:<input type="text" name="ape" value="" size="20" /><br/>
            Ciudad:<input type="text" name="ciu" value="" size="20" />
            Teléfono:<input type="text" name="tel" value="" size="20" /><br/>
            <input type="submit" value="insertar" />
        </form>

        </body>
</html>
```

La (no) página de UPDATE

La página que realiza el “update” de la base de datos es, como ya se ha indicado, una JSP “falsa”, ya que no realiza ninguna presentación. Se mantiene así para no complicar el ejemplo, e incluso con esta misma intención no se ha desarrollado del todo correctamente. Como se ve, al igual que en la página “query.jsp” se intenta recuperar la conexión y si no es posible se redirecciona a la página de

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.sql.*, java.io.*"%>

<%
 java.sql.Connection con;
 if (null == (con = (java.sql.Connection) session.getAttribute("conexion"))) {
%><jsp:forward page="acceso.jsp"/><%
 }

 String n_a = request.getParameter("n_a");
 String nom = request.getParameter("nom");
 String ape = request.getParameter("ape");
 String ciu = request.getParameter("ciu");
 String tel = request.getParameter("tel");

 // OJO código simplificado. Leer detenidamente comentario en el texto
 if (n_a != null && nom != null && ape != null && ciu != null && tel != null)
  con.createStatement().executeUpdate(
    "INSERT INTO abonados (n_abon , nombre , apellido , ciudad, telefono) VALUES(" +
    n_a + ",\"" + nom + "\",\"" + ape + "\",\"" + ciu + "\"," + tel + ");"
    );

%>
<jsp:forward page="/opciones.html"/>
```

acceso. Si se obtiene la conexión, se recogen los datos de sus parámetros, y se ejecuta el "update" construyendo la "String" correspondiente en lenguaje SQL. Una vez hecho esto se vuelve incondicionalmente a la página de opciones.

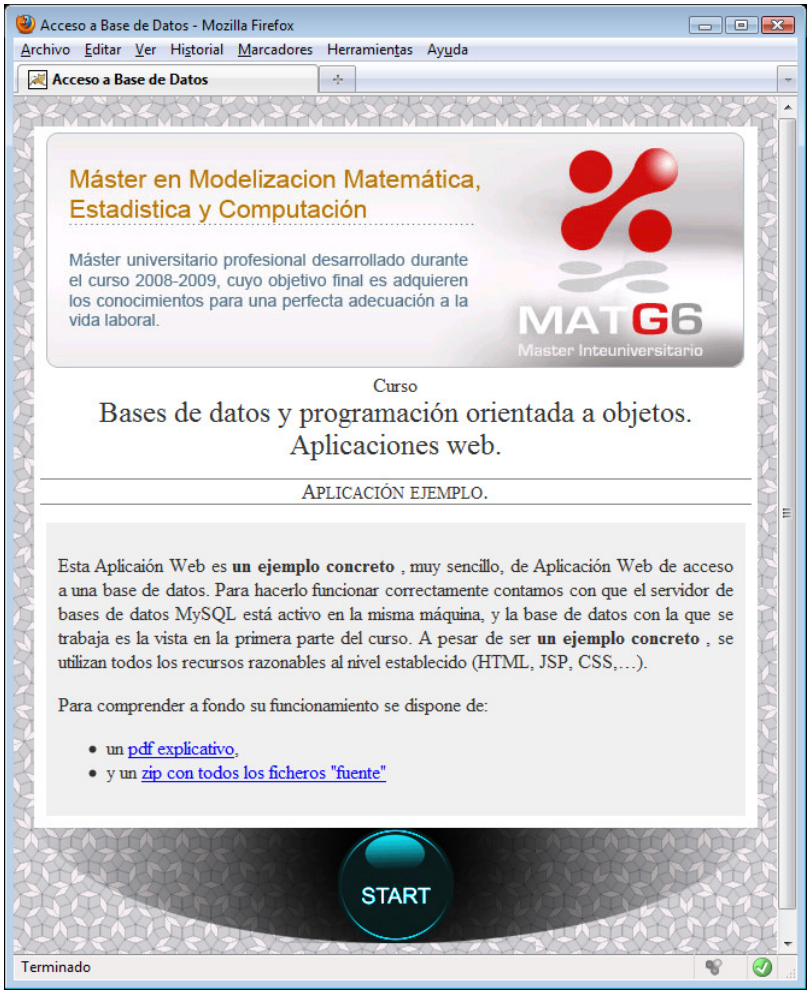
Obsérvese que el "update" se ejecuta sólo en el caso de disponer de todos los datos necesarios, en caso de que falte al menos uno, no se realiza ninguna acción, pero se redirecciona igualmente a la página de opciones, con lo que el usuario no tiene conocimiento de que la actualización no se ha realizado. Es preciso corregir esto para que la aplicación sea correcta (programar una parte "else" para el "if" y, razonablemente volver al formulario manteniendo los campos ya cumplimentados e indicando el error, para lo cual hemos de retocar la página de formulario haciendo que permita llegar a ella con datos predefinidos y el mensaje).

Igualmente faltaría atender a posibles errores en la ejecución del "update", por lo que lo adecuado será programar este en una sección "try" y escribir una "catch" que permita igualmente que en el caso anterior, volver al formulario si procede o actuar en consecuencia en función del tipo de error.

La página Index.

Ya hemos visto todo lo relacionado con lo que es propiamente la aplicación web de acceso a la base de datos. Como ya se ha indicado, lo único que nos queda es estudiar algunas cuestiones menores relativas al aspecto de las páginas web, cosa que haremos situando un poco de contenido en la página "index.jsp" que aparecerá en el navegador si ponemos en la URL sólo la dirección de la carpeta que contiene la aplicación (http://localhost:8084/MasterMateWeb2 en el ejemplo desarrollado en este ejemplo).

Comenzamos por ver el aspecto que presenta la página en la figura relacionarlo con el código.



Comenzaremos por el fondo de la página que se presenta cubierto por una teselación de Penrose[7]. En realidad se trata de una imagen parcial que se repite para rellenar todo el espacio y que no está directamente referenciada en la JSP, sino que forma parte de la hoja de estilo CSS asociada. Esta asociación viene dada por el elemento "link" de la línea 7. Éste indica una relación con otro fichero de tipo hoja de estilo ("stylesheet") indicando su identificador ("estilo.css") su tipo ("text/css") y que es el aplicable a toda clase de medios (podríamos definir distintos estilos para presentación en pantalla, proyección, impresión, etc).

```
1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
3
4  <html>
5   <head>
6    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7    <link rel="StyleSheet" href="estilo.css" type="text/css" media="all">
8   <title>Acceso a Base de Datos</title>
9   </head>
10  <body>
11   <div class="contenido">
12    <img src="imagenes/5ixz46ch.png" alt="Banner del Master en Modelización Matemática, Estadística y Computación">
13    Curso
14    <h1>Bases de datos y programación orientada a objetos. Aplicaciones web. </h1>
15    <h2>Aplicación ejemplo.</h2>
16    <div class="intro">
17     <p>
18      Esta Aplicaión Web es <strong> un ejemplo concreto </strong>, muy sencillo, de Aplicación Web de acceso a una
19      base de datos. Para hacerlo funcionar correctamente contamos con que el servidor de bases de datos MySQL está
20      activo en la misma máquina, y la base de datos con la que se trabaja es la vista en la primera parte del curso.
21      A pesar de ser <strong> un ejemplo concreto </strong>, se utilizan todos los recursos razonables al nivel
22      establecido (HTML, JSP, CSS,...).
23     </p>
24     <p> Para comprender a fondo su funcionamiento se dispone de:</p>
25     <ul style="text-align:left;">
26      <li>un <a href="accesoWebDB.pdf">pdf explicativo</a>,</li>
27      <li>y un <a href="accesoWebDB.zip">zip con todos los ficheros "fuente"</a></li>
28     </ul>
29    </div>
30   </div>
31   <img class="acceso"  name="startImage" src="imagenes/start_off.png" alt="" usemap="#boton">
32   <map name="boton">
33   <area shape="circle" coords="300,48,48" href="acceso.jsp" alt="boton de comienzo"
34    onmouseover="javascript:startImage.src='imagenes/start_on.png'"
35    onmouseout="javascript:startImage.src='imagenes/start_off.png'"
36    >
37   </map>
38  </body>
39 </html>
```

La hoja de estilo no solo especifica la imagen que tesela el fondo sino, en general, todos los aspectos que no son propiamente contenido (es decir, los que son "aspecto" de la presentación). Iremos viéndola a la par.

La razón de que se muestre el fondo es que, asociado a la etiqueta "body", en la hoja de estilo se especifica que se usará una "background-image" dando su dirección (url) que es "penrose_tiles.gif" en la carpeta "imágenes"[8]. Vemos también en el fichero de estilo que al cuerpo de la página le afectan otras tres

[7] En realidad se trata de un cubrimiento de la página con repeticiones de una imagen cuadrada que es un recorte sobre una teselación de Penrose. Se ha elegido esta teselación por ser matemáticamente muy interesante, si bien su particularidad especial es todo lo contrario de lo que se necesita en una página web: es completamente aperiódica. Si se observa con detenimiento se verán las líneas de unión de las imágenes. Las teselas típicas para la web presentan continuidad entre lados opuestos (por decirlo de un modo más matemático, son desarrollos en el plano de superficies toroidales)

[8] Para llevar imágenes a una carpeta de un proyecto Netbeans podemos crear la carpeta (botón derecho en "Web Pages" y seleccionar "new" y "folder" -u "Other" si no aparece "folder" y de nuevo "Other" en el cuadro de dialogo para llegar a "folder"-). Una vez creada la carpeta podemos arrastrar las imágenes directamente.

especificaciones: el ancho será de 600 pixels, los márgenes superior e inferior serán de 25 pixels y los laterales automáticos (el navegador reparte por igual y el contenido queda centrado), y por último se indica que el contenido debe mostrarse centrado.

```
body {
    background-image: url("imagenes/penrose_tiles.gif");
    width:600px;
    margin:25px auto;
    text-align:center;
}

div.contenido {
    color: rgb(50,50,50);
    background: white;
    padding: 5px;
}

h1 {
    font: x-large/110% "New Century Schoolbook", serif;
}

h2 {
    font: large/110% "New Century Schoolbook", serif;
    font-variant:small-caps;
    border-top:thin solid gray;
    border-bottom:thin solid gray;
}

div.intro {
    background:rgb(235,235,235);
    text-align:justify;
    padding:10px;
}

img.acceso {
    border:0px;
}
```

Una vez dentro del cuerpo de la página tenemos una zona (división) con el atributo "contenido" que hace que le sean aplicadas nuevas especificaciones de estilo: el texto se mostrará de un color gris bastante oscuro (50 en una escala de 0 a 255 para cada componente de color –red,green,blue-); el fondo en blanco; el contenido se mostrará separado de los bordes en 5 pixeles.

Esta zona comienza con el "banner" del Master en Modelización Matemática, Estadística y Computación (línea 12)[9]. A continuación hay unas líneas de texto dos de las cuales se indican como títulos (Header) de niveles 1 y 2, y estos están afectados de estilos específicos por nuestra hoja de estilo: en ambos se especifica el tipo de letra con dos tamaños distintos y, en el caso de "h2" se indica que el texto irá en "mayúsculas pequeñas" (small-caps) y que sus bordes superior e inferior serán visibles como una fina línea continua de color gris.

Después de esto tenemos otra zona anidada con un nuevo estilo asociado ("intro"). El contenido es el párrafo explicativo de la intención del ejercicio y los enlaces a documentos. El estilo ya puede entenderse con lo visto anteriormente… un fondo gris bastante claro, con el contenido apartado 10 pixels de los bordes y el texto alineado en ambos laterales. El contenido consiste en un párrafo ( ), que incluye un par de segmentos en negrita (), y una lista sin numerar (unordered list) cuyos dos elementos (
) contienen hipervínculos (anchor <a>).

Con esto salimos ya de las dos zonas anidadas afectadas de estilos específicos y nos encontramos con la imagen de acceso a la aplicación web. Aprovechamos este elemento para ver un par de cuestiones puntuales de HTML. En primer lugar vemos que se trata de un par de elementos relacionados: una imagen que usa un "mapa" (usemap="#boton") y el mapa (<map></map>). Esto nos permite utilizar una imagen como base de hipervínculos diversos, siendo el mapa el que define áreas sensibles (<area>) que nos pueden llevar a distintos documentos. En segundo lugar vemos que en HTML podemos tener en cuenta situaciones concretas denominadas "eventos", como es el caso de que el ratón se sitúe sobre un elemento ("onmouseover") o salga de él ("onmouseout") y asociar alguna acción.

[9] Obsérvese que el elemento "img", además de la referencia a la imagen a mostrar (atributo"src") lleva otro atributo con un texto alternativo ("alt") descriptor de la imagen. Esto es necesario si se quiere cumplir con las normas de accesibilidad WAI (Web Accessibility Initiative) que permiten que la web sea "usable" por personas con discapacidades. En el caso que estamos comentando, un sintetizador de voz permitiría a un ciego conocer el contenido de nuestra página, incluyendo la presencia del "banner" (en realidad este banner contiene un párrafo que le estaríamos ocultando… podríamos incluirlo en el texto de "alt")

Hemos aprovechado la existencia de mapas para hacer que no toda nuestra imagen sea un hipervínculo, sino sólo la zona que representa un botón (con un área circular situada adecuadamente). La imagen lleva asociado un estilo que le fuerza a no tener borde, porque los navegadores suelen mostrar el borde a toda imagen asociada a mapas o dentro de un hipervínculo (<a>).

En cuanto a los eventos, en este caso lo que hacemos es cambiar la imagen presentada accediendo al atributo “src” de nuestro elemento imagen, que hemos llamado “start” (name=”start”). Disponemos de dos imágenes que sólo varían ligeramente en el color de modo que el efecto de entrar o salir con el ratón de la zona del botón da una sensación de “reacción”.