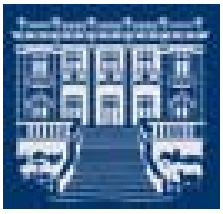


UNIVERSIDAD DE CHILE
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Industrial
IN72J – Arquitectura Tecnológica de un E-Business
Prof.: Juan Velásquez – Sebastián Ríos – Pablo Román
Aux.: René Díaz M. – Ángel Jiménez

# TUTORIAL
# IMPLEMENTACIÓN DE APLICACIONES WEB USANDO WSAD

**René Díaz Montenegro**
**rediaz@manquehue.net**
**Fecha : Julio de 2004**

# 1 ÍNDICE

# 2 INTRODUCCIÓN

El siguiente tutorial tiene como finalidad básica el asistir a los alumnos del curso IN72J – Arquitectura Tecnológica de un E-Business en el desarrollo de una aplicación web, a partir de un diseño orientado a objetos, empleando tecnología Java y utilizando las herramientas IBM WebSphere Studio Application Developer 4.03, IBM DB2 Server Edition 8.1 y Sybase Power Designer 9.5.1. De ninguna manera se pretende aquí dar una completa visión de las potencialidades de estas herramientas ya que ello sin duda podría ser objeto de un libro completo.

El desarrollo del tutorial supone que contamos con un diseño orientado a objetos. A través de tres ejemplos, ascendentes en complejidad, se presenta la interacción entre páginas jsp, servlet – jsp y servlet – jsp – bases de datos. Además se incluye consideraciones respecto de cómo diseñar una base de datos usando Power Designer, generar los scripts de construcción de la misma, implementarla y poblarla por medio de DB2.

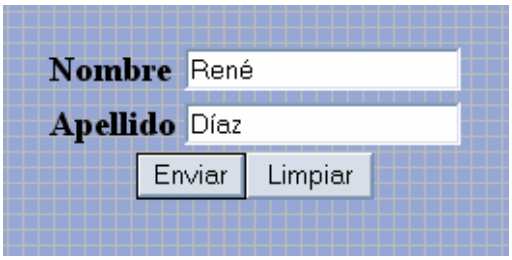
Por último, se asume que los lectores tienen conocimientos respecto de bases de datos, diseño orientado a objetos, HTML, java y arquitectura cliente servidor de tres capas. Además este tutorial es complementado por el tutorial de WebSphere Studio Application Developer entregado oportunamente. Su lectura es necesaria para la comprensión de este documento.

René Díaz Montenegro

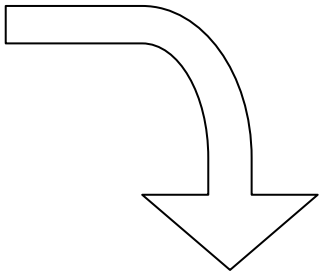
# 3 EJEMPLO 1

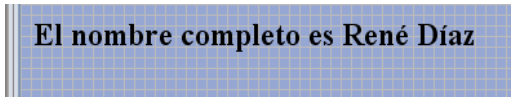
## 3.1 Planteamiento del problema

Deseamos implementar un concatenador de palabras, particularmente queremos que el cliente nos entregue su nombre y apellido en dos campos separados para luego concatenar ambos string retornando uno solo con el nombre completo. EL siguiente esquema muestra lo que buscamos:




inicio.html






respuesta.jsp

La idea es entonces seguir los siguientes pasos:

1. Construir un archivo HTML de nombre inicio.html que contenga un formulario con dos campos de entrada, nombre y apellido y que postee los datos a la página respuesta.jsp
2. Construir una página jsp, de nombre respuesta.jsp que se encargue de decepcionar los datos, construir el string con el nombre completo y luego presentarlo de manera gráficamente adecuada, esto es, con un fondo a elección, fuente de letras adecuada, y anteponiendo al nombre completo la frase “El nombre completo es:”.

## 3.2 Consideraciones teóricas

Lo que haremos es una implementación sin utilizar objetos. Solamente emplearemos un documento de recepción de datos y otro que contiene ambas, la lógica y la presentación gráfica de la información.

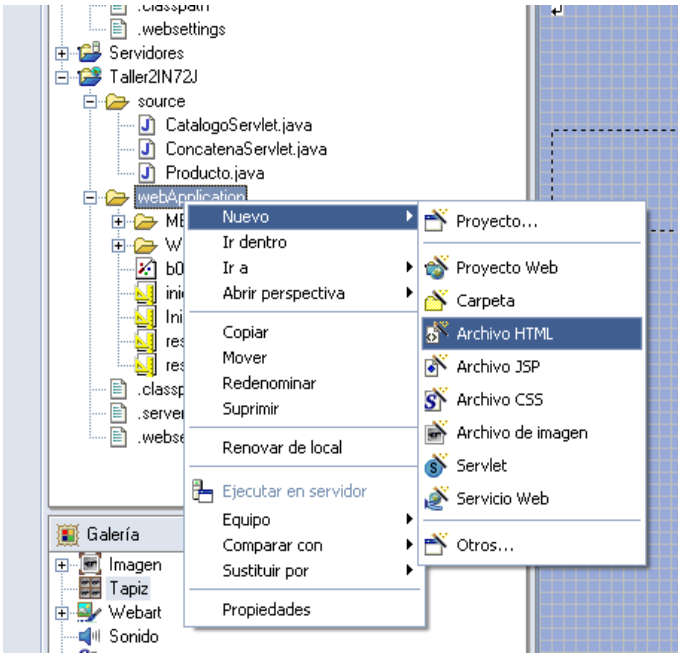
Aquí es importante aclarar lo siguiente:

- El documento inicio.html, es solicitado por el cliente ingresando la URL, por ejemplo, www.ejemplo.cl/inicio.html. Esta petición es recibida por el web server quien se encarga de enviar una copia de inicio.html la cual, una vez en el computador del cliente, es interpretada por el browser quien despliega la página en pantalla de acuerdo a lo especificado por medio del código html.
- El cliente, al llenar el formulario, no tiene interacción alguna con el web server. Si se estuviese usando sesiones, tampoco la hay ya que el servidor revisará si la sesión se encuentra activa cuando se realice una nueva petición por parte del cliente.
- Cuando el cliente presiona el botón enviar del formulario, de no haber algún script que valide los datos (javascript, VBScript), la información contenida en este es enviada al servidor quien invocará el documento señalado en el campo *action* del tag <*FORM*>. En nuestro ejemplo se ejecutará el archivo respuesta.jsp. Una página jsp tiene la capacidad de **generar código html**, en otras palabras, una página jsp es un programa escrito en java con incrustaciones de HTML (también puede verse como código HTML con incrustaciones de java), luego tiene la capacidad de manejar interfaz gráfica y lógica del negocio en un mismo programa o archivo. Cuando un a página jsp es invocada o ejecutada **por el servidor,** el resultado de ello será un documento HTML puro, sin código java. Este documento será enviado **por el servidor al cliente** y el browser, como sabe interpretar html, lo desplegará en pantalla. Nótese que al cliente no jamás le llega documento alguno que posea un programa o parte de este escrito en java.
- Los "trozos" de código java de una página jsp deben ir encerrados entre <% al inicio y %> al final Del código java incluido en el documento jsp, se desplegará en pantalla aquel que tenga la siguiente sintaxis <*%=nombre_variable%*>. El código anterior desplegará, en la posición del documento en que esté, el valor que posee la variable *nombre_variable* respecto del programa java. Cualquier otro "tipo" de código java no

será desplegado en pantalla, si no que será compilado y ejecutado como un programa tradicional.

- Por último, el ejemplo que construiremos puede ser implementado de manera análoga por medio de php o asp, las consideraciones teóricas planteadas son válidas para esos lenguajes.

## 3.3 Solución




En primer lugar construiremos el documento inicio.html. Para ello, en WSAD[1] vamos a la vista web, seleccionamos la carpeta *webApplication* del proyecto sobre el cual estamos trabajando. Luego hacemos clic con el botón derecho del mouse y vamos al menú *Nuevo,* y seleccionamos *Archivo Html*.

Se desplegará un wizard en el cual solo debemos ingresar el nombre que queremos dar al documento, en este caso inicio.html. Luego haciendo clic en finalizar el programa se encargará de construir el archivo.

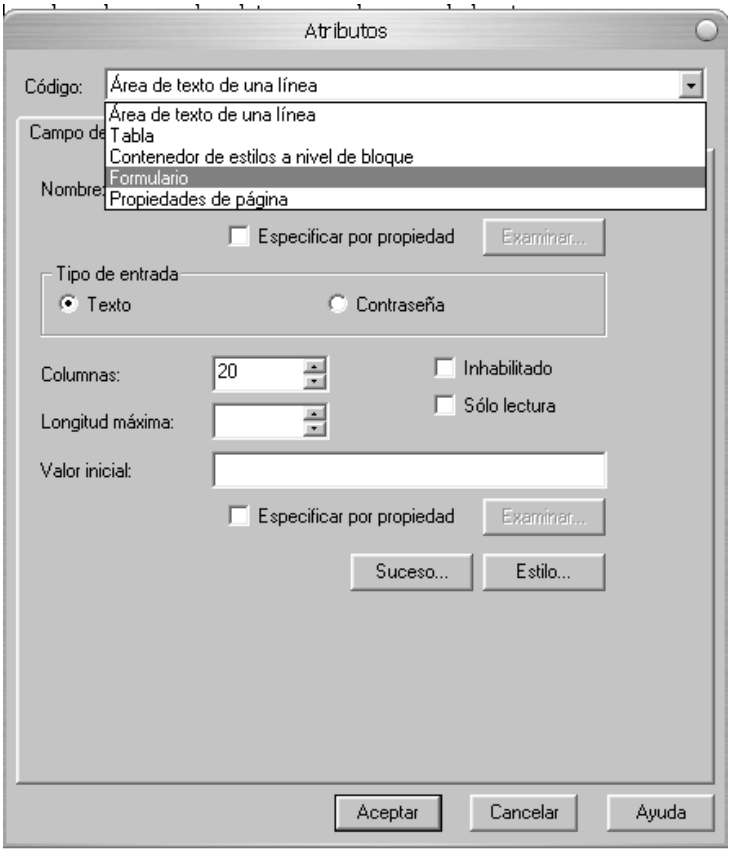
A la derecha del navegador aparecerá el documento recién creado. En la parte inferior de este hay tres tags:

- Diseño: Permite construir el documento html usando herramientas de asistencia que evitan tener escribir código html.
- Código fuente: muestra el código html del documento. Aquí se puede editar el archivo directamente a través de éste lenguaje.

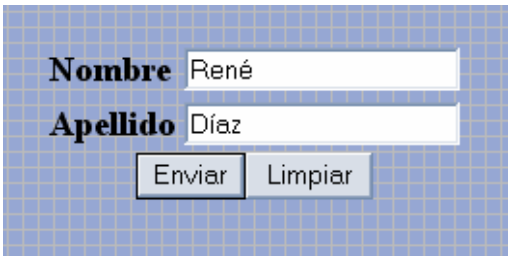
[1] WebSphere Studio Application Developer. Suponemos que Ud. leyó atentamente el tutorial sobre la aplicación, por lo que sabe como construir proyectos.

- Vista Previa: como su nombre lo indica permite ver una vista previa del documento.

Ahora bien, en la vista diseño usando los menús, particularmente el menú *insertar* y luego dentro de aquel en *Formulario y campos de entrada* encontraremos todas las opciones que nos permiten insertar un formulario, dentro de el insertar dos campos de texto y los botones para enviar y limpiar el formulario. Haciendo clic en un campo de texto se desplegará el wizard que aparece a continuación. En el debemos completar el campo *nombre* mediante el cual establecemos el nombre con que llamaremos a ese campo y por ende con el cual recibiremos los datos enviados por el cliente.




Ahora bien, en este mismo wizard, podemos seleccionar otros elementos para ser editados. Por ejemplo podemos seleccionar el elemento *formulario* en el cual debemos indicar la página o archivo al cual serán posteados los datos. Para ello completamos el campo *Acción*. En nuestro caso colocamos *respuesta.jsp*. En este wizard también podemos seleccionar el método de envío (GET o POST), agregar parámetros para enviar a través de la URL o bien incluir parámetros de tipo hidden que acompañen al formulario.




Una vez que tengamos construido el documento en la vista diseño debiésemos tener algo similar a lo que se muestra en la figura de la derecha. Ahora bien, el código fuente de este archivo html es el

siguiente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>inicio.html</TITLE>
</HEAD>
<BODY background="b002bcg.gif">
<P><FONT size="+2"><B>Complete el Formulario</B><BR>
<BR>
</FONT></P>
<P align="center"><BR>
</P>
<FORM method="POST" action="respuesta.jsp"> Creamos el formulario
<CENTER>
<TABLE border="0">
    <TBODY>
        <TR>
            <TD><B><FONT size="+1">Nombre</FONT></B></TD>
            Campo de entrada para el nombre
            <TD><INPUT size="20" type="text" name="nombre"></TD>
        </TR>
        <TR>
            <TD><B><FONT size="+1">Apellido</FONT></B></TD>
            Campo de entrada para el apellido
            <TD><INPUT size="20" type="text" name="apellido"></TD>
        </TR>
    </TBODY>
</TABLE>
<INPUT type="submit" name="Enviar" value="Enviar"><INPUT type="reset"
name="Limpiar" value="Limpiar"></CENTER>
</FORM>
</BODY>
</HTML>
```

***Resumen:*** **hasta aquí lo que hemos hecho es crear un documento html que contiene un formulario con dos campos de entrada, *nombre* y *apellido*, los cuales son enviados a la página *respuesta.jsp*.**

Corresponde ahora crear el documento respuesta.jsp. Para ello desde la vista web, seguiremos la misma secuencia de pasos que para crear un documento html, pero en el menú *Nuevo* seleccionamos *Archivo JSP*. Se desplegará una interfaz similar a la de un archivo html, con la diferencia que aparecerán algunos menús propios de un documento jsp los cuales explicaremos más adelante en la medida que se requiera.

Como señalamos una página jsp es código html con incrustaciones de código java. El documento respuesta.jsp debe realizar las siguientes operaciones en el orden que se indica:

1. Recibir los datos del formulario de la página inicio.html.
2. Concatenar en un solo string el campo *nombre* y *apellido* de manera de desplegar el nombre completo.
3. Desplegar en pantalla el nombre completo, es decir el string con las variables concatenadas.

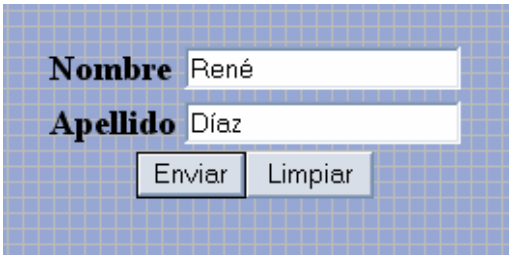
Para efectos de su construcción podemos usar los menús de un documento html. Sin embargo, para incrustar el código java debemos editar el documento desde la vista *Código Fuente*. El código final es el siguiente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>respuesta.jsp</TITLE>
</HEAD>
<BODY>
<%
//DEFINIMOS LAS VARIABLES A USAR

String n, a, nombre_completo=null;

//Variable "n", para recibir el campo nombre
//Variable "a", para recibir el campo apellido
//Variable nombre_completo, para concatenar el nombre y el apellido en un
solo string

//Recibimos los campos nombre y apellido
//el código "(String)" es para dar formato a la variable nombre como
//string y de esa forma poder asignarla a la variable n que es un string

n = (String) request.getParameter("nombre");
a = (String) request.getParameter("apellido");

//Concatenamos n y a

nombre_completo=n+" "+a;

//Si n=René y a=Díaz, entonces ahora la variable nombre_completo="René
Díaz"%>
//DESPLEGAMOS EN PANTALLA EL RESULTADO
<CENTER>El nombre completo es <%=nombre_completo%></CENTER>
</BODY>
</HTML>
```

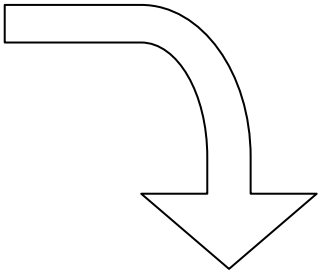
Una vez que hayamos guardado y compilado ambos archivos mediante CTRL+S debemos iniciar el servidor de prueba. Para ello hacemos clic con el botón

derecho del mouse sobre el archivo inicio.jsp (en la vista web) y seleccionamos la opción *ejecutar en servidor*. Ello hará que se inicie el servidor de prueba incorporado en websphere y que se abra la vista de servidor[2]

Luego al completar el formulario el resultado debiese ser el siguiente:




inicio.html




respuesta.jsp

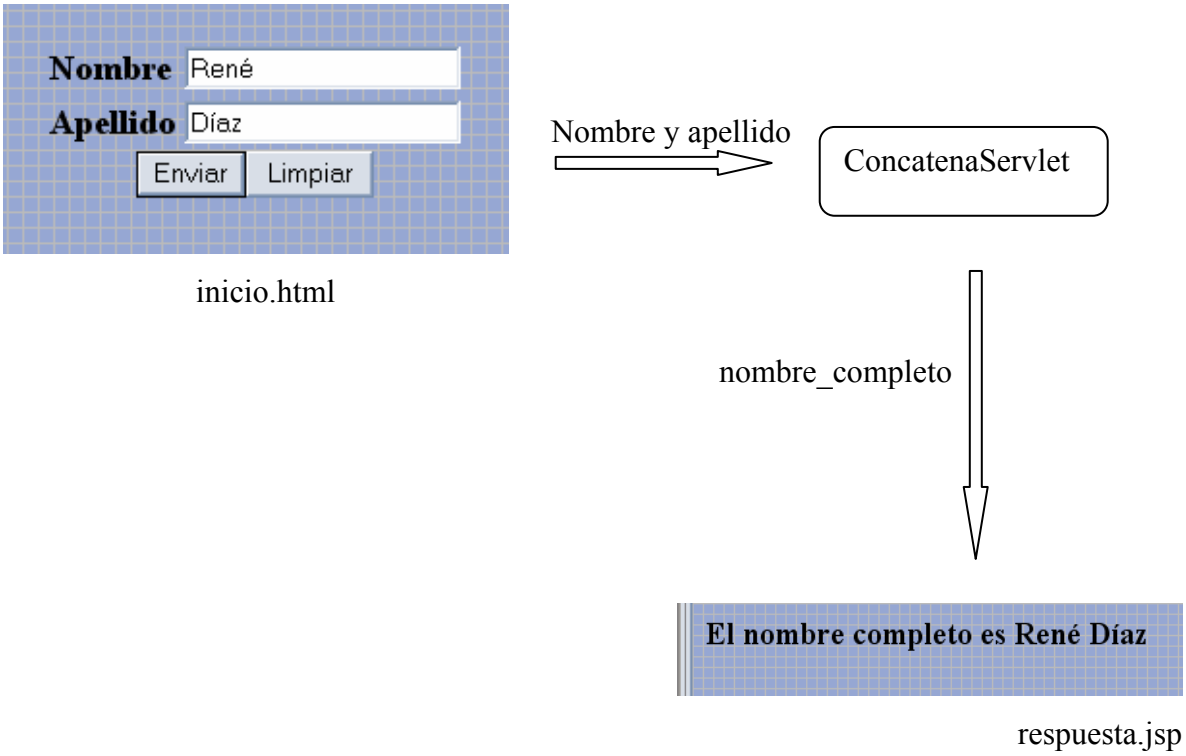
***Resumen:*** **El cliente ingresa los datos a través de la página inicio.html. Al hacer clic en enviar, los datos son posteados a la página respuesta.jsp la cual recepciona estos valores, los concatena en un solo String y luego los "despliega en pantalla" (En estricto rigor lo que hace es construir el código html y enviarlo al cliente).**

[2] Si bien en la vista servidor Ud. puede continuar editando el documento, es preferible que para ello se cambie a la vista web.

# 4 EJEMPLO 2

## 4.1 Planteamiento del problema

Nos proponemos ahora evolucionar levemente respecto del ejemplo uno. La idea entonces es construir nuevamente un concatenador, pero en esta ocasión queremos que el archivo *inicio.html* envié los datos a un servlet, *ConcatenaServlet* el cual se encargue de concatenar el nombre y apellido en un solo string enviándoselo luego a un documento jsp a través de un bean para que este despliegue la información en pantalla:




inicio.html

respuesta.jsp

## 4.2 Consideraciones Teóricas

Lo que haremos en este ejemplo es traspasar la lógica del problema a un servlet dejando en manos de una página jsp la presentación de la información. Mayores detalles al respecto de lo que es un servlet y sus características han sido entregados en clases de cátedra por lo que resulta inoficioso hacerlo aquí. Solo diremos que es una clase java que extiende a la clase HttpServlet por lo que permite operar sobre el web. Sin

embargo, es importante puntualizar que el envío de información desde el servlet a la página jsp será a través de un objeto RequestDispatcher y un bean. El primero permite registrar y despachar la información a enviar, el segundo permite recepcionarla en un documento jsp. También es importante señalar que al cliente se le enviará el resultado del procesamiento de la página *respuesta.jsp* que es código html.

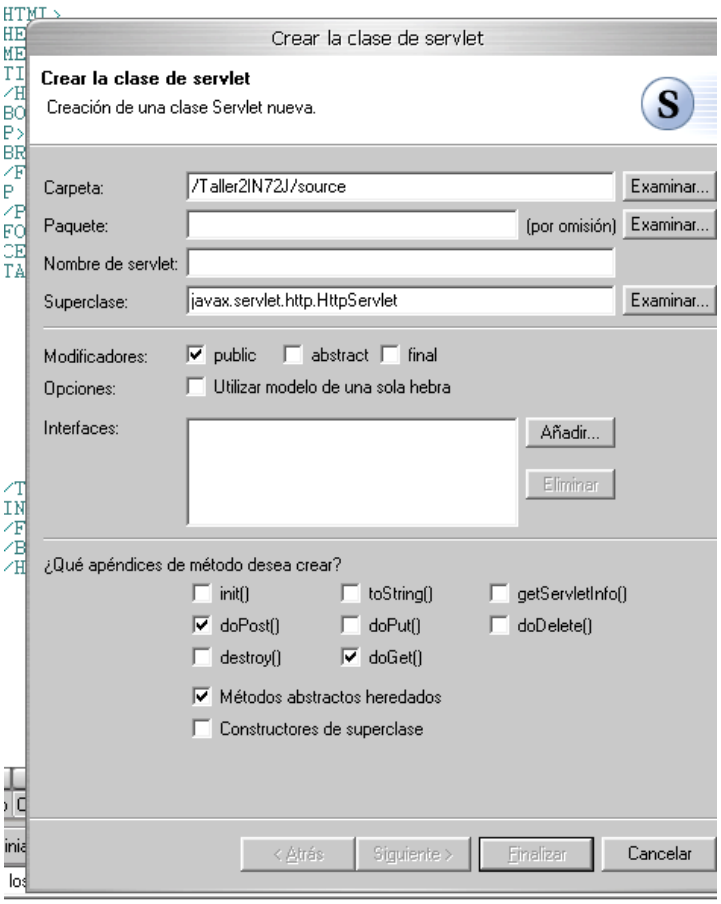
Por último, tanto para este ejemplo, como para el que sigue son válidas las consideraciones teóricas entregadas en el ejemplo uno respecto del funcionamiento de una página jsp.

### 4.3 Solución

Nuevamente requerimos de un documento html con un formulario para recepcionar la información del cliente. Sin embargo, el documento *inicio.html* que construimos en el ejemplo uno es válido para este. Solo es necesario modificar el campo action del formulario ya que en esta oportunidad la información del formulario será enviada al servlet *ConcatenaServlet*, luego es este valor el que debemos colocar en dicho campo.

Ahora debemos construir el servlet que nos permitirá implementar la lógica del problema. Para ello vamos a la carpeta source, hacemos clic con el botón derecho del mouse, seleccionamos el menú *Nuevo* y dentro de el la opción *Servlet.* Se desplegará en wizard como el que se muestra en la siguiente página.

En el es necesario completar el nombre del Servlet, en nuestro caso *ConcatenaServlet*. Además podemos indicar que el servlet sea creado en un paquete específico. También podemos indicar al programa que preconstruya los encabezados de los métodos estándar de un servlet que necesitemos. En este caso no es necesario modificar ninguna de las opciones que vienen por defecto, basta con señalar el nombre del servlet. Una vez que haga clic en siguiente el archivo será construido y a la derecha aparecerá el código para ser editado por Ud.




Ahora bien, el código que implementa el servlet deberá considerar en uno de los métodos (doPost o doGet) los siguientes:

1. Recibir el nombre y apellido desde la página inicio.jsp.
2. Concatenar el nombre y apellido en un solo String
3. Enviar el string concatenado a la página respuesta.jsp.

El código que implementa el servlet se presenta a continuación.

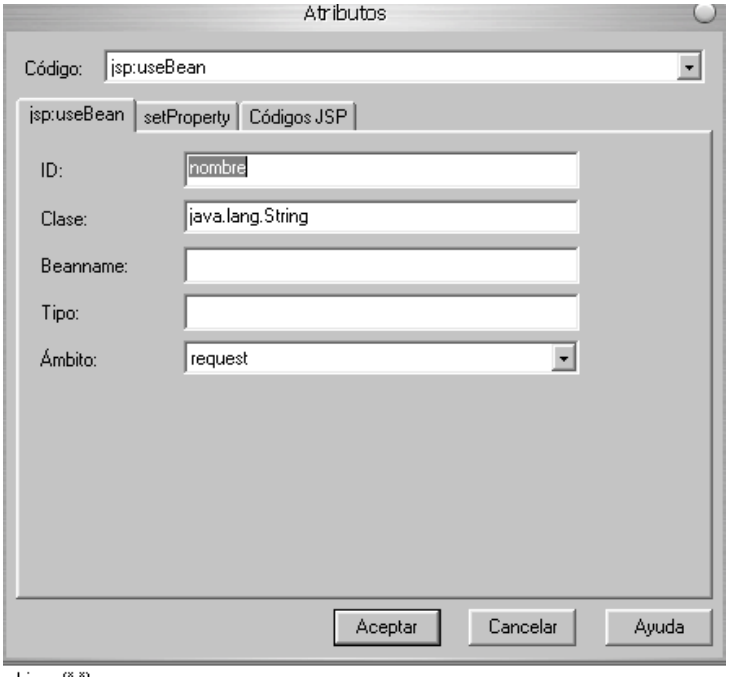
```
//Importamos paquetes estándar.
import javax.servlet.http.HttpServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.lang.Object.*;
import java.util.*;


public class ConcatenaServlet extends HttpServlet {


      public void doPost(
            javax.servlet.http.HttpServletRequest request,
            javax.servlet.http.HttpServletResponse response)
            throws javax.servlet.ServletException, java.io.IOException {

//Definimos las variables en que recibiremos el nombre y el apellido
                  String n,a=null;

//variable en la que concatenaremos el nombre completo
                  String nombre_completo=null;


//variable de tipo RequestDispatcher que se utiliza para enviar
//resultados a una página jsp
```

```
                  RequestDispatcher disp = null;

//Recibimos las variables desde la página inicio.html

                  n = (String) request.getParameter("nombre");
                  a = (String) request.getParameter("apellido");


//Concatenamos el nombre

                  nombre_completo=n+" "+a;
// Si n=Juan y a= Moya => nombre_completo="Juan Moya"


//Registramos la variable nombre_completo en el objeto request dentro del
cual se conocerá como n_a_mostrar

//Eventualmente podemos registrar en el objeto request muchas variables,
//tantas como recursos computacionales tengamos. Todas serán despachadas
//a la página jsp de destino.

                  request.setAttribute("n_a_mostrar", nombre_completo);



//Le indicamos al despachador donde enviar el objeto request

      disp = getServletContext().getRequestDispatcher("/respuesta.jsp");


//Enviamos el objeto request usando el objeto RequestDispatcher.

                  disp.forward(request, response);

      }

      public void doGet(
            javax.servlet.http.HttpServletRequest request,
            javax.servlet.http.HttpServletResponse response)
            throws javax.servlet.ServletException, java.io.IOException {

//Esta línea redirecciona el método Donet al método doPost del servlet.
//Así evitamos replicar el código

                  this.doPost(request, response);
      }

}
```

***Resumen:*** **Hasta aquí contamos con la página inicio.html que recibe los datos del cliente y los envía al servlet ConcatenaServlet. Este recibe los datos, los concatena y luego los despacha a la página respuesta.jsp. Falta entonces construir la página respuesta .jsp.**

La página *respuesta.jsp* debe recibir por medio de un bean la variable *n_a_mostrar*, que esta contenida en el objeto request, para luego desplegarla en pantalla. No olvidemos que una página jsp es código html con incrustaciones de código java, que es compilada y procesada en el servidor y que al cliente solo se le envía el resultado de este proceso que corresponde a un documento html construido a partir de las especificaciones de la página jsp.

Creamos entonces un documento JSP de nombre *respuesta.jsp*. Lo primero que debemos hacer es insertar un bean por cada variable del objeto request que queramos manejar en el documento JSP. Para ello vamos a la vista de diseño del documento jsp, menú JSP, *Insertar bean* y se desplegará el siguiente wizard:



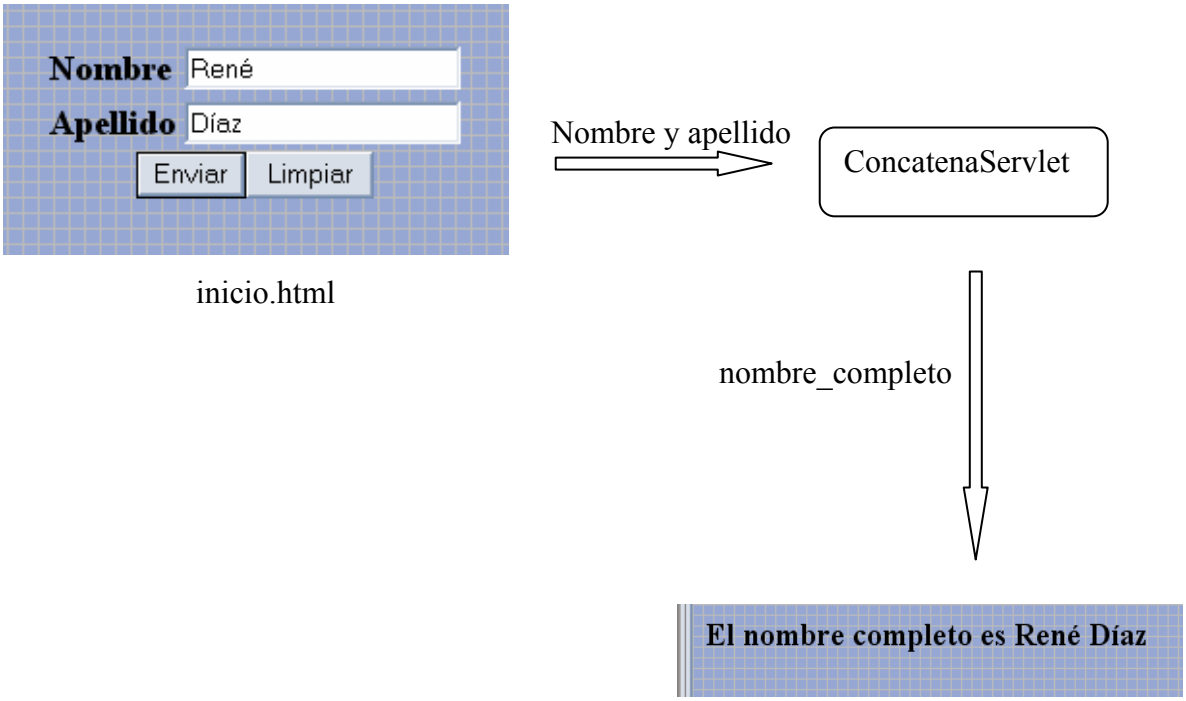

En el campo ID debemos colocar el nombre que identifica la variable a rescatar, en nuestro caso n_a_mostrar. En el campo clase debemos identificar el tipo de objeto que es, en nuestro caso, la variable n del servlet que generó la variable n_a_mostrar es del tipo java.lang.String. Por último en el campo ámbito debemos seleccionar request.

Hecho esto editamos el contenido html y luego desde la vista *código fuente* insertamos la variable donde corresponda. EL código fuente es el siguiente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>respuesta.jsp</TITLE>
</HEAD>
<BODY background="b002bcg.gif">
<jsp:useBean id="n_a_mostrar" class="java.lang.String" scope="request"/>
<P><BR>
```

```
<BR>
<B><FONT size="+2">El nombre completo es <%=n_a_mostrar%></FONT></B></P>
</BODY>
</HTML>
```

Luego realizamos el mismo procedimiento que en el ejemplo uno sobre el archivo inicio.html para ejecutarlo en el servidor. Al hacerlo el resultado debiese ser el siguiente:




inicio.html

respuesta.jsp

# 5 EJEMPLO 3

## 5.1 Planteamiento del problema

Deseamos ahora mostrar la interacción de un servlet con una base de datos. Para tales efectos mostraremos como crear una base de datos, utilizando Power Designer, sobre

IBM DB2. Luego crearemos un documento html con dos links que establecen parámetros de búsqueda sobre la base de datos. Esta información será recepcionada por el servlet quien ejecutará la consulta a la base de datos aplicando los parámetros de filtro entregados por el cliente para luego enviar los resultados a una página jsp que se encargará de desplegarlos en pantalla construyendo un documento html.

Para que el cliente entregue sus parámetros de búsqueda usaremos links. Sin duda esta opción no es la usual, su único objetivo es mostrar como enviar información a través de un link. Lo mismo se podría realizar mediante un formulario.

Por otro lado obviaremos aquí consideraciones teóricas respecto del problema ya que estas fueron explicitadas en los ejemplos anteriores y aplican igualmente a este. Respecto de las consideraciones para el diseño de una base de datos se adjunta un documento pdf con transparencias del curso IN55A – Sistemas de información administrativos, dictado por el profesor Eduardo Schälchli. Mayores antecedentes respecto del modelo entidad relación y modelo relacional de bases de datos se pueden encontrar en internet o bien en literatura especializada.

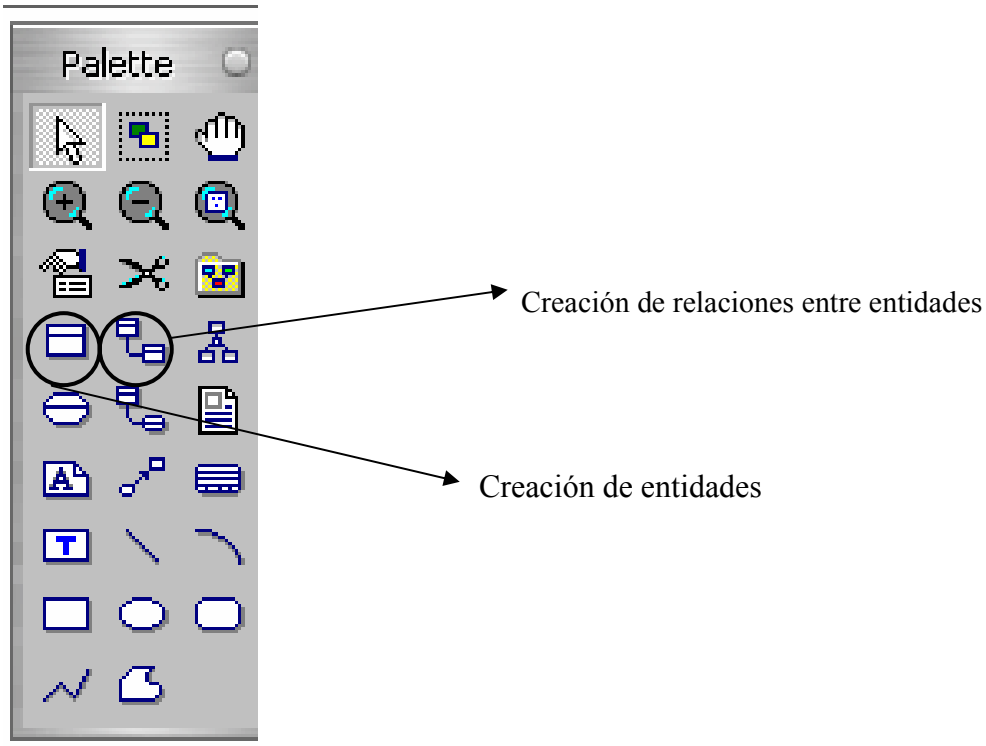
## 5.2 Creación de una base de datos.

Como señalamos no es nuestro objetivo aquí enseñar a diseñar una base de datos por lo que obviaremos comentarios al respecto. Para crear una base de datos, lo primero que debemos hacer es contar con un diseño. Usaremos para tales efectos el programa Power Designer. Los pasos a seguir para crear un documento de diseño de bases de datos entidad relación son los siguientes:

1. Menú File
2. New
3. Conceptual Data Model, Aceptar

En el navegador aparecerá una “carpeta” con el modelo recién creado. Dentro de el aparece un diagrama que es sobre el cual diseñaremos la base de datos.

A la derecha aparecerá una barra de herramientas que nos permitirá insertar los distintos elementos que componen el modelo:




Una vez diseñada la base de datos[3] procedemos a crear el diseño físico que corresponde al modelo relacional a partir del cual crearemos la base de datos. Vamos al menú *Tools*, *Generate Phisical Model.*

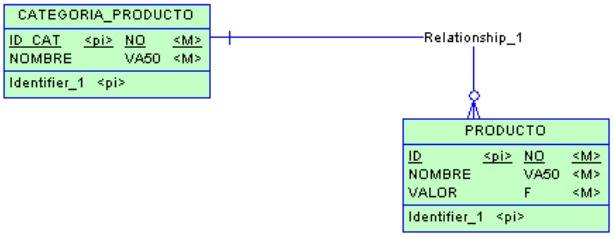
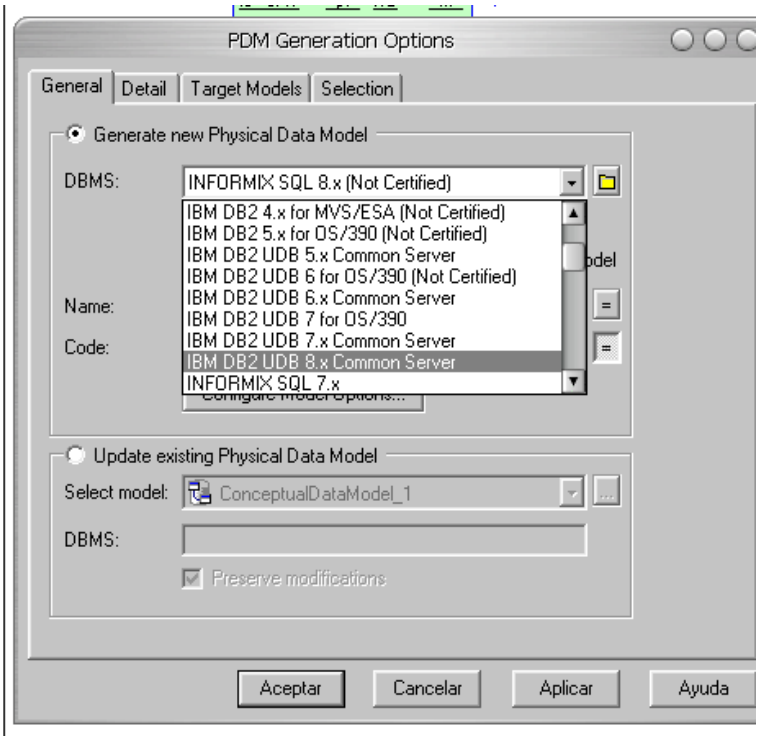



**Ilustración 1: Modelo Conceptual Canónico**

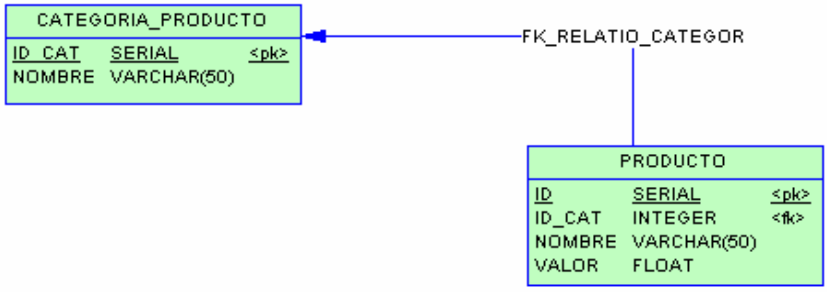
Se desplegará un wizard como el siguiente:

[3] Entendemos por un modelo correcto, para efectos del software, aquel que tiene establecidos el nombre de las entidades, atributos, el tipo de cada atributo, las llaves primarias y las relaciones entre entidades.

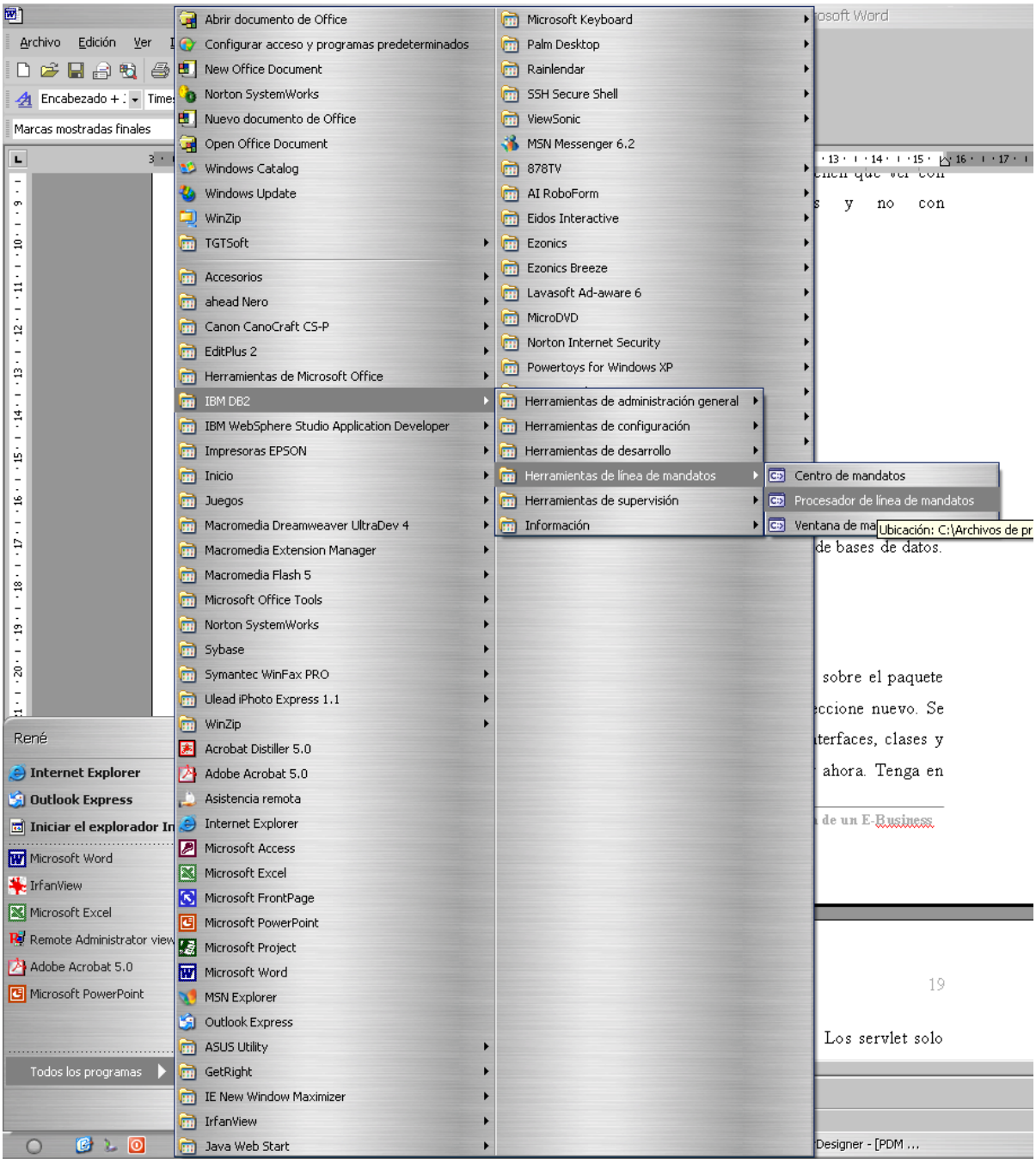



EN el campo DBMS seleccionamos el motor de bases de datos en el cual la crearemos. En nuestro caso *IBM DB2 UDB 8.x Common Server* y luego aceptamos. Se generará el modelo físico de datos y aparecerá una ventana con los errores encontrados. Si aparecen warnings no se preocupe, tienen que ver con formalidades y no con errores.

El modelo físico de datos es el siguiente:

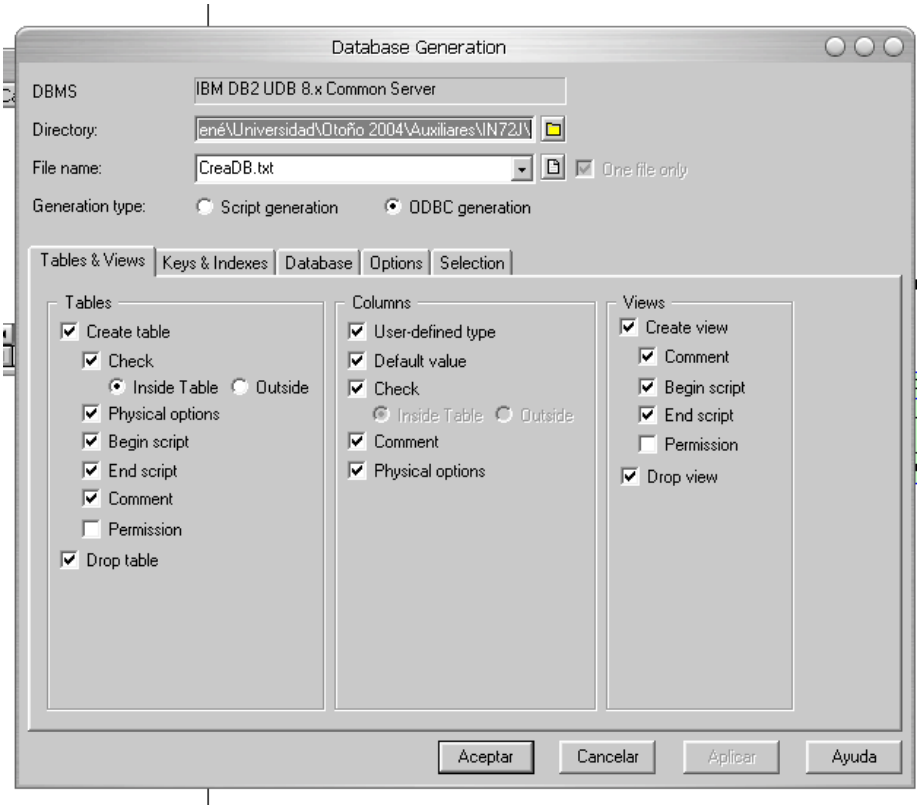



Ahora bien, procederemos a crear la base de datos en el motor de bases de datos. Para eso abriremos una ventana de procesamiento de mandatos del DB2. Se abrirá una ventana similar a las de DOS. En ella ejecutaremos el mandato *create database nombre_base_datos.* En nuestro caso ejecutaremos *create database EJ72J.* Este comando creará la base de datos, es decir, la estructura que permite implementar las tablas y relaciones que hemos diseñado.


Abrir documento de Office
Configurar acceso y programas predeterminados
New Office Document
Norton SystemWorks
Nuevo documento de Office
Open Office Document
Windows Catalog
Windows Update
WinZip
TGTSoft
Accesorios
ahead Nero
Canon CanoCraft CS-P
EditPlus 2
Herramientas de Microsoft Office
IBM DB2
IBM WebSphere Studio Application Developer
Impresoras EPSON
Inicio
Juegos
Macromedia Dreamweaver UltraDev 4
Macromedia Extension Manager
Macromedia Flash 5
Microsoft Office Tools
Norton SystemWorks
Sybase
Symantec WinFax PRO
Ulead iPhoto Express 1.1
WinZip
Acrobat Distiller 5.0
Adobe Acrobat 5.0
Asistencia remota
Internet Explorer
Microsoft Access
Microsoft Excel
Microsoft FrontPage
Microsoft PowerPoint
Microsoft Project
Microsoft Word
MSN Explorer
Outlook Express
ASUS Utility
GetRight
IE New Window Maximizer
IrfanView
Java Web Start
Microsoft Keyboard
Palm Desktop
Rainlendar
SSH Secure Shell
ViewSonic
MSN Messenger 6.2
878TV
AI RoboForm
Eidos Interactive
Ezonics
Ezonics Breeze
Lavasoft Ad-aware 6
MicroDVD
Norton Internet Security
Powertoys for Windows XP
Herramientas de administración general
Herramientas de configuración
Herramientas de desarrollo
Herramientas de línea de mandatos
Herramientas de supervisión
Información
Centro de mandatos
Procesador de línea de mandatos
Ventana de ma
Ubicación: C:\Archivos de pr
René
Internet Explorer
Outlook Express
Iniciar el explorador In
Microsoft Word
IrfanView
Microsoft Excel
Remote Administrator view
Adobe Acrobat 5.0
Microsoft PowerPoint
Todos los programas
Archivo
Edición
Ver
Encabezado + :
Marcas mostradas finales
de bases de datos.
sobre el paquete
ahora. Tenga en
de un E-Business
19
Los servlet solo
Designer - [PDM ...


El siguiente paso consiste en crear las tablas y relaciones que implementan nuestra base de datos. Para ello emplearemos el Power Designer.

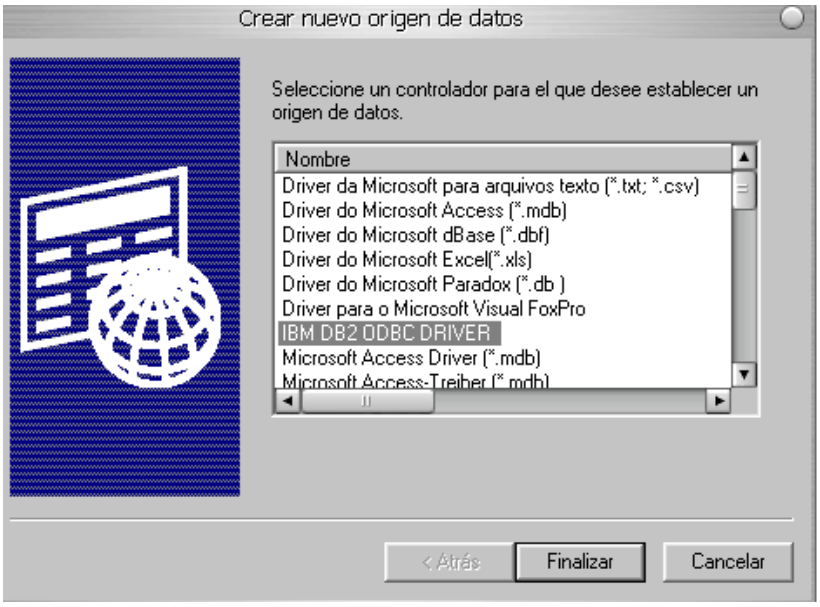
En el modelo físico vamos al menú *Database* y seleccionamos *Generate Database*. Se desplegará e

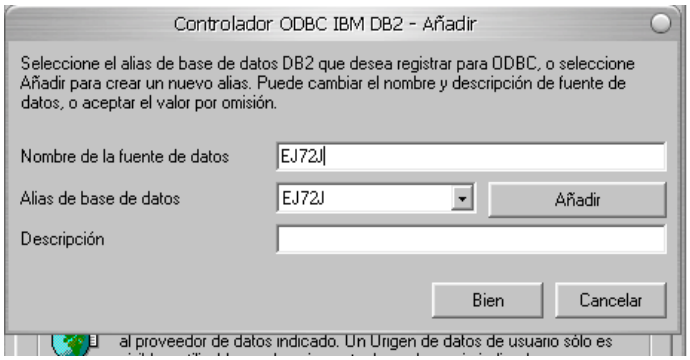



En primer lugar debemos seleccionar en *Generation type* la opción ODBC generation. Esto nos permitirá implementar directamente la base de datos en el motor sobre la BD EJ72J y además nos creará un archivo con las sentencias de

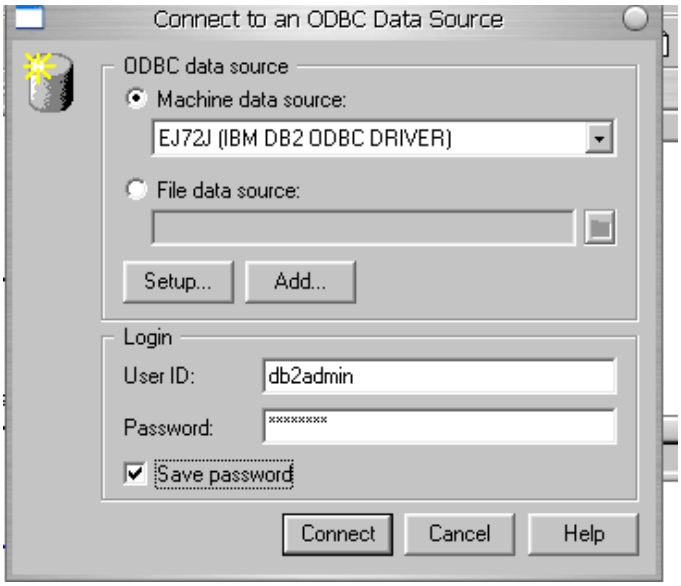
creación. El nombre del archivo lo especificamos en *File name* y en *directory* indicamos el directorio donde este archivo será creado. El resto de las opciones las dejamos tal como están. Hacemos clic en aceptar y aparecerá la siguiente ventana:


Seleccionamos *Machina data source* y luego hacemos clic en add. Lo que haremos es crear un driver de conexión con nuestra base de datos de manera que podamos implementar las tablas y relaciones. Aparecerá un wizard con el administrador de ODBC de windows. En el seleccionamos *dBASE Files* y luego hacemos clic en agregar. Se desplegará un wizard como el que se muestra en la figura a la siguiente. En el debemos seleccionar el driver *IBM DB2 ODBC DRIVER*. Hacemos clic en finalizar y se desplegará la ventana que se muestra en la siguiente página.

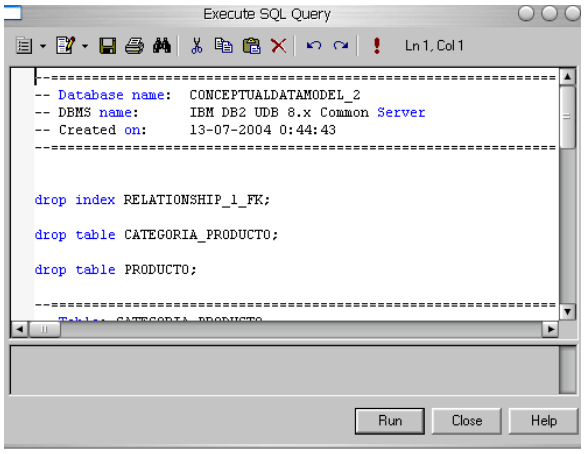





En el campo *nombre de la fuente de datos* colocamos el nombre de nuestra base de datos EJ72J. En el campo *Alias de la base de datos*, seleccionamos de la lista la base de datos EJ72J que recién creamos. Luego hacemos clic en *bien* y *aceptar*, hasta llegar a la ventana original de conexión a la base de datos (se muestra a continuación). Ahí en el campo *User ID* y *Password* señalamos el username y password del administrador del motor de bases de datos (este user y password Ud. lo definió al instalar DB2 en su computador.)

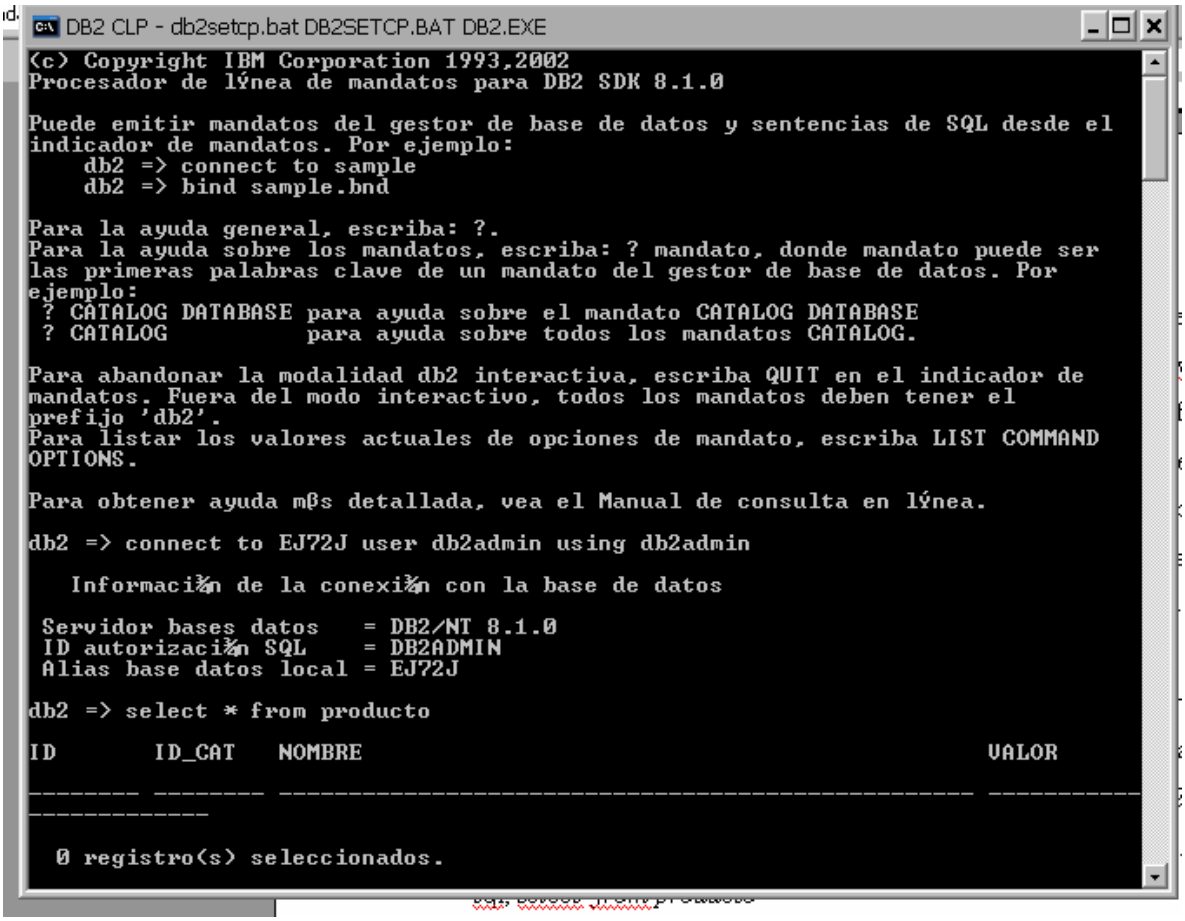



Además en *Machine data source* seleccionamos el driver que recién creamos, *EJ72J IBM DB2 ODBC DRIVER*. Luego hacemos click en *Connect*. Luego de esto el programa creará el archivo con las sentencias de creación de tablas, relaciones, llaves primarias y foráneas de la base de datos a implementar. Una vez hecho aquello desplegará una ventana como la que se muestra a la derecha. En ella se muestran las sentencias señaladas. Haciendo clic en *Run* estas se ejecutarán contra la base de datos. Si no aparece esta ventana significa que Power Designer no pudo conectarse a la base de datos EJ72J, lo más probable es que Ud. haya cometido algún error o que esté usando un sistema operativo que no posee



tecnología NT para el caso de windows por ejemplo. Al hacer clic en run, puede que le aparezcan algunas ventanas de error, ellas responden a los warnings que aparecieron al crear el modelo físico y usualmente tienen que ver con que no fueron definidos nombres de relaciones u otros. En estos casos ignórelos (hacer clic en ignore). En el caso que el error en cuestión hable de algún atributo no definido, llave primaria o foránea no definida, entonces aborte la operación, corrija el error y repita la secuencia para la implementación de la base de datos. NO CREE LA BASE DE DATOS NUEVAMENTE DESDE EL DB2. Una vez terminado el proceso haga clic en close.

Ud. podrá verificar que todo se ha creado correctamente por medio del procesador de línea de mandatos del DB2( el que usamos para crear la base de datos). Conéctese a la base de datos ejecutando la sentencia *connect to EJ72J user db2admin using db2admin,* donde db2admin es en mi caso el username y password de administrador del DB2. Luego ejecute la sentencia sql, *select * from producto*. Si aparece lo siguiente, debería estar todo ok.



A continuación debemos poblar nuestra base de datos con algunos datos de prueba. Para ello completaremos las tablas como se muestra a continuación:

**TABLA CATEGORIA_PRODUCTO**

| ID_CAT | NOMBRE |
|---|---|
| *Este valor es generado automáticamente por el motor de base de datos ya que fue definido como un dato de tipo serial. El valor es secuencial entero y positivo partiendo de 1.* | ELECTRONICA |

**TABLA PRODUCTO**

| .ID | ID_CAT | NOMBRE | VALOR |
|---|---|---|---|
| *Este valor es generado automáticamente por el motor de base de datos ya que fue definido como un dato de tipo serial. El valor es secuencial entero y positivo partiendo de 1.* | 1 | Cámara Sony TRV260 | 260000 |
| | 1 | Trípode Sony | 35000 |
| | 1 | Mic. Ambiental | 68000 |
| | 1 | Atril Mic | 28000 |
| | 1 | Cable mic | 6000 |
| | 1 | Parlantes Creative | 24000 |

Como se señala tanto el campo *ID_CAT* como *ID* son generados automáticamente por la base de datos porque así fue diseñada. Ahora bien, el campo *ID_CAT* de la tabla *Producto* es el que establece la relación a nivel físico entre las dos tablas, también se conoce como llave foránea. Debe existir integridad referencial entre ambos, esto es, en el campo *ID_CAT* de la tabla *Producto* no puede haber un registro con un valor que no esté registrado en el correspondiente campo de la tabla *Categoría_producto*.

El conjunto de sentencias que registran los datos antes generados se presenta a continuación en el mismo orden de las tablas.

**TABLA CATEGORIA_PRODUCTO**

*insert into CATEGORIA_PRODUCTO(NOMBRE) values('ELECTRONICA')*

**TABLA PRODUCTO**

*insert into PRODUCTO(ID_CAT, NOMBRE, VALOR) values(1,'Camara Sony TRV260', 260000)*

*insert into PRODUCTO(ID_CAT, NOMBRE, VALOR) values(1,'Tripode Sony' , 35000)*
*insert into PRODUCTO(ID_CAT, NOMBRE, VALOR) values(1,'Mic. Ambiental', 68000)*
*insert into PRODUCTO(ID_CAT, NOMBRE, VALOR) values(1,'Atril Mic', 28000)*
*insert into PRODUCTO(ID_CAT, NOMBRE, VALOR) values(1,'Cable Mic',6000)*
*insert into PRODUCTO(ID_CAT, NOMBRE, VALOR) values(1,'Parlantes Creative',24000)*

Nótese que los campos que son de tipo texto se ingresan entre cremillas simples y que los campos numéricos se ingresan sin cremillas. También note que los campos seriales o autonuméricos no son declarados en la lista de columnas de la tabla y por su puesto, dada su calidad no son ingresados en la sentencia (caso tabla categoría_producto).

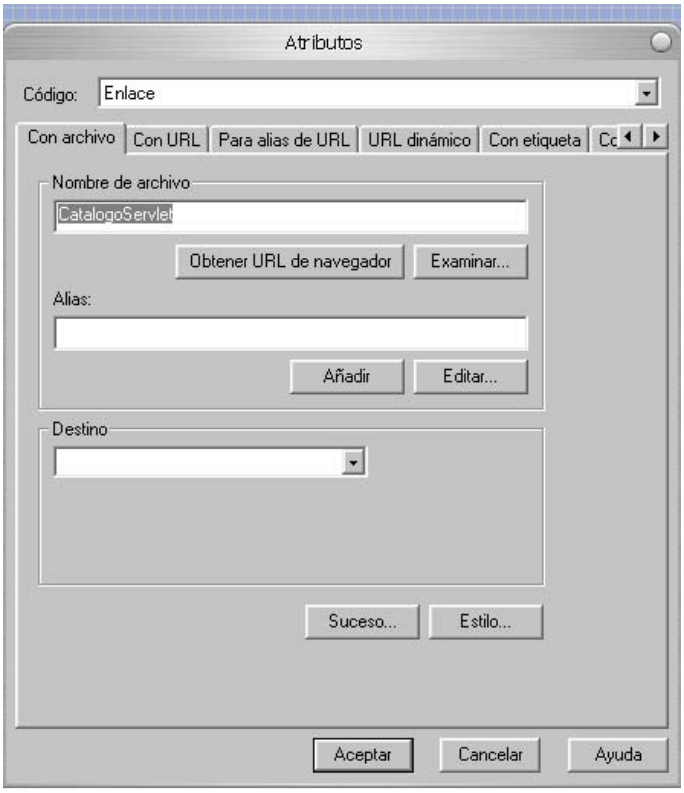
En la misma ventana de procesamiento de mandatos del DB2 o en otra ejecute estas sentencias contra la base de datos y los registros serán ingresados a la misma. (es necesario estar conectado a la base de datos, para ello refiérase a la página 23). En esta misma ventana podrá verificar que lo datos hayan sido ingresados correctamente ejecutando *select * from categoría_producto* y *select * from producto*.

***Resumen:*** **Hasta aquí solo hemos creado la base de datos. Corresponde ahora crear la aplicación que interactuar con ella.**

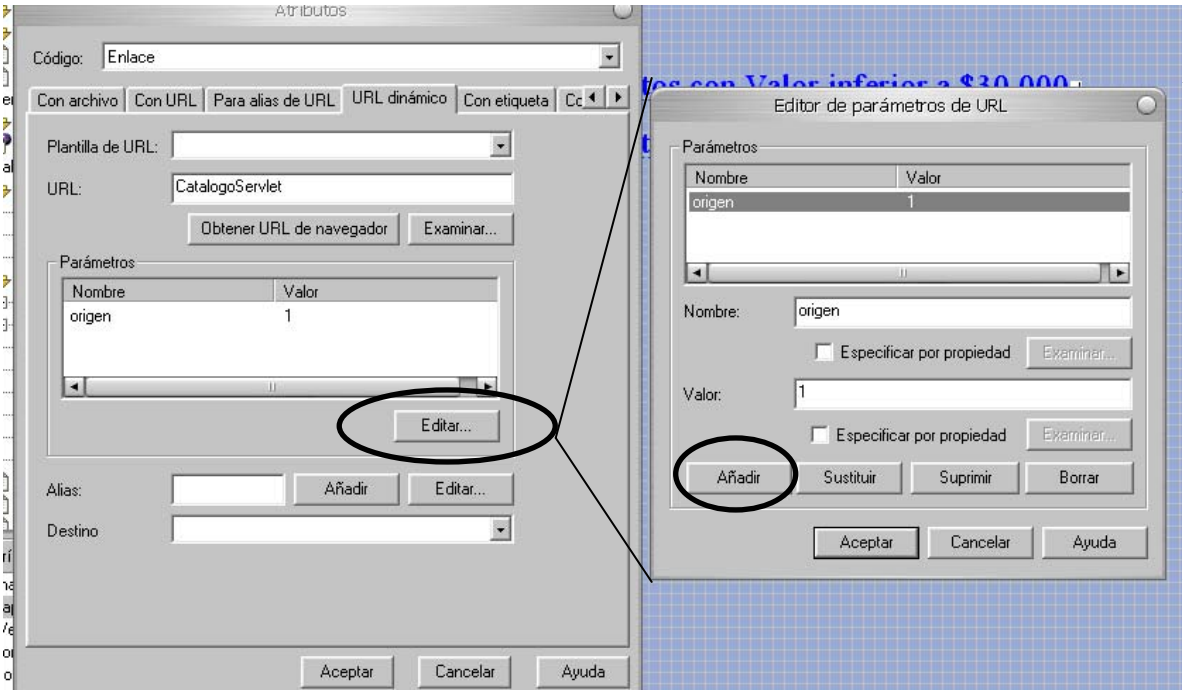
### 5.3 Construcción de la aplicación

Nos disponemos ahora a construir la aplicación que nos permitirá conectarnos a la base de datos recién creada. Como señalamos anteriormente, para ello debemos crear una página de recepción de información del cliente, *inicio.html*, un servlet, *CatalogoServlet* y una página de respuesta, *respuesta.jsp*. La idea es entonces presentar en la página *inicio.jsp* dos links al cliente, uno en el cual se solicita desplegar los productos con un valor inferior a 30.000 y otra en la cual se solicita desplegar los productos con un valor igual o superior a 30.000. De esta forma, a través de los links enviaremos una variable llamada origen la que para el primer caso tendrá un valor 1 y para el segundo 2. De esta forma en el Servlet podremos distinguir cual fue la opción del cliente. Esto es análogo a crear un formulario con un campo de selección donde el cliente escoja una u otra opción.

De la manera que ya conocemos crearemos un documento html. Desde la vista de diseño vamos al menú *insertar* y luego *enlazar*. Se desplegará la siguiente ventana




En el campo *Nombre de Archivo* colocaremos el destino del enlace o link, en nuestro caso *CatalogoServlet.* Luego vamos a URL dinámico y aparecerá la ventana que se presenta a continuación. En *URL* colocamos *CatalgoServlet* y luego hacemos clic en editar. Se abrirá un wizard en el cual ingresaremos las variables a postear con el link y sus valores. En nuestro caso el campo *Nombre* lo completamos con *origen* y en el campo valor colocamos el valor 1. Luego hacemos clic en *añadir* y *aceptar*. Aceptamos en la ventana original y aparecerá en



un documento jsp podría señalarse el valor de la variable mediante un valor dinámico, por ejemplo <A HREF=”CatalogoServlet?origen=<%=monto%>”>Texto</A>.

Para efectos de la implementación de nuestra aplicación será necesario contar con una clase a la cual llamaremos Producto. Para ello seguimos el mismo procedimiento que para crear un servlet, pero en el menú nuevo seleccionamos *Otros*. Se abrirá una ventana en la que seleccionaremos del navegador izquierdo la opción *java*. Luego en el navegador derecho se desplegarán todos los documentos java que podemos crear. Seleccionamos *Clase* y luego hacemos clic en siguiente. El wizard que se abrirá le parecerá familiar. Solo debe completar el nombre de la clase, en nuestro caso *Producto*. El código que implementa la clase es el siguiente:

```
public class Producto {


	//Variables de la clase
	String id, nombre=null;
	int valor=0;


	//Constructor de la clase que recibe tres variables, identificador,
nombrey valor. Y las asigna a las variables de la clase

public Producto(String identificador, String nombre_producto, int
valor_producto){

		id=identificador;
		nombre=nombre_producto;
		valor=valor_producto;

	}

	//Getters

	public String RecuperaIdentificador(){

		return id;
	}

	public String RecuperaNombre(){

		return nombre;
	}

	public int RecuperaValor(){

		return valor;
	}


}
```

Como se observa esta clase define una estructura de datos que nos permitirá almacenar en un objeto de tipo Producto cada registro de la base de datos que rescatemos de ella, en particular los campos id, nombre y valor.

Corresponde ahora crear el Servlet *CatalogoServlet*. Lo construimos de la misma forma que explicamos en el ejemplo 2 con la diferencia que entre los métodos a implementar seleccionamos además de los ya seleccionados el método INIT(). El servlet deberá contemplar los siguientes aspectos:

1. Recibir la variable origen desde el cliente.
2. Conectarse a la base de datos
3. Ejecutar una consulta sql a la base de datos para rescatar los productos de la tabla producto de la base de datos. Rescatamos el campo id,

nombre y valor. Luego los almacenamos en un vector de objetos de tipo producto.

4. Filtramos el vector con los datos que obtuvimos según el valor de la variable origen. Vamos guardando los objetos Producto que contienen información que debe ser desplegada en pantalla en otro vector de objetos Producto auxiliar.
5. Luego usamos un objeto RequestDispatcher para enviar la información a la página respuesta.jsp.

Antes de seguir es importante aclarar que es un vector. Un Vector es un objeto del tipo java.util.Vector. En palabras simples se trata de un arreglo de largo variable y que en sus registros puede almacenar cualquier tipo de objetos, enteros, String y en particular para nuestro caso objetos de tipo Producto. Posee una serie de métodos que permiten administrar el arreglo, a saber:

- vector.addElement(p): agrega el objeto p al final de vector de nombre vector.
- vector.elementAt(i): recupera el objeto del casillero i-ésimo del vector de nombre vector. La indexación de un vector parte en 0, es decir, el primer casillero del vector es referenciado por el valor 0.
- vector.size(): recupera el largo del vector de nombre vector. El largo se refiere a la cantidad de registros.

Otro aspecto a tener en cuenta es que un Servlet como cualquier clase permite implementar métodos que pueden ser usados en ella o inclusive heredados por otra clase de nivel inferior. En particular un Servlet puede manejar ciertos métodos por "defecto" tales como doPost, doGet, init, destroy, etc, pero además podemos crear otros métodos. Para efectos de la conexión a la base de datos crearemos un método llamado Conectar() que no recibe parámetros, se encarga de establecer la conexión y que retorna un objeto de tipo *Connection* que contiene la conexión a la base de datos.

Por último respecto de la conexión a la base de datos y la consulta de datos debemos considerar lo siguiente:

- La conexión a la base de datos se “almacena” en un objeto de tipo *Connection*. Para establecer la conexión es necesario llamar al driver que corresponde y entregar la ruta, username y password de conexión.
- Para poder ejecutar consultas a la base de datos se requiere de una suerte de administrador de la conexión, en java ello corresponde a un objeto Statement.
- Por medio del objeto Statement y usando un objeto ResultSet se ejecuta una query o consulta a la base de datos. Un objeto ResultSet almacena el resultado de la consulta ejecutada contra la base de datos en un formato de tabla. Luego permite moverse por cada una de las filas de las tablas mediante el método next() que se mueve al siguiente registro de la tabla y retorna trae en caso de existir o false en caso contrario. Ahora bien, para rescatar el valor de una columna o campo de la fila en que está situado el objeto ResultSet se emplea el método getInt(“nombre_columna”) o getString(“nombre_columna”) según se desee traer el dato con formato entero o string respectivamente.

El código que implementa el servlet es el siguiente:

```
//Importamos paquetes estándar
import javax.servlet.http.HttpServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.lang.Object.*;
import java.util.*;
import Producto; //Importamos la clase Producto


public class CatalogoServlet extends HttpServlet {

      //Definimos las variables a utilizar.

      Vector productos= new Vector();//Vector para registrar la info de
      //la base de datos
      Connection conexion = null; //Variable de tipo conexión
      String origen=null; //Variable en la que recibiremos la variable
      //origen enviada por el cliente
      Producto producto, p_aux; //Objetos de Tipo producto

      //En el método init establecemos la conexión con la base de datos y
      //guardamos la información de esta en el vector productos
      public void init() throws javax.servlet.ServletException {


            Statement statement = null; //Variable que administra la
            //conexión con la base de datos
            ResultSet rs = null; //Objeto Resultset que recibe los datos
            //consultados a la base de datos por medio de una query

            //Guardamos en la variable conexión el resultado de la
            //ejecución del método Conectar() que es la conexión a la
            //base de datos EJ72J
            conexion = Conectar();

            //Ejecutamos la rutina para consultar los datos de la tabla
            //producto

            try {
                  //Inicializamos la variable de administración de la
                  //conexión
                  statement = conexion.createStatement();

                  //Ejecutamos la query
                  rs =
                        statement.executeQuery("SELECT * FROM PRODUCTO");

                  //mediante un ciclo while vamos retirando una a una las
                  //filas de resultado de la consulta y las vamos
                  //guardando en un objeto producto el cual a su vez lo
                  //registramos en el vector productos
                  while (rs.next()) {

                        //Construimos el objeto producto mediante el
                        //constructor de la clase
                        producto = new Producto(rs.getString("ID"),
                  rs.getString("NOMBRE"), rs.getInt("VALOR"));

                        //Agregamos el objeto producto al vector productos
                        this.productos.addElement(producto);
```

```
                    //Si quedan más registros (rs.next()) volvemos a
                    //repetir el ciclo while
              }

              //Cerramos el objeto ResultSet y el Objeto Statement
              rs.close();
              statement.close();

        } catch (Exception e) {
              e.printStackTrace();

        }


        //El uso de las sentencias try y match tiene que ver con el
        //manejo de excepciones en java. EL programa trata de
        //ejecutar el código dentro del try y de no poder hacerlo
        //ejecuta el match enviando la excepción. Este aspecto es
        //obligatorio en el manejo de consultas a bases de datos en
        //java
}

public void doPost(
        javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response)
        throws javax.servlet.ServletException, java.io.IOException {

              Vector catalogo= new Vector();//Vector en el cual
              //guardamos los objetos de tipo producto ya filtrados y
              //que serán enviados a la página respuesta.jsp


              //Recibimos la variable origen
              origen = (String) request.getParameter("origen");

              //Objeto disk de tipo RequestDispatcher para enviar la
              //información a la página respuesta.jsp
              RequestDispatcher disp = null;


              //Ejecutamos un ciclo while donde evaluamos la variable
              //origen y dependiendo del caso seleccionamos del
              //vector productos los objetos Producto que satisfacen
              //la condición almacenándolos en el vector catálogo.

              if (origen.equals("1")){

                    for(int i = 0; i < productos.size(); i = i + 1){

                          p_aux=(Producto)productos.elementAt(i);

                          if(p_aux.RecuperaValor()<30000){

                                catalogo.addElement(p_aux);
                          }

                    }


              }
              else{

                    for(int i = 0; i < productos.size(); i = i + 1){

                          p_aux=(Producto)productos.elementAt(i);
```

```
                              if(p_aux.RecuperaValor()>=30000){

                                    catalogo.addElement(p_aux);
                              }

                        }

                  }


                  //Registramos en el objeto request la variable
                  //catálogo.
                  request.setAttribute("catalogo", catalogo);

                  //Indicamos la página a la cual enviaremos el objeto
                  //request

                  disp =
getServletContext().getRequestDispatcher("/respuesta.jsp");


                  //Despachamos el resultado.
                  disp.forward(request, response);

      }

      public void doGet(
            javax.servlet.http.HttpServletRequest request,
            javax.servlet.http.HttpServletResponse response)
            throws javax.servlet.ServletException, java.io.IOException {

                  this.doPost(request, response);
      }

      //Método Conectar, privado y que retorna un objeto de tipo
      //Connection
      private Connection Conectar() {

            //Variable de conexión a al base de datos
            Connection conexion = null;



            //String que contiene la ruta a la base de datos, en nuestro
            //caso la base de datos es EJ72J, si se llamase BASE sería
            //jdbc:db2:EJ72J
            String DBurl = "jdbc:db2:EJ72J";


            //Ejecutamos un try-catch que no permite conectarnos a la
            //base de datos

            try {
                  //Llamamos el driver de conexión (estándar para
                  //cualquier base de datos DB2)
                  Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");

//Establecemos la conexión a la base de datos señalando el username y
password. DriverManager.getConnection(DBurl, “username”, “password”. El
username y password son los valores con que fue instalado el db2.
conexion = DriverManager.getConnection(DBurl, "db2admin", "db2admin");

            } catch (Exception e) {
```

```
                e.printStackTrace();
            }
            //Retornamos la conexión.
            return conexion;
        }
}
```

***Resumen:*** **Contamos con una clase producto que define una estructura para manejar la información de cada producto. Además implementamos un servlet que recibe la variable origen del cliente la cual permite discriminar que tipos de productos se requiere desplegar en pantalla según el valor de estos. Este servlet se conecta a la base de datos EJ72J, rescata toda la información de la tabla productos y luego selecciona aquellos registros que cumplen con el criterio expresado por el cliente a través de la variable origen. La salida del servlet es un vector llamado catálogo el cual contiene los objetos producto, con la información de estos, que satisfacen el criterio de búsqueda. Corresponde entonces crear la página de resultados**

Una vez construido el servlet, solo nos falta construir la página de respuesta en la cual se desplegarán los resultados. Esta página deberá considerar lo siguiente:

- Importar la clase Producto ya que trabajaremos con un vector de objetos producto los cuales debemos ser capaces de manejar. En particular necesitaremos rescatar la variable id, nombre y valor de cada objeto, para ello emplearemos los métodos getters de la clase.
- Usar un bean para traer el vector catálogo.
- Definir el encabezado de una tabla en html para desplegar los contenidos del vector catalogo. Para ello usaremos un ciclo for de cero al largo del vector y que en cada "vuelta" creará una fila de la tabla html e insertará los valores rescatados (id, nombre y valor) del objeto i-ésimo del vector catálogo.

Creamos el documento *respuesta.jsp* y al igual que en el ejemplo 2 insertamos un bean para recibir ahora la variable catálogo. En este caso, dado que catálogo es de tipo vector, en el campo clase colocamos *java.util.Vector*.

Finalmente escribimos el código que implementa la página el cual se presenta a continuación:

```
//Importamos la clase producto
<%@ page import="Producto" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>respuesta2.jsp</TITLE>
</HEAD>
<BODY background="b002bcg.gif">
<P><B><FONT size="+2">CATÁLOGO DE PRODUCTOS</FONT></B><BR>
<BR>
<jsp:useBean id="catalogo" class="java.util.Vector" scope="request"/>
<BR>
<BR>
</P>
//Definimos el encabezado de la tabla
<TABLE border="1">
    <TBODY>
        <TR>
            <TD><B>CÓDIGO</B></TD>
            <TD><B>PRODUCTO</B></TD>
            <TD><B>VALOR</B></TD>
        </TR>

      //Iniciamos un ciclo for de cero hasta el tamaño del vector
      //(catalogo.size()) menos 1. De esta manera recorremos todos los
      //índices del vector
        <%
            Producto producto;

            for(int i=0; i<catalogo.size();i=i+1){

            //En cada ciclo recuperamos el i-ésimo elemento del vector
            //catálogo y lo almacenamos en el objeto producto que es de
            //tipo producto
                  producto=(Producto)catalogo.elementAt(i);

            %>

            //En cada ciclo agregamos una fila a la tabla y por medio de
            //los métodos de la clase producto recuperamos el id, nombre
            //y valor y lo insertamos en la respectiva columna de la
            tabla
        <TR>
            <TD><%=producto.RecuperaIdentificador()%></TD>
            <TD><%=producto.RecuperaNombre()%></TD>
            <TD><%=producto.RecuperaValor()%></TD>
        </TR>


        <%}%> //Finalizamos el ciclo for
    </TBODY>
</TABLE>//Cerramos la tabla
</BODY>
</HTML>
```