FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

CURSOS INSTITUCIONALES

DESARROLLO DE COMPONENTES WEB CON JAVA

Del 17 al 25 de Septiembre de 2007

APUNTES GENERALES

CI - 187

Instructor: Ing. Alejandro Velázquez Mena
FONHAPO
Septiembre de 2007

Palacio de Minería, Calle de Tacuba No. 5, Primer piso, Delegación Cuauhtémoc, CP 06000, Centro Histórico, México D.F., APDO Postal M-2285 • Tels: 5521 4021 al 24, 5623 2910 y 5623 2971 • Fax: 5510.0573

Índice

1. JDBC 4
1.1 Introducción 4
1.2 Conceptos básicos 4
1.2.2 ¿Qué es JDBC? 4
1.2.3 ¿Qué hace JDBC? 4
1.2.4 JDBC frente a ODBC y otros APIs 5
1.3 Conectividad a base de datos con aplicaciones standalone 6
1.3.1 Drivers JDBC 6
1.3.1 Pasos para usar JDBC 6
1.3.2 Envío de Sentencias SQL 9
2. Servlets 16
2.1 Protocolo http 16
2.2 El Ciclo de Vida de un Servlet 18
2.2.1 Cargando e instanciando un servlet 19
2.2.2 Inicializar un Servlet 19
2.2.3 Interactuar con Clientes 19
2.2.4 Destruir un Servlet 20
2.2.5 Descargando el Servlet 20
2.3 Servlet API 20
2.3.1 Paquete javax.servlet 20
2.3.2 Paquete javax.servlet.http 21
2.3.3 Peticiones y respuestas. 21
2.4 Interface ServletConfig 23
2.4.1 Métodos de ServletConfig 23
2.5 Interfaz ServletContext 25
2.6 Interfaz RequestDispatcher 27
2.7 Descriptor de aplicaciones 29
2.7.1 Elemento <servlet> 29
2.7.2 Elemento <servlet-mapping> 30
2.8 Manejo de Sesiones 30
2.8.1 Obtener una Sesión 31
2.8.2 Almacenar y Obtener Datos desde la Sesión 31
2.8.3 Invalidar una sesión 32
3. Java Server Page (JSP) 32

3.1 Sintaxis de los JSPs 32
3.1.1 Directivas. 33
3.1.2 Declaraciones. 33
3.1.3 Scriptlets. 34
3.1.4 Expresiones. 34
3.1.5 Acciones. 35
3.1.6 Comentarios. 35
3.2 Atributos de la directiva. 35
3.3 Usando Scriptlets. 36
3.4 Objetos implícitos 38
3.5 Alcances de los JSPs. 38
3.5.1 Aplication 39
3.5.2 Session 39
3.5.3 Request 39
3.5.4 Page 39
4. JavaBeans 39
4.1 Utilización de JavaBeans en JSP's 39
4.2 Usando JavaBeans y JSPs. 41
4.2.1 Declarando JavaBeans usando <jsp:useBean> 41
4.2.2 Inicializando las propiedades del bean. 42
4.2.3 Uso de <jsp:setProperty> 42
4.2.3 Uso de <jsp:getProperty>. 44
Anexo C: Repaso de HTML 45

1. JDBC

1.1 Introducción

Java Database Connectivity (JDBC) es una interfase de acceso a bases de datos estándar SQL que proporciona un acceso uniforme a una gran variedad de bases de datos relacionales. JDBC también proporciona una base común para la construcción de herramientas y utilidades de alto nivel.

El paquete actual de JDK incluye JDBC y el puente JDBC-ODBC.

1.2 Conceptos básicos

1.2.2 ¿Qué es JDBC?

JDBC es el API para la ejecución de sentencias SQL. (Como punto de interés JDBC es una marca registrada y no un acrónimo, no obstante a menudo es conocido como "Java Database Connectivity"). Consiste en un conjunto de clases e interfases escritas en el lenguaje de programación Java. JDBC suministra un API estándar para los desarrolladores y hace posible escribir aplicaciones de base de datos usando un API puro Java.

Usando JDBC es fácil enviar sentencias SQL virtualmente a cualquier sistema de base de datos. En otras palabras, con el API JDBC, no es necesario escribir un programa que acceda a una base de datos Sybase, otro para acceder a Oracle y otro para acceder a Informix. Un único programa escrito usando el API JDBC y el programa será capaz de enviar sentencias SQL a la base de datos apropiada. Y, con una aplicación escrita en el lenguaje de programación Java, tampoco es necesario escribir diferentes aplicaciones para ejecutar en diferentes plataformas. La combinación de Java y JDBC permite al programador escribir una sola vez y ejecutarlo en cualquier entorno.

JDBC expande las posibilidades de Java. Por ejemplo, con Java y JDBC, es posible publicar una página web que contenga un applet que usa información obtenida de una base de datos remota. O una empresa puede usar JDBC para conectar a todos sus empleados (incluso si usan un conglomerado de máquinas Windows, Macintosh y UNIX) a una base de datos interna vía intranet.

1.2.3 ¿Qué hace JDBC?

Simplemente JDBC hace posible estas tres cosas:

- Establece una conexión con la base de datos.
- Envía sentencias SQL
- Procesa los resultados.

El siguiente fragmento de código nos muestra un ejemplo básico de estas tres cosas:

```
Connection con = DriverManager.getConnection (
            "jdbc:odbc:wombat", "login", "password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next()) {
  int x = rs.getInt("a");
  String s = rs.getString("b");
  float f = rs.getFloat("c");
}
```

1.2.4 JDBC frente a ODBC y otros APIs

En este punto, el ODBC de Microsoft (Open Database Connectvity), es probablemente el API más extendido para el acceso a bases de datos relacionales. Ofrece la posibilidad de conectar a la mayoría de las bases de datos en casi todas las plataformas. ¿Por qué no usar, entonces, ODBC, desde Java?

La respuesta es que se puede usar ODBC desde Java, pero es preferible hacerlo con la ayuda de JDBC mediante el puente JDBC-ODBC. La pregunta es ahora ¿por qué necesito JDBC? Hay varias respuestas a estas preguntas:

1.- ODBC no es apropiado para su uso directo con Java porque usa una interface C. Las llamadas desde Java a código nativo C tienen un número de inconvenientes en la seguridad, la implementación, la robustez y en la portabilidad automática de las aplicaciones.

2.- Una traducción literal del API C de ODBC en el API Java podría no ser deseable. Por ejemplo, Java no tiene punteros, y ODBC hace un uso copioso de ellos, incluyendo el notoriamente propenso a errores "void * ".. Se puede pensar en JDBC como un ODBC traducido a una interfase orientada a objeto que es el natural para programadores Java.

3. ODBC es difícil de aprender. Mezcla características simples y avanzadas juntas, y sus opciones son complejas para 'querys' simples. JDBC por otro lado, ha sido diseñado para mantener las cosas sencillas mientras que permite las características avanzadas cuando éstas son necesarias.

4. Un API Java como JDBC es necesario en orden a permitir una solución Java "puro". Cuando se usa ODBC, el gestor de drivers de ODBC y los drivers deben instalarse manualmente en cada máquina cliente. Como el driver JDBC esta completamente escrito en Java, el código JDBC es automáticamente instalable, portable y seguro en todas las plataformas Java.

En resumen, el API JDBC es la interfase natural de Java para las abstracciones y conceptos básicos de SQL. JDBC retiene las características básicas de diseño de ODBC.

1.3 Conectividad a base de datos con aplicaciones standalone

Gracias al API de JDBC tenemos la oportunidad de poder hacer conexión con la base de datos y ocupar cualquier componente que ofrece la plataforma Java, ya sea un Servlet, un JSP, una GUI o simplemente usando una aplicación standalone como se verá a continuación.

1.3.1 Drivers JDBC

Para usar JDBC con un sistema gestor de base de datos en particular, es necesario disponer del driver JDBC apropiado que haga de intermediario entre ésta y JDBC. Dependiendo de varios factores, este driver puede estar escrito en Java puro, o ser una mezcla de Java y métodos nativos JNI (Java Native Interface).

Para cargar o registrar el driver, simplemente se siguen los siguientes pasos:

1. Se agrega a la variable CLASSPATH el archivo que contiene las clases del driver (con su ruta de directorios). Una alternativa a este paso es agregar el archivo JAR al directorio `jre\lib\ext` que se encuentra en el directorio raíz del JDK Standard Edition.

2. Se hace uso del método estático `forName()` de la clase `Class` del paquete `java.lang`:

   ```
   Class.forName("nombre_de_la_clase_del_driver");
   ```

   Por ejemplo, para cargar el driver propio de Sybase es:

   ```
   Class.forName("com.sybase.jdbc2.jdbc.SybDriver");
   ```

   y para cargar el driver de Oracle es:

   ```
   Class.forName("oracle.jdbc.driver.OracleDriver");
   ```

Y así como se ejemplificó para estos dos manejadores de bases de datos, así se realiza para los demás. Posteriormente, ya en el código, se requiere realizar la conexión a la base de datos.

1.3.1 Pasos para usar JDBC

A continuación se muestran los pasos que se tienen que seguir para hacer la conexión a una base de datos de terminada.

1) Crear una instancia del JDBC driver.

```
Class.forName("paquete.driver.nombreDriver");
```

2) Especificar la *url* de la base de datos.

```
String url = "jdbc:subprotocolo:subname"
```

- subprotocolo es un nombre corto que identifica al driver.
- subname es dependiente del DBMS, normalmente contiene el nombre del servidor y el nombre de la base de datos cuando menos.

3) Establecer una conexión usando el driver que crea el objeto *Connection*.

Para crear y establecer una conexión a una base de datos desde JDBC, se requiere utilizar el método estático `getConnection()` de la clase `DriverManager` del paquete `java.sql`:

```
Connection con =
 DriverManager.getConnection(String url, String user, String
password);
```

donde el primer argumento es un String que le indica a JDBC el sistema en donde reside la base de datos. La sintaxis estándar del URL de JDBC es de la siguiente manera:

```
jdbc:subprotocolo:servidor.dominio:puerto{/|:}base_de_datos
```

- El método estático `getConnecion` usa el valor de la url como argumento.
- Si se establece la conexión sin problema, regresa un objeto de clase `Connection`.
- Si hay problemas, se genera una `SQLException`.
- El objeto clase `Connection` representa una sesión con una base de datos específica.

Por ejemplo, si se tiene una base de datos Sybase llamada `pubs` en el sistema `odin.fi-b.unam.mx` en el puerto `1521` y cuyo usuario tiene el login `dba` y el password `sql`, entonces la sentencia de conexión es:

```
con = DriverManager.getConnection(
"jdbc:sybase:Tds:odin.fi-b.unam.mx:1521/pubs", "dba","sql");
```

4) Crear un objeto *Statement,* usando *Connection.*

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

• El método `createStatement` regresa un objeto de clase `Statement`.
• Si hay problemas, se genera una `SQLException`.
• El objeto clase `Statement` es usado para enviar postulados SQL a la Base de Datos.

5) Armar el postulado SQL y enviarlo a ejecución usando el *Statement.*

```
ResultSet rs = stmt.executeQuery("postulado Select SQL");
int num = stmst.ExecuteUpdate("otro postulado SQL");
```

• El método `executeQuery` regresa un objeto de clase `ResultSet` que contiene los resultados del query SQL.
• El objeto clase `ResultSet` es usado para interpretar el resultado del query de SQL.
• El método `executeUpdate` regresa un entero que contiene el número de renglones de la tabla relacional afectados por "otro postulado SQL" que puede ser UPDATE, INSERT, DELETE u otro postulado de SQL.
• Si hay problemas, se genera una `SQLException`.

6) Recibir los resultados en el objeto *ResultSet.*

```
while(rs.next()) {
      System.out.println("Columna 1: " + rs.getString(1));
      System.out.println("Columna 2: " + rs.getString(2));
}
```

• Los resultados del Query se almacenan en un conjunto de renglones en el objeto `ResultSet`.
• El objeto `ResultSet` inicialmente apunta a la primera fila.
• El método `next` de `ResultSet` mueve a la siguiente fila.
• Los métodos getXXX de ResultSet permiten acceso a las columnas de la fila apuntada.

Aquí se presentan los requisitos correspondientes para algunos manejadores:

- **DB2**
  - Clase Driver : COM.ibm.db2.jdbc.app.DB2Driver
  - URL de conexión:: jdbc:db2:<database>
  - Archivo .jar/.zip: db2java.zip
- **Sybase**
  - Clase Driver : com.sybase.jdbc2.jdbc.SybDriver
  - URL de conexión: jdbc:sybase:Tds:<host>:<port>/<database>
  - Archivo .jar/.zip: jconn2.jar
- **Oracle**
  - Clase Driver : oracle.jdbc.driver.OracleDriver
  - URL de conexión: jdbc:oracle:thin:@ <host>:<port>:<sid>
  - Archivo .jar/.zip: classes12.zip

- **SQLServer**
  - Clase Driver : com.microsoft.jdbc.sqlserver.SQLServerDriver
  - URL de conexión: jdbc:microsoft:sqlserver://localhost:1433
  - Archivo .jar/.zip: mssqlserver.jar, msbase.jar, msutil.jar
- **PostgreSQL**
  - Clase Driver: org.postgresql.Driver
  - URL de conexión: jdbc:postgresql://<server>:<port>/<database>
  - Archivo .jar/.zip: postgresql.jar

1.3.2 Envío de Sentencias SQL

Una vez que la conexión se haya establecido, se usa para pasar sentencias SQL a la base de datos subyacente. JDBC no pone ninguna restricción sobre los tipos de sentencias que pueden enviarse, esto da un alto grado de flexibilidad, permitiendo el uso de sentencias específicas de la base de datos o incluso sentencias no SQL. Se requiere de cualquier modo, que el usuario sea responsable de asegurarse que la base de datos subyacente sea capaz de procesar las sentencias SQL que le están siendo enviadas y soportar las consecuencias si no es así. Por ejemplo, una aplicación que intenta enviar una llamada a un procedimiento almacenado a una DBMS (sistema manejador de bases de datos) que no soporta procedimientos almacenados no tendrá éxito y generará una excepción. JDBC requiere que un driver cumpla al menos ANSI SQL-2 Entry Level para ser designado JDBC COMPLIANT. Esto significa que los usuarios pueden contar al menos con este nivel de funcionalidad.

JDBC suministra tres clases para el envío de sentencias SQL y tres métodos en la interfaz `Connection` para crear instancias de estas tres clases. Estas clases y métodos son los siguientes:

1.- **Statement** - El objeto de tipo `Statement` se utiliza para crear sentencias SQL estáticas y regresar el resultado que éstas producen. Por lo general, el objeto `Statement` se utiliza para la ejecución de sentencias SELECT y en ese caso, se invoca al método `executeQuery()` el cual regresa un objeto `ResultSet` que contiene los registros resultantes de la consulta:

```
Statement stm = con.createStatement();
ResultSet res =
      stm.executeQuery("select cve_emp,nom_emp from emp");
```

También se utiliza para realizar sentencias INSERT, UPDATE y DELETE y en ese caso, se utiliza el método `executeUpdate()` llevando como argumento el `String` con la sentencia SQL:

```
Statement stm = con.createStatement();
int res = stm.executeUpdate("insert into emp
                              values('123','Nina Anderson')");
```

2.- **PreparedStatement** - El objeto de tipo `PreparedStatement` se utiliza para ejecutar sentencias SQL precompiladas (o dinámicas en cuanto a los valores de los parámetros en la condición de la sentencia SQL). Por lo general, `PreparedStatement` se utiliza para la ejecución de sentencias INSERT, DELETE y UPDATE ya que éstas últimas requieren condicionantes por lo que se crea el `String` con la sentencia SQL desde la obtención del objeto `PreparedStatement`. Para establecer el valor de una condicionante, se utilizan los diversos métodos `setXXX()` donde `XXX` representa a cada uno de los diversos tipos de datos que existen en Java (`int`, `byte`, `float`, `double`, `String`, etc) y posteriormente se invoca al método `executeUpdate()` el cual regresa un valor `int` que indica el número de registros afectados por la sentencia (ya sea INSERT, UPDATE o DELETE). Por ejemplo, utilizando el objeto `PreparedStatement`:

```
PreparedStatement pstmt =
   con.prepareStatement("update emp
                          set nom_emp=?
                          where cve_emp=?)");
pstmt.setString(1,"Nina Anderson");
pstmt.setInt(2,123);
int res = pstmt.executeUpdate();
```

Se observa que JDBC sustituye los signos de interrogación (?) por los valores según el orden establecido por el primer argumento del método `setXXX()`. También existe la posibilidad de crear una sentencia SELECT con argumentos asignados dinámicamente así que `PreparedStatement` también tiene el método `executeQuery()` que regresa un objeto `ResultSet`, con los registros resultantes:

```
PreparedStatement pstmt =
   con.prepareStatement("select cve_emp,nom_emp from emp
                          where cve_emp=?");
pstmt.setInt(1,123);
ResultSet res = pstmt.executeQuery();
```

3.- **`CallableStatement`** – El objeto de tipo `CallableStatement` se utiliza para ejecutar procedimientos almacenados (*stored procedures*) SQL. La API de JDBC provee una sintaxis para que todos los procedimientos almacenados sean llamados de la misma manera en los diferentes sistemas manejadores de bases de datos. Dicha sintaxis provee una manera que incluye un parámetro de resultado y otra manera que no incluye dicho parámetro. Si el parámetro es utilizado, debe ser registrado como un parámetro OUT. Los parámetros restantes pueden ser utilizados para entrada, salida o ambos. Los parámetros son referenciados a través de un número secuencial, empezando por 1. Las sintaxis son:

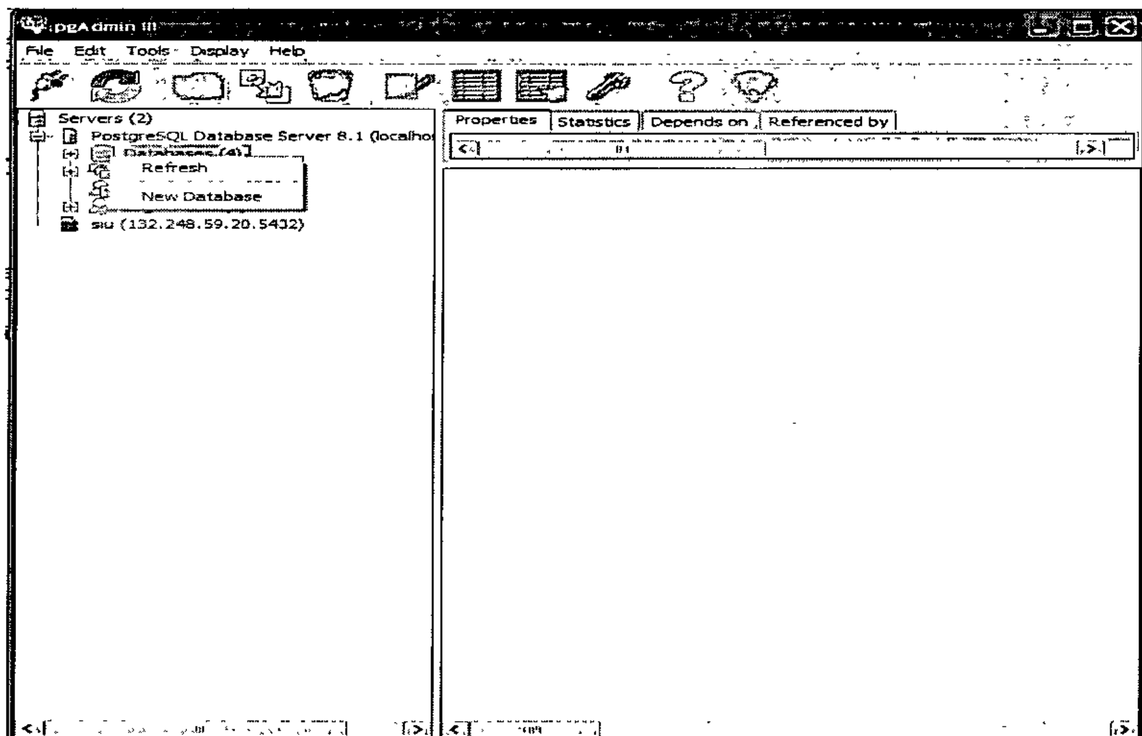
```
{?= call <nombre_del_procedimiento_almacenado>[<arg1>,<arg2>, ...]}
{call < nombre_del_procedimiento_almacenado >[<arg1>,<arg2>, ...]}
```

Los valores de los parámetros IN (de entrada) están establecidos por los métodos `setXXX()` heredados de `PreparedStatement`. Los tipos de todos los parámetros OUT (de salida) deben ser registrados ejecutando el procedimiento almacenado, sus valores son recuperados después de la ejecución a través de los métodos `getXXX()` que provee `CallableStatement`.

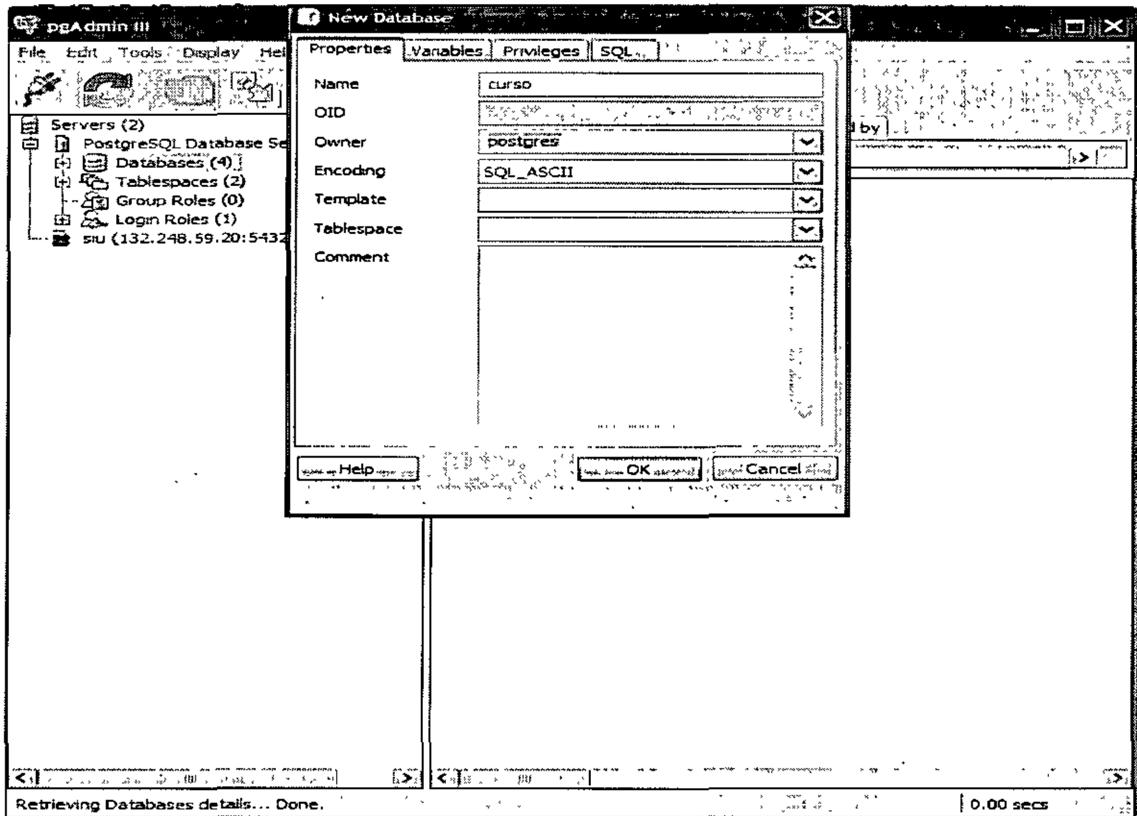
Un objeto `CallableStatement` también puede regresar un objeto `ResultSet` o múltiples objetos `ResultSet`. En este último caso, los múltiples objetos `ResultSet` son manejados por métodos heredados de `Statement`.

A continuación se mostrarán los pasos para crear una base de datos en PostgresSQL.

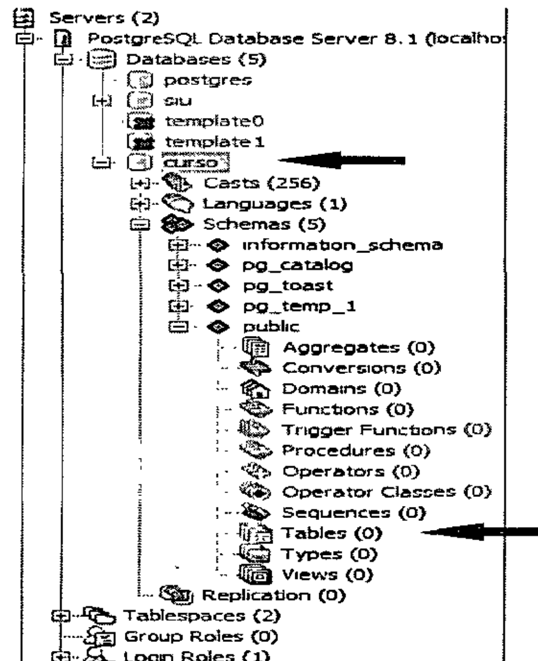
1. Se abre el programa pgAdmin III y se establece la conexión con la base de datos y en la opción de Databases se da clic con el botón derecho y se escoge "New Database".



2. A continuación saldrá un recuadro donde se llenarán los campos tal y como se muestra en la figura y se dará aceptar.

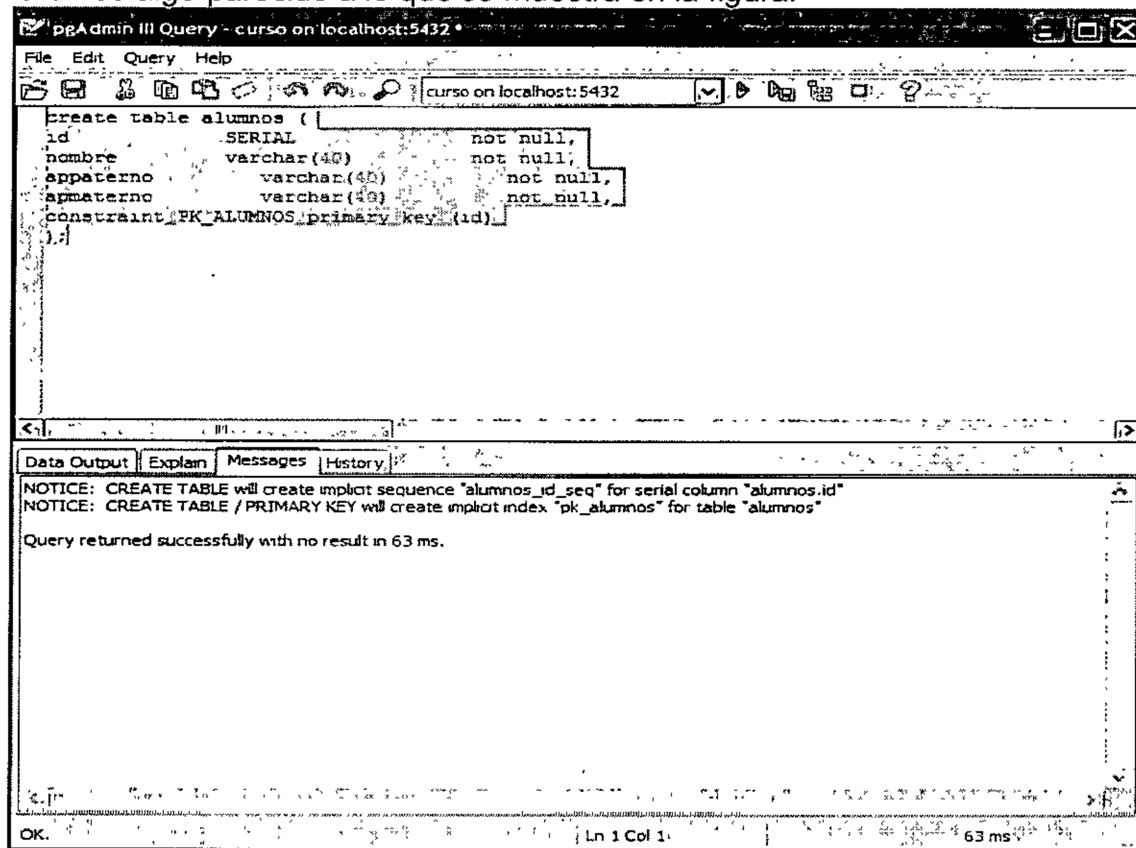


3. A continuación crearemos una tabla dentro de nuestra base de datos, para eso expandemos el árbol de la opción Databases y veremos una base de datos con el nombre "curso".



Después damos clic sobre el botón y nos abrirá otra ventana donde podemos ejecutar sentencias de SQL. A continuación ingresamos el siguiente código para crear

nuestra primera tabla y apretamos el botón para ejecutar nuestra sentencia y obtendremos algo parecido a lo que se muestra en la figura.



Ya que tenemos creada nuestra base de datos, ya podemos crear código Java para poder realizar la conexión a la base, pero antes de todo hay que verificar que se tengan agregado el `Driver` en la variable CLASSPATH. En éste caso como se va a utilizar la base de datos PostgresSQL, el driver a utilizar es "pg73jdbc3.jar" o la versión más actual con que se cuente. Los drivers más recientes de postgres se pueden bajar de http://jdbc.postgresql.org/download.html.

A continuación se muestra el código DBConector.java para hacer una inserción en la base de datos.

```
import java.sql.*;
import java.io.*;


public class DBConector
{

        private Connection conn;
        private Statement sqlStatement;
        private ResultSet rs;
        private ResultSetMetaData rsmd;
        private Statement stm = null;
        private static String sql=null;
        private int rowsAffected = 0;
```

```
        public DBConector(String className,String ip,String base,String
susuario,String spass)
        {
                try
                {
                    Class.forName(className);
                    conn =
DriverManager.getConnection(ip+base+"?user="+susuario+"&password="+spass);
                }
                catch(Exception e)
                {
                    System.out.println("Error:"+e.getMessage());
                }
        }

        public void close()
        {
                try{
                    conn.close();
                }catch(SQLException e){}
        }


        public int hacerInsert(String sql) throws SQLException
      {
            System.out.println(sql);
              try
                {
                        stm = conn.createStatement();
                  rowsAffected = stm.executeUpdate(sql);
            }
                catch(Exception e)
                {
                        System.out.println("--------------------------------------
--");

                        System.out.println("EXCEPTION GENERADA");
                        System.out.println(e.toString());
                        System.out.println("--------------------------------------
--");

                }
            return rowsAffected;
       }



        public static void main(String args[])
        {
                try
                {
                    sql = "insert into alumnos values(DEFAULT,\'edgar\',\'garcia
cano\',\'castillo\')";
                    DBConector c = new
DBConector("org.postgresql.Driver","jdbc:postgresql://127.0.0.1:5432/","curso","
postgres","hola123");

                    System.out.println("Resultdo de la inserción"+
c.hacerInsert(sql));
                    c.close();
                }
```

```
                catch(Exception e)
                {
                   System.out.println("Error:"+e);
                }
        }
}
```

A continuación se muestra el método para realizar selecciones sobre una tabla de la base de datos.

```
    public void hacerSelect(String sql) throws SQLException
        {

            try
            {

               sqlStatement = conn.createStatement();
               rs = sqlStatement.executeQuery(sql);

               while(rs.next())
               {
                     System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3)+" "+rs.getString(4));

               }

            }
            catch(SQLException e)
            {
                System.out.println("Error en select : "+e);
            }
            catch(NullPointerException e)
            {
                System.out.println("Error en select : "+e);
            }

        }
```

Para mandar llamar éste método en el main se escribiría lo siguiente:

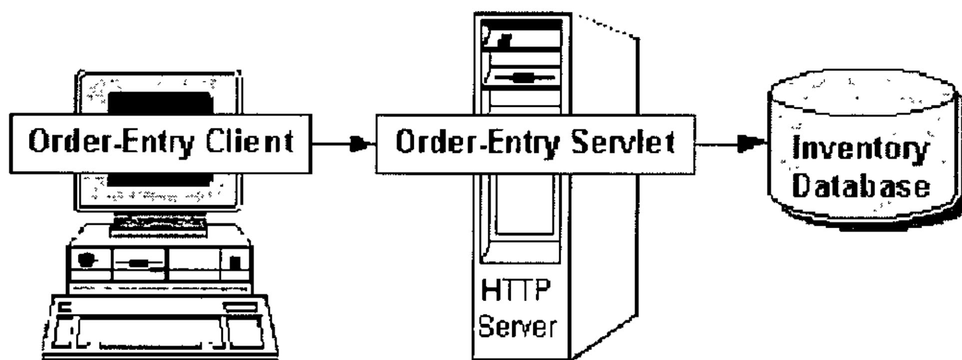
```
    c.hacerSelect("select * from alumnos");
```

2. Servlets

La tecnología de Servlets de Java es comúnmente usada para manejar la lógica de negocio de una aplicación web. Los Servlets pueden ser aplicados en cualquier protocolo, pero en la práctica, los Servlets son escritos para ser utilizados para el protocolo HTTP.

Los Servlets son módulos que extienden los servidores orientados a petición-respuesta, como los servidores web compatibles con Java. Por ejemplo, un Servlet podría ser responsable de tomar los datos de un formulario de entrada de pedidos en HTML y aplicarle la lógica de negocios utilizada para actualizar la base de datos de pedidos de la compañía.

Los paquetes de clases javax.servlet y javax.servlet.http proveen interfaces y clases que nos van a servir para escribir Servlets. Todos los Servlets deben implementar la interface Servlet, la cual define los métodos del ciclo de vida de los Servlets.



2.1 Protocolo http

El **protocolo de transferencia de hipertexto** (***HTTP***, *HyperText Transfer Protocol*) es el protocolo usado en cada transacción de la Web. El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido. También sirve el protocolo para enviar 'información adicional en ambos sentidos, como formularios con campos de texto.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. Al finalizar la transacción todos los datos se pierden.

Un mensaje mandado por un cliente a un servidor es llamado HTTP request. La línea inicial de un HTTP request tiene tres partes separadas por espacios.

- Nombre del método.
- La ruta local.
- La versión de http.
-

Una típica línea es la siguiente:

```
GET /reports/sales/index.html HTTP/1.1
```

El nombre del método especifica la acción que el cliente quiere que el servidor ejecute. A continuación se enuncian los principales métodos.

GET

Éste método es usado para traer un recurso. Si se necesita algún parámetro, son pegados al URI.

HEAD

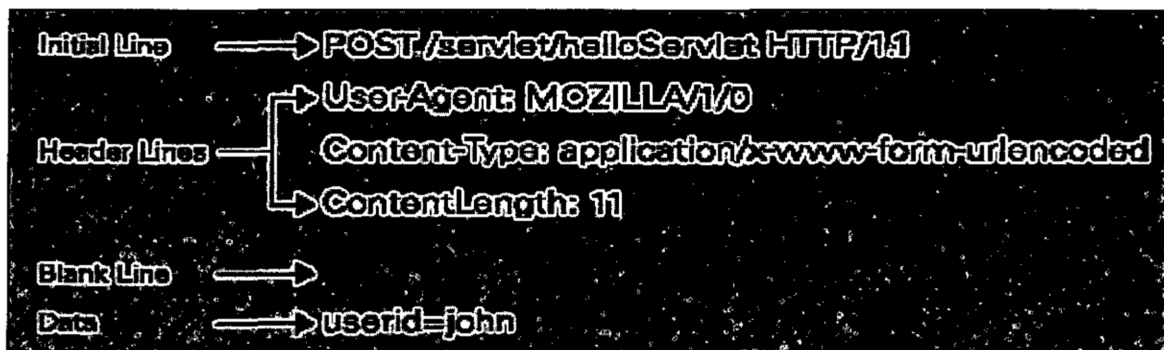
El método HEAD es igual que el método GET, salvo que el servidor no tiene que devolver el contenido, sólo las cabeceras. Estas cabeceras que se devuelven en el método HEAD deberían ser las mismas que las que se devolverían si fuese una petición GET.

Este método se puede usar para obtener información sobre el contenido que se va a devolver en respuesta a la petición. Se suele usar también para chequear la validez de *links*, accesibilidad y modificaciones recientes.

POST

El método POST se usa para hacer peticiones en las que el servidor destino acepta el contenido de la petición como un nuevo subordinado del recurso pedido. El método POST se creó para cubrir funciones como la de enviar un mensaje a grupos de usuarios, dar un bloque de datos como resultado de un formulario a un proceso de datos, añadir nuevos datos a una base de datos.

La función llevada a cabo por el método POST está determinada por el servidor y suele depender de la URI de la petición. El resultado de la acción realizada por el método POST puede ser un recurso que no sea identificable mediante una URI.



PUT

El método PUT permite guardar el contenido de la petición en el servidor bajo la URI de la petición. Si esta URI ya existe, entonces el servidor considera que esta petición proporciona una versión actualizada del recurso. Si la URI indicada no existe y es válida para definir un nuevo recurso, el servidor puede crear el recurso con esa URI. Si se crea un nuevo recurso, debe responder con un código 201 (creado), si se modifica se contesta con un código 200 (OK) o 204 (sin contenido). En caso de que no se pueda crear el recurso se devuelve un mensaje con el código de error apropiado.

La principal diferencia entre POST y PUT se encuentra en el significado de la URI. En el caso del método POST, la URI identifica el recurso que va a manejar en contenido, mientras que en el PUT identifica el contenido.

En general un navegador es un cliente HTTP y los recursos son páginas web, imágenes, Servlets y más. Cada recurso es identificado por un único URI (Uniform Resource Identifier o Identificador Uniforme de Recursos), se pueden encontrar 3 términos que pueden ser intercambiables entre sí: URI,URL y URN. Aunque son similares, tienen ciertas diferencias.

URI: Es una cadena que identifica cualquier recurso. Identificar un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc), no indica que podamos traerlo, éste término es más general que URL y URN.

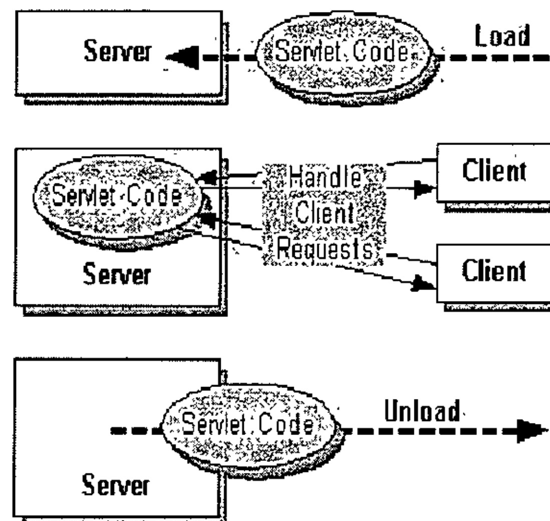
URL: (Uniform Resource Locator o Localizador Uniforme de Recurso) estas son URIs que especifican protocolos comunes como HTTP, FTP y malito. Usado para nombrar recursos, como documentos e imágenes en Internet, por su localización.

URN: (Uniform Resource Name o Nombre Uniforme de Recurso) es un identificador que únicamente identifica un recurso, pero no especifica como accesar a él. URNs son estandarizadas para instituciones oficiales para mantener un único recurso.

2.2 El Ciclo de Vida de un Servlet

Cada servlet tiene el mismo ciclo de vida.

- Un servidor carga e inicializa el servlet.
- El servlet maneja cero o más peticiones de cliente.
- El servidor elimina el servlet. (Algunos servidores sólo cumplen este paso cuando se desconectan).



2.2.1 Cargando e instanciando un servlet

Cuando el contenedor de Servlets se inicia, busca los archivos de configuración de las aplicaciones, también llamados descriptores. Cada aplicación web tiene su propio descriptor llamado `web.xml`, que incluye una descripción de cada Servlet que compone la aplicación. El contenedor de Servlets crea una instancia del Servlet usando `Class.forName(className).newInstance()`. Sin embargo, para hacer lo anterior el Servlet debe de tener un constructor público y sin argumentos, generalmente no se define ningún constructor en la clase.

2.2.2 Inicializar un Servlet

Cuando un servidor carga un Servlet, ejecuta el método `init(ServletConfig)` del Servlet. La inicialización se completa antes de manejar peticiones de clientes y antes de que el Servlet sea destruido. El objeto ServletConfig contiene todos los parámetros de inicialización que se especifican en el descriptor de la aplicación.
Aunque muchos Servlets se ejecutan en servidores multithread, los Servlets no tienen problemas de concurrencia durante su inicialización. El servidor llama sólo una vez al método `init()`, cuando carga el Servlet, y no lo llamará de nuevo a menos que vuelva a recargar el servlet. El servidor no puede recargar un Servlet sin primero haber destruido el servlet llamando al método `destroy()`.

2.2.3 Interactuar con Clientes

Después de la inicialización, el Servlet puede manejar peticiones de clientes. Esta parte del ciclo de vida de un Servlet se pudo ver en la sección anterior. Después de que la instancia es propiamente inicializada, el Servlet esta listo para dar servicio. Cuando el contenedor recibe una petición para el Servlet, éste la despachará llamando a `Servlet.service(ServletRequest,ServletResponse)`.

2.2.4 Destruir un Servlet

Los Servlets se ejecutan hasta que el servidor los destruye, por ejemplo, a petición del administrador del sistema. Cuando un servidor destruye un Servlet, ejecuta el método `destroy()` del propio Servlet. Este método sólo se ejecuta una vez. El servidor no ejecutará de nuevo el Servlet, hasta haberlo cargado e inicializado de nuevo.

2.2.5 Descargando el Servlet

Una vez destrúido, la instancia puede ser recogida por el recolector de basura. Si el Servlet ha sido destruido porque el Servicio del contenedor se paro, el Servlet también es descargado.

2.3 Servlet API

La especificación de los Servlets de Sun proveen un estándar e independencia de la plataforma para poder realizar comunicación entre los Servlets y sus contenedores. Las grupo de clases e interfaces es llamado Servlet Application Programming Interfaces (Servlet API).

El Servlet API está dividido en dos paquetes: javax.servlet y javax.servlet.http.

2.3.1 Paquete javax.servlet

Este paquete contiene las interfaces y clases genéricas de los Servlets que son independientes de cualquier plataforma.

Interface javax.servlet.Servlet

Esta es la interface central en el API de los Servlets. Cada Servlet implementa directa o indirectamente esta interfaz y contiene los métodos:

- init()
- service()
- destroy()
- getServletConfig()
- getServletInfo()

Clase javax.servlet.GenericServlet

Es una clase abstracta que tiene implementación para todos los métodos excepto `service()`.

Interface javax.servlet.ServletRequest

Provee una vista genérica de la petición que fue enviada por un cliente. Define métodos para obtener la información de la petición.

Interface javax.servlet.ServletResponse

Provee una forma genérica de envíar respuestas. Posee métodos que permiten envíar respuestas al cliente.

2.3.2 Paquete javax.servlet.http

Este paquete provee la funcionalidad básica requerida para los HTTP Servlets. Las clases e interfaces de éste paquete tienen soporte para utilizar el protocolo HTTP.

Clase javax.servlet.http.HttpServlet

Es una clase abstracta que extiende de GenericServlet y asigna un método service con la siguiente firma.

```
protected void service(HttpServletRequest,HttpServletResponse) throws
ServletException, IOException
```

Interface javax.servlet.http.HttpServletRequest

Esta interface hereda de ServletRequest y prove formas específicas HTTP para las peticiones. Define métodos para obtener información como cabeceras o cookies en la petición.

Interface javax.servlet.http.HttpServletRequest

Esta interface hereda de `ServletResponse` y provee fomas específicas para de envío de respuestas HTTP. Define métodos para el envío de información como cabeceras y cookies en la respuesta.

2.3.3 Peticiones y respuestas.

Un Servlet HTTP maneja peticiones del cliente a través de su método `service`. Este método soporta peticiones estándar de cliente HTTP despachando cada petición a un método designado para manejar esa petición.

Los métodos de la clase `HttpServlet` que manejan peticiones de cliente toman dos argumentos.

- Un objeto `HttpServletRequest`, que encapsula los datos desde el cliente.
- Un objeto `HttpServletResponse`, que encapsula la respuesta hacia el cliente.

Objetos HttpServletRequest

Un objeto `HttpServletRequest` proporciona acceso a los datos de cabecera HTTP, como cualquier cookie encontrada en la petición, y el método HTTP con el que se ha realizado la petición. El objeto `HttpServletRequest` también permite obtener los argumentos que el cliente envía como parte de la petición.

Para acceder a los datos del cliente:

El método `getParameter` devuelve el valor de un parámetro nombrado. Si nuestro parámetro pudiera tener más de un valor, deberíamos utilizar `getParameterValues` en su lugar. El método `getParameterValues` devuelve un arreglo de valores del parámetro nombrado. (El método `getParameterNames` proporciona los nombres de los parámetros.

Objetos HttpServletResponse

Un objeto `HttpServletResponse` proporciona dos formas de devolver datos al usuario:

- El método `getWriter` devuelve un `Writer`.
- El método `getOutputStream` devuelve un `ServletOutputStream`.

Se utiliza el método `getWriter` para devolver datos en formato texto al usuario y el método `getOutputStream` para devolver datos binarios.

Usando PrintWriter.

Utilizando el método `getWriter()` regresa un objeto `java.io.PrintWriter` que es usado para envíar datos. `PrintWriter` es extensivamente usado por los Servlets para generar páginas HTML dinámicas.

Una vez que se ha analizado el request, el Servlet puede decidir si quiere redireccionar a algún recurso. Para realizqar éste trabajo `HttpServletResponse` provee el método `sendRedirect()` que se muestra a continuación.

```
if("companynews".equals(request.getParameter("news_category")))
{
//retrieve internal company news and generate
//the page dynamically
}
else
{
response.sendRedirect("http://www.cnn.com");
}
```

En el ejemplo se checa news_category para tomar la decisión de qué hacer. Lo que no se puede hacer con `sendRedirect()` es enviar un flujo de información ya que generará una `IllegalStateException`.

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
{
      PrintWriter pw = res.getWriter();
      pw.println("<html><body>Hello World!</body></html>");
      pw.flush();         <- Envia la respuesta
      res.sendRedirect("http://www.cnn.com"); <- Trata de
      direccionar
}
```

En el ejemplo anterior se esta forzando al contenedor de Servlets enviar una cabecera y luego se direcciona, lo que hace que mande una excepción.

2.4 Interface ServletConfig

Como ya se vió en el ciclo de vida del servlet, el contenedor de Servlets pasa un objeto de tipo `ServletConfig` en el método `init(ServletConfig)`, a continuación se detallará la manera en que trabja `ServletConfig`. Hay que notar que `ServletConfig` sólo tiene métodos para traer los parámetros, no para envíar.

2.4.1 Métodos de ServletConfig

Método	Descripción
`String getInitParameter(String paramName)`	Regresa sólo un valor asociado con el parámetro dado o null si no esta disponible.
`Enumeration getInitParameterNames()`	Regresa un Enumeration de Strings de todos los nombres de los parámetros.
`ServletContext getServletContext()`	Regresa el ServletContext de éste servidor.
`String getServletName()`	Regresa el nombre del servlet especificado en el archivo de configuración.

A continuación se muestra un ejemplo de uso de `ServletConfig`.

a.- Especificando el archivo web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>
      <servlet>
            <servlet-name>TestServlet</servlet-name>
            <servlet-class>TestServlet</servlet-class>
                  <init-param>
                        <param-name>driverclassname</param-name>
                        <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
                  </init-param>
                  <init-param>
                        <param-name>dburl</param-name>
                        <param-value>jdbc:odbc:MySQLODBC</param-value>
                  </init-param>
                  <init-param>
```

```
                    <param-name>username</param-name>
                    <param-value>testuser</param-value>
              </init-param>
              <init-param>
                    <param-name>password</param-name>
                    <param-value>test</param-value>
              </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
</web-app>
```

b. Código de TestServlet.java

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class TestServlet extends HttpServlet
{
      Connection dbConnection;

      public void init(){

      System.out.println(getServletName()+" : Initializing...");
      ServletConfig config = getServletConfig();
      String driverClassName = config.getInitParameter("driverclassname");
      String dbURL = config.getInitParameter("dburl");
      String username = config.getInitParameter("username");
      String password = config.getInitParameter("password");

      //Load the driver class
      Class.forName(driverClassName);

      //get a database connection
      dbConnection = DriverManager.getConnection(dbURL,username,password);
      System.out.println("Initialized.");
      }

      public void service(HttpServletRequest req,HttpServletResponse res)throws
      ServletException, java.io.IOException{

      //get the requested data from the database and
      //generate an HTML page.
      }

      public void destroy()
      {
            try
            {
                  dbConnection.close();
            }
            catch(Exception e)
            {
                  e.printStackTrace();
            }
      }
}
```

2.5 Interfaz ServletContext

Se puede pensar que la interface `ServletContext` es la ventana para que un Servlet vea su entorno. El Servlet puede usar esta interface para obtener información como la inicialización de parámetros. Cada aplicación web tiene un y sólo un `ServletContext`. El contexto es inicializado al mismo tiempo que la aplicación es cargada.

La información del contexto del servidor web está disponible en cualquier momento a través de un objeto `ServletContext`. Un Servlet puede obtener este objeto llamando al método `getServletContext()` del objeto `ServletConfig` y hay que recordar que este objeto se pasa al servlet en el método `init()`.

La interface `ServletContext` define varios métodos, como se puede ver a continuación:

Método	Descripción
`getAttribute()`	Obtiene información acerca del servidor en pares de atributos nombre/valor.
`getMimeType`	Regresa el tipo MIME de un archivo dado.
`getRealPath()`	Este método convierte una ruta relativa o virtual a una nueva ruta en relación con el directorio raíz de los documentos HTML.
`getServerInfo`	Regresa el nombre y la versión de los servicios de red en los que está ejecutándose el servlet.
`getServlet()`	Regresa un objeto Servlet de un nombre dado. Útil cuando se desea tener acceso a los servicios de otro servlet.
`getServletNames()`	Regresa un objeto Enumeration con los nombres de los servlets disponibles en el actual espacio de nombres.
`log()`	Escribe información al archivo de bitácora. Ese archivo y su formato son específicos de cada servidor web.

El siguiente código de ejemplo muestra la manera en que un servlet utiliza el servidor para escribir un mensaje al archivo de bitácora cuando el servlet se inicializa:

```
private ServletConfig config;
public void init(ServletConfig config) {
  // Store config in an instance variable
  this.config = config;
  ServletContext sc = config.getServletContext();
  sc.log( "Started OK!" );
}
```

Los parámetros definidos para el contexto en el descriptor de la aplicación web, van dentro de las etiquetas <context-param>.

```
<web-app>
    ...
    <context-param>
        <param-name>dburl</param-name>
        <param-value>jdbc:databaseurl</param-value>
    </context-param>
    ...
<web-app>
```

Los métodos de ServletContext que sirven para obtener los parámetros de inicialización son.

Método	Descripción
`String getInitParameter(String paramName)`	Regresa sólo un valor asociado con el parámetro dado o null si no esta disponible.
`Enumeration getInitParameterNames()`	Regresa un Enumeration de Strings de todos los nombres de los parámetros.

Interface javax.servletServletContextListener

Ésta interface permite al desarrollador saber cuando el servlet context es inicializado o destruido. Por ejemplo se puede querer crear una conexión a una base de datos tan pronto como el contexto sea inicializado y cerrarla cuando el contexto sea destruido.

Método	Descripción
`String contextDestroyed(ServletContextEvent sce)`	Llamado cuando el contexto es destruido.
`String contextInitialized(ServletContextEvent sce)`	Llamado cuando el contexto es iniciado.

A continuación se muestra un ejemplo que implementala interface ServletContextListener

```
package com.abcinc;
import javax.servlet.*;
import java.sql.*;

public class MyServletContextListener implements ServletContextListener
{
      public void contextInitialized(ServletContextEvent sce)
      {
            try
            {
```

```
                Connection c = //crea conexión a la base de datos;
                sce.getServletContext().setAttribute("connection", c);
            }catch(Exception e) { }

        }

        public void contextDestroyed(ServletContextEvent sce)
        {
            try
            {
                Connection c = (Connection)
                sce.getServletContext().getAttribute("connection");
                c.close();
            }catch(Exception e) { }
        }
    }
```

Para agregar Listeners en el descriptor de la aplicación se hace lo siguiente:

```
<listener>
   <listener-class>
      com.abcinc.MyServletContextListener
</listener-class>
</listener>
```

2.6 Interfaz RequestDispatcher

Un componente web puede invocar a otros recursos web de 2 maneras: indirecta y directamente. Un componente web invoca indirectamente a otros recursos web cuando se agrega a su respuesta enviada al cliente un URL que apunta a otro componente web.

Un componente web en ejecución también puede invocar a otros recursos y existen 2 posibilidades en este caso: puede incluir el contenido de otro recurso o puede redireccionar una petición al otro recurso.

Para invocar un recurso disponible en el servidor en el que se está corriendo el componente web, primero se debe obtener un objeto `RequestDispatcher` utilizando el método `getRequestDispatcher("URL")`. Se puede obtener un objeto `RequestDispatcher` ya sea desde una petición o desde el contexto web, sin embargo, ambas alternativas se comportan de manera un poco diferente. El método `getRequestDispatcher` toma la ruta del recurso solicitado como un argumento. Una petición puede tomar una ruta relativa (esto es, una ruta que no comienza con una diagonal /), pero el contexto web requiere una ruta relativa. Si el recurso no está disponible, o si el servidor no ha implementado un objeto `RequestDispatcher` para ese

tipo de recurso, el método getRequestDispatcher regresará null. Se aconseja que los servlets deban estar preparados para tratar esta situación.

Por lo general, es útil incluir otros recursos web en la respuesta regresada por un componente web, como por ejemplo texto o imágenes, un banner o información de los derechos de autor. Para incluir otros recursos, se invoca al método include de un objeto RequestDispatcher:

```
include(request, response);
```

Si el recurso es estático (como texto fijo o imágenes), el método include "incrusta" el recurso desde la ejecución del servlet en el servidor (server-side include). Si el recurso es un componente web, el efecto del método es enviar la petición al componente web incluido, ejecuta el componente web y entonces incluye el resultado de esa ejecución en la respuesta del servlet contenedor. Un componente web incluido tiene acceso al objeto request, pero está limitado en lo que puede hacer con el objeto response:

- Puede escribir en el cuerpo de la respuesta (objeto response) y realizar la asignación (commit) de una respuesta.

- No puede establecer o configurar cabeceras ni llamar a algún método (por ejemplo setCookie) que afecte a las cabeceras de la respuesta.

En algunas aplicaciones se desea tener un componente web que realice procesamiento preliminar de una petición y tener otro componente que genere la respuesta. Para transferir el control a otro componente web, se debe invocar al método forward de un objeto RequestDispatcher. Cuando una petición es redireccionada, el URL de la petición se establece a la ruta de la nueva página a la que se direccionó la petición. Si el URL original es requerido para algún proceso, se puede salvar como un atributo más de la petición.

El método forward debe ser utilizado para dar a otro recurso la responsabilidad de responderle al cliente. Si el servlet ha accesado a un objeto ServletOutputStream o a un objeto PrintWriter, utilizar el método forward lanzará una excepción IllegalStateException.

2.7 Descriptor de aplicaciones

El descriptor de una aplicación, como su nombre lo indica describe la aplicación para el contendor de Servlets la entienda. A continuación se muestra un archivo `web.xml`.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>        <- Declares the XML version and
                                                       character set used in this file
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >   <- Declares the
                                                            schema
  <display-name>Test Webapp</display-name>                  definition for
                                                            this file
    <context-param>
      <param-name>author</param-name>                    Specifies a parameter for
      <param-value>john@abc.com</param-value>            this web application
    </context-param>

  <servlet>    <- Specifies a servlet
    <servlet-name>test</servlet-name>
    <servlet-class>com.abc.TestServlet</servlet-class>
    <init-param>    <- Specifies a parameter for this servlet
      <param-name>greeting</param-name>
      <param-value>Good Morning</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>test</servlet-name>              Maps /test/*
    <url-pattern>/test/*</url-pattern>             to test servlet
  </servlet-mapping>

  <mime-mapping>
```

2.7.1 Elemento <servlet>

Cada `<servlet>` debajo de `<web-app>` define un servlet en la aplicación, en la siguiente imagen se muestra el uso de esta etiqueta.

```
<servlet>
  <servlet-name>us-sales</servlet-name>    <- The servlet name
  <servlet-class>com.xyz.SalesServlet</servlet-class>    <- The
                                                            servlet
  <init-param>                                              class
    <param-name>region</param-name>
    <param-value>USA</param-value>        The servlet
  </init-param>                           parameters
  <init-param>
```

El contenedor de servlets hace una instancia y lo asocia con el nombre del servlet.

<servlet-name> : Esta etiqueta define el nombre del servlet.
<servlet-class> : Esta etiqueta especifica la clase Java que será usada por el contenedor de Servlets.
<init-param>: Esta etiqueta sirve para agregar parámetros iniciales al servlet, se tiene que crear una de éstas por cada parámetro que se necesite. Contiene las etiquetas <param-name> que indica el nombre del parámetro y <param-value> que indica el calor del parámetro. En el servlet los parámetros se obtienen con `ServletConfig.getInitParameter("paramname")`.

2.7.2 Elemento <servlet-mapping>

Simplemente hay que especificar el URL que será manejada por el Servlet. El contenedor de Servlets utiliza este mapeo para invocar el Servlet apropiado.

El siguiente es un ejemplo que muestra cómo se hace el mapeo de un Servlet.

```
<servlet-mapping>
   <servlet-name>accountServlet</servlet-name>
   <url-pattern>/account/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>accountServlet</servlet-name>
   <url-pattern>/myaccount/*</url-pattern>
</servlet-mapping>
```

En estos mapeos se esta asociando /account y /myaccount con accountServlet.

2.8 Manejo de Sesiones

El seguimiento de sesión es un mecanismo que los Servlets utilizan para mantener el estado sobre la serie de peticiones desde un mismo usuario (esto es, peticiones originadas desde el mismo navegador) durante algún periodo de tiempo.

Las sesiones son compartidas por los Servlets a los que accede el cliente. Esto es conveniente para aplicaciones compuestas por varios Servlets.

Para utilizar el seguimiento de sesión debemos.

- Obtener una sesión (un objeto `HttpSession`) para un usuario.
- Almacenar u obtener datos desde el objeto `HttpSession`.
- Invalidar la sesión (opcional).

2.8.1 Obtener una Sesión

El método `getSession` del objeto `HttpServletRequest` devuelve una sesión de usuario. Cuando llamamos al método con su argumento `create` como `true`, la implementación creará una sesión si es necesario. Para mantener la sesión apropiadamente, debemos llamar a `getSession` antes de escribir cualquier respuesta. En el siguiente código se muestra la creación de una sesión.

```
public class CatalogServlet extends HttpServlet {
      public void doGet (HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException
      {
            // Get the user's session and shopping cart
            HttpSession session = request.getSession(true);
            ...
            out = response.getWriter();
            ...
      }
}
```

2.8.2 Almacenar y Obtener Datos desde la Sesión

El Interface `HttpSession` proporciona métodos que almacenan y recuperan:

- Propiedades de Sesión estándar, como un identificador de sesión.
- Datos de la aplicación, que son almacenados como parejas nombre-valor, donde el nombre es un `string` y los valores son objetos del lenguaje de programación Java. Como varios Servlets pueden acceder a la sesión de usuario, deberemos adoptar una convención de nombrado para organizar los nombres con los datos de la aplicación.

```
public class CatalogServlet extends HttpServlet {

   public void doGet (HttpServletRequest request,
   HttpServletResponse response)
   throws ServletException, IOException
   {

         HttpSession session = request.getSession(true);

         //obteniendo el shopping cart.

         ShoppingCart cart = (ShoppingCart)session.getValue(session.getId());

         if (cart == null) {
         cart = new ShoppingCart();
         session.putValue(session.getId(), cart);
         }
   ...
   }
}
```

Una sesión puede ser designada como nueva cuando hace que el método `isNew` de la clase `HttpSession` devuelva `true`, indicando que, por ejemplo, el cliente, todavía no sabe nada de la sesión. Una nueva sesión no tiene datos asociados.

2.8.3 Invalidar una sesión

Una sesión de usuario puede ser invalidada manual o automáticamente, dependiendo de donde se esté ejecutando el servlet. (Por ejemplo, el Java Web Server, invalida una sesión cuando no hay peticiones de página por un periodo de tiempo, unos 30 minutos por defecto). Invalidar una sesión significa eliminar el objeto `HttpSession` y todos sus valores del sistema.
Para invalidar manualmente una sesión, se utiliza el método `invalidate` de "session". Algunas aplicaciones tienen un punto natural en el que invalidar la sesión.

```
public class ReceiptServlet extends HttpServlet {
      public void doPost(HttpServletRequest request,HttpServletResponse
      response)
      throws ServletException, IOException {
            ...
            scart = (ShoppingCart)session.getValue(session.getId());
            ...
            // Clear out shopping cart by invalidating the session
            session.invalidate();
            // set content type header before accessing the Writer
            response.setContentType("text/html");
            out = response.getWriter();
            ...
      }
}
```

3. Java Server Page (JSP)

Una página JSP es una página Web que contiene código Java insertado dentro del código HTML. Todo el código Java es interpretado del lado del servidor, esto implica que el cliente no puede ver ningún tipo de código, lo único que ve es una página HTML.

Gracias a un JSP que tiene incrustado código Java se pueden hacer páginas dinámicas. Una página dinámica es aquella que no tiene un contenido fijo, si no, que éste puede variar con el tiempo. El contenido dinámico puede ser obtenido de una base de datos, de archivos, etc.

3.1 Sintaxis de los JSPs

Como cualquier otro lenguaje, los JSPs tienen su gramática bien definida. La siguiente tabla muestra los elementos pertenecientes a los JSPs.

Tipo de etiqueta	Descripción	Sintaxis
Directivas	Información acerca de los JSPs.	<%@ Directivas %>
Declaraciónes	Declara y define métodos y variables.	<%! Declaraciones Java %>
Scriptlets	Permiten escribir código Java.	<% Código Java %>
Expresiones	Sirve para imprimir variables.	<%= Expresión %>
Acciones	Comandos al motor JSP	<jsp:acción />
Comentarios	Usado para documentar	<%-- Texto --%>

3.1.1 Directivas.

Poseen información acerca de las páginas JSPs a la máquina JSP. Existen tres tipos de directivas mencionadas a continuación.

page: Informa al motor acerca de las propiedades de un JSP.

```
<%@ page language="java" %>
```

include: Informa al motor que incluya los contenidos de otro archivo en la página actual.

```
<%@ include file="copyright.html" %>
```

taglib: Se usa para asociar un taglib.

```
<%@ taglib prefix="test" uri="taglib.tld" %>
```

3.1.2 Declaraciones.

Declaran y definen variables y métodos que pueden ser usados en el JSP. Las variables sólo son inicializadas una vez y retiene el valor para los siguientes request.

```
<%! int count = 0; %>
```

También se puedè hacer lo siguiente:

```
1. <%!
       String color[] = {"red", "green", "blue"};
       String getColor(int i)
       {
       return color[i];
       }
   %>
```

```
2. <%! private int j = 0; %>
   <%! public int suma(int a, int b)
           {
            return a + b;
           }
       %>
```

Típicamente, las declaraciones son utilizadas para crear una variable a usar en JSP o para crear un método que se quiera utilizar en un JSP sin tener que estar declarándolo cada vez que se tenga que utilizar.

3.1.3 Scriptlets.

Son fragmentos de código Java, el scriptlet es ejecutado cada vez que la página es accesada.

```
<%
        out.print("<html><body>");
        count++;
        out.print("Welcome! You are visitor number " + count);
        out.print("</body></html>");
%>
```

3.1.4 Expresiones.

Sirve para imprimir el valor de una variable o expresión. La expresión es evaluada cada vez se accesa a la página. Prácticamente son una única línea de código y no incluyen operadores de control de flujo como `if`, `while`, `for`, etc.

```
<html><body>
<%@ page language="java" %>
<%! int count = 0; %>
     Bienvenido! Visitante numero:<%= ++count %>
</body></html>
```

Importante: Las expresiones no terminan con punto y coma (;) mientras que las declaraciones y los scriptlets sí llevan punto y coma. Además las expresiones son tomadas de manera interna por el motor de JSP como cadenas (como objetos de tipo `java.lang.String`).

3.1.5 Acciones.

Son comandos dados al motor de JSPs. Sirven para tareas específicas durante al ejecución de la página. Existen 6 acciones estándar:

- `jsp:include`
- `jsp:forward`
- `jsp:useBean`
- `jsp:setProperty`
- `jsp:getProperty`
- `jsp:pliguin`

Las dos primeras acciones `jsp:incluye` y `jsp:forward`, habilitan a los JSPs el reuso de componentes. Las siguientes `jsp:useBean,jsp:setProperty` y `jsp:getProperty` están relacionados con los javaBeans. Feinalmete `jsp:plugin` indica a la motor de JSPs que genere el código apropiado para componentes embebidos como applets.

3.1.6 Comentarios.

Los comentarios no tienen ningún efecto sobre la salida de los JSPs, generalmente se utilizan para hacer documentación sobre la página.

```
<html><body>
Welcome!
<%-- JSP comment --%>
<% //Java comment %>
<!-- HTML comment -->
</body></html>
```

3.2 Atributos de la directiva.

La directiva `page` informa al motor de JSPs sobre todas las propiedades de la página JSP. Existen 12 posibles atributos, pero en la práctica los que más se suelen usar son los siguientes cuatro.

`import`: Sirve para importar clases java y paquetes. Los valores que tiene por default son los siguientes:

- `java.lang.*;`
- `javax.servlet.*;`
- `javax.servlet.jsp*;`
- `javax.servlet.http.*;`

Y se usa de la siguiente manera:

```
<%@ page import="java.util.*, java.io.*, java.text.*,
com.mycom.*, com.mycom.util.MyClass " %>
```

O también se puede colocar en varias directivas:

```
<%@ page import="java.util.* " %>
<%@ page import="java.io.* " %>
<%@ page import="java.text.* " %>
<%@ page import="com.mycom.*, com.mycom.util.MyClass " %>
```

`session`: Este atributo indica cuando el JSP es parte de la sesión HTTP y tiene por valor default `true`.

```
<%@ page session="false" %>
```

`errorPage`: se utiliza para especificar una URL de error, por defecto tiene valor `null`.

```
<%@ page errorPage="errorHandler.jsp" %>
```

`isErrorPage`: Indica si el JSP es capaz de manejar errores. Su valor predeterminado es `false`.

```
<%@ page isErrorPage="true" %>
```

3.3 Usando Scriptlets.

Como todos los miembros definidos en un JSP y finalmente se convierten en un Servlet, no importa el orden.

```
<html>
<body>
Using pi = <%=pi%>, the area of a circle<br>
with a radius of 3 is <%=area(3)%>
<%!
double area(double r)
{
return r*r*pi;
}
%>
<%! final double pi=3.14159; %>
</body>
</html>
```

Las variables declaradas en un scriptlet se convierten en locales, por lo tanto el orden en que aparecen sí tiene importancia.

```
<html>
<body>
<% String s = s1+s2; %>          <- No se ha definido la variable s2
<%! String s1 = "hello"; %>      <- Variable s1
<% String s2 = "world"; %>       <- Variable s2
<% out.print(s); %>
</body>.
</html>
```

Inicialización de variables.

En Java, las variables de instancia son automáticamente inicializadas con sus valores de default, mientras que las variables locales deben de ser inicializadas explícitamente cuando son usadas. Por lo tanto las variables declaradas en "declaraciones" son iniciadas automáticamente y las declaradas en scriptlets deben de ser inicializadas explícitamente.

```
<html>
<body>
<%! int i; %>
<% int j; %>
The value of i is <%= i++ %> <br>       <- El valor es 0.
The value of j is <%= j++ %> <br>       <- No ha sido inicializado.
</body>
</html>
```

Para que el código anterior compile:

```
<% int j=0; %>
```

Hay que recordar que las variables de instancia son creadas e inicializadas sólo una vez.

Como ya se mencionó, los scriptlets sirven para insertar todo tipo de código Java y se puede hacer una mezcla entre código Java y etiquetas HTML. La mayor ventaja de un scriptlet es poder abrir una llave ({) y poder cerrarla líneas de código del JSP más abajo (}), cómo por ejemplo:

```
1. <%
      if( i < 0 )
      {
       out.println("i es mayor que cero");
      }
   %>

2. <%
      if( nombre != null )
      {
       %>
       Bienvenido <%= nombre %>
       <%
      }
   %>
```

3.4 Objetos implícitos

Para los JSPs, Java cuenta con nueve objetos implícitos que están disponibles en cualquier lugar de un JSP y que por lo tanto, no necesitan crearse ni iniciarse:

Objetos Implícitos	Tipo	Alcance	Métodos más utilizados
request	Subclase de javax.servlet.ServletRequest	Petición (objeto Request)	getAttribute, getParameter, getParameterNames, getParameterValues
response	Subclase de javax.servlet.ServletResponse	Página	Comúnmente utilizada de manera interna y no por los autores de documentos JSP
pageContext	javax.servlet.jsp.PageContext	Página	findAttribute, getAttribute, getAttributesScope, getAttributeNamesInScope
session	javax.servlet.http.HttpSession	Sesión	getId, getValue, getValueNames, putValue
application	javax.servlet.ServletContext	Aplicación	getMimeType, getRealPath
out	javax.servlet.jsp.JspWriter	Página	clear, clearBuffer, flush, getBufferSize, getRemaining
config	javax.servlet.ServletConfig	Página	getInitParameter, getInitParameterNames
page	java.lang.Object	Página	Comúnmente utilizada de manera interna y no por los autores de documentos JSP
exception	java.lang.Throwable	Página	getMessage, getLocalizedMessage, printStackTrace, toString

3.5 Alcances de los JSPs.

El concepto de alcance significa la forma que los datos se comparten entre los Servlets usando los tres contenedores de objetos `ServletContext`, `HttpSession` y `ServletRequest`. Los alcances asociados con los tres alcances son respectivamente `aplication`, `session` y `request`. Para los JSPs existen los mismos alcances, pero se agrega otro: `page`, que esta dentro del objeto `PageContext`.

3.5.1 Aplication

En el alcance `aplication` los objetos son compartidos a través de todos los componentes de la aplicación web y son accesibles durante el tiempo de vida de la aplicación. Estos objetos son mantenidos en el `ServletContext`. Para compartir u obtener objetos de `ServletContext` se utilizan los métodos `setAttribute()` y `getAttribute()`.

3.5.2 Session

Los objetos en sesión son compartidos a través de todas las peticiones que haga un solo usuario y están accesibles mientras que la sesión esté activa. Para compartir objetos a este nivel se utiliza `session.setAttribute` y `session.getAttribute`.

3.5.3 Request

Los objetos en request son compartidos a través de los componentes que hayan realizado la misma petición y seguirán activos mientras siga la misma petición. Para obtener un objeto se utiliza: `request.setAttribute` y `request.getAttribute`.

3.5.4 Page

Los objetos en éste tipo de alcance son accesibles sólo y sólo en la traducción del JSP. Estos objetos están disponibles sólo a través de pageContext. También tiene los métodos `setAttribute()` y `getAttribute()`.

4. JavaBeans.

4.1 Utilización de JavaBeans en JSP's

Los JavaBeans son componentes de software independiente que se usan para ensamblar otras aplicaciones. Simplemente es son datos encapsulados en forma de variables de instancia; éstas variables de instancia son referidas como propiedades del bean. La clase provee métodos para accesar y cambiar un valor.
Para utilizar JavaBeans en los JSP's es necesario conocer la etiqueta `<jsp:useBean>` la cual crea una instancia de un JavaBean. Se utilizan los JavaBeans porque:

1. En JSPs y servlets, los JavaBeans son utilizados principalmente porque sus propiedades pueden ser dinámicamente modificadas y accesadas en tiempo de ejecución.

2. Los JSPs por lo general utilizan JavaBeans que tienen un tiempo de vida de una única página o de una petición de usuario, por lo que no tiene mucho sentido almacenar el estado de un JavaBean. Si un JavaBean está siendo utilizado con el alcance de sesión, entonces será serializado automáticamente.

Para que una clase sea considerada un JavaBean en un JSP debe de seguir las siguientes características:

- La clase debe de tener un constructor público sin argumentos. Esto permita que la clase sea instanciada por el motor de JSPs.
- Por cada propiedad la clase debe contener dos métodos públicos que permiten obtener o cambiar el valor de la propiedad. El nombre de los métodos que accesan a un valor se llaman accesors y deben de ser `getXXX()` y el nombre de los métodos que cambian la propiedad se llaman mutators y deben de ser `setXXX()`, dónde XXX es el nombre de la propiedad. Por ejemplo:

```
public String getColor();
public void setColor(String);
```

El siguiente código es un ejemplo de un JavaBean llamado AddressBean.

```
public class AddressBean
{
//properties
private String street;
private String city;
private String state;
private String zip;

//setters or mutators
public void setStreet(String street){ this.street = street; }
public void setCity(String city) { this.city = city; }
public void setState(String state) { this.state = state; }
public void setZip(String zip) { this.zip = zip; }

//getters
public String getStreet(){ return this.street; }
public String getCity() { return this.city; }
public String getState() { return this.state; }
public String getZip() { return this.zip; }
}
```

El nombre de la clase trae pegada la palabra Bean, lo cual no es necesario pero se utiliza como una convención.

4.2 Usando JavaBeans y JSPs.

Hay tres acciones estándar que pueden usar los JavaBeans con los JSPs.

Acción	Descripción
`<jsp:useBean>`	Declara el uso del Javabean en una página JSP.
`<jsp:setProperty>`	Envío de valores a las propiedades de los JavaBeans.
`<jsp:getProperty>`	Obtención de los valores de las propiedades del JavaBean.

4.2.1 Declarando JavaBeans usando <jsp:useBean>

El `jsp:useBean` declara las variables en el JSP y asocia la instancia con en JavaBean. Este tipo de acción tiene cinco atributos:

Atributo	Descripción	Ejemplo
`id`	El nombre con el que el bean es identificado en el JSP.	Id="direccion"
`scope`	El alance del bean.	scope="session"
`class`	La clase Java del bean.	class="AddressBean"
`type`	Especifica el tipo de la variable que será usada para referirse al bean.	type="AdderssBean"
`beanName`	El nombre del bean	beanName="AddressBean"

De los atributos anteriores: id es forzoso, scope es opcional y class,type y beanName deben de ser usadas en las siguientes combinaciones:

- class
- type
- class y type
- beanName y type

La sintaxis de la etiqueta queda de la siguiente manera:

```
<jsp: useBean id=" beanInstanceName "
scope=" page |request| session| application"
{
 class=" package. class " [ type=" package. class " ] |
 type=" package. class "|
 beanName="{package. class | <%= expression %>}" type="package.class"
}
{
 />|> other elements </ jsp: useBean>
}
```

La sintaxis anterior nos indica que hay 3 maneras de instanciar un JavaBean y sigue la convención de Sun Microsystems de indicar una lista de opciones mutuamente excluyentes (se requiere solo una opción) que se encuentran entre llaves ({}). Una linea vertical (|) separa cada una de esas opciones excluyentes. Los atributos opcionales son listados entre corchetes ([]). Es importante notar que cuando el atributo beanName es utilizado, se evalúa una expresión para el tipo o nombre de archivo del Bean. Sin embargo, la forma más simple y común de instanciar un JavaBean es especificando un id para el Bean, dejar el alcance (scope) al de por defecto (page) y dar el nombre de la clase del JavaBean. Este es un ejemplo de un JavaBean que es un contador de accesos a una página:

```
<jsp:useBean id="miContador" class="HitCountBean" scope="session"/>
```

La anterior etiqueta es equivalente a:

```
HitCountBean miContador = (HitCountBean)
session.getAttribute("miContador ");
if (miContador == null)
{
      miContador = new HitCountBean ();
      session.setAttribute("miContador ", miContador);
}
```

4.2.2 Inicializando las propiedades del bean.

Dado que el Javabean es instanciado por el motor de JSPs usando un constructor sin argumentos, no se pueden inicializar las propiedades del bean; para resolver este problema se tiene la siguiente solución:

```
<jsp:useBean id="address" scope="session" class="AddressBean" >
<%
address.setStreet("123 Main St. ");
%>
</jsp:useBean>
```

4.2.3 Uso de <jsp:setProperty>.

El uso de `jsp:setProperty` consiste en asignar nuevos valores a las propiedades del bean. Esta acción tiene cuatro propiedades:

Nombre	Descripción
name	El nombre del bean con que es identificado en el JSP.
property	El nombre de la propiedad del bean.
value	El nuevo valor asignado a la propiedad.
param	El nombre del parámetro disponible en HttpServletRequest, que será asignado como nuevo valor de la propiedad del bean.

Enseguida se muestra un ejemplo de cómo se utiliza esta acción.

```
<jsp:setProperty name="address" property="city" value="Albany" />
<jsp:setProperty name="address" property="state" value="NY" />
```

El anterior código es equivalente:

```
<%
address.setCity("Albany");
address.setState("NY");
%>
```

En el siguiente ejemplo se muestra la forma de utilizar al atributo `param` en vez de `value`.

```
<jsp:setProperty name="address" property="city" param="myCity" />
<jsp:setProperty name="address" property="state" param="myState"/>
```

En este caso el valor asignado a la propiedad del bean viene del request; esto también es equivalente a:

```
<jsp:setProperty name="address" property="city" value="<%= theCity %>" />
```

En un scriptlet también es equivalente a:

```
<%
address.setCity(request.getParameter("myCity"));
address.setState(request.getParameter("myState"));
%>
```

Las anteriores técnicas se utilizan cuando los nombres de los parámetros que se obtienen del request, no encajan con las propiedades de los beans. De lo contrario se puede utilizar:

```
<jsp:setProperty name="address" property="city" />
<jsp:setProperty name="address" property="state" />
```

Lo anterior es equivalente a:

```
<jsp:setProperty name="address" property="city" param="city" />
<jsp:setProperty name="address" property="state" param="state" />
```

Cuando todos los nombres de todos los parámetros recibidos del request son iguales a los de las propiedades del JavaBean, se puede hacer lo siguiente:

```
<jsp:setProperty name="address" property="*" />
```

En vez de envíar propiedad por propiedad, se mandan todas al mismo tiempo.

4.2.3 Uso de <jsp:getProperty>.

El uso de esta acción es para traer los valores de las propiedades del bean. La sintaxis de ésta acción es muy sencilla:

```
<jsp:getProperty name="beanInstanceName" property="propertyName" />
```

Como se observa en la sentencia sólo se tienen dos atributos que no se pueden discriminar el uno del otro. El atributo `name` indica el nombre de la instancia del bean declarada en `<jsp:useBean>` y `property` especifica la propiedad cuyo valor será impreso.

Por ejemplo:

```
<jsp:getProperty name="address" property="state" />
<jsp:getProperty name="address" property="zip" />
```

Que es equivalente a:

```
<%
out.print(address.getState());
out.print(address.getZip());
%>
```

Anexo C: Repaso de HTML

HTML (*HyperText Markup Language*) es un lenguaje muy sencillo que permite describir hipertexto, es decir, texto presentado de forma estructurada y agradable, con *enlaces* (*hyperlinks*) que conducen a otros documentos o fuentes de información relacionadas, y con *inserciones* multimedia (gráficos, sonido...).

Mi primer pagina

```
<HTML>

<HEAD >

<TITLE>Yo soy el titulo de tu
página</TITLE>
</HEAD>
<BODY >

<center><h1>Hola mundo</h1></center>

<p> <p>
Hola bienvenido a tu primer pagina con html
</BODY>
</HTML>
```

Explicación del código fuente:

Al principio de la página

<HTML> lo ultimo es ***</HTML>***

Esto es una manera de decirle a tu navegador que lo que hay en medio de esas dos etiquetas es texto HTML.

De aquí podemos sacar ya varias ideas básicas:

- Cada marca va siempre entre los signos < y >.
- Las Marcas siempre son dobles (casi siempre), una para indicar el principio y otra para el final.
- La marca que indica el final es igual que la del principio pero con el signo / delante.

Dentro de estas Etiquetas (Entre Etiqueta-Principio y Etiqueta-Final) encontramos otro que dice:

<HEAD> </HEAD>

Que indica la cabecera de la página, la cual sirve para meter otras cosas cómo:

<TITLE> Esto-Debería-Ser-El-Título-De-La-Pagina </TITLE>

Que es, como su propio nombre indica (en inglés), el titulo de la Pagina (El que aparece en la ventana del navegador).

Después viene el cuerpo de la página en sí: El texto, imágenes, links, etc. todo ello dentro de los siguientes Etiquetas:

<BODY> </BODY> Aquí va el cuerpo de la página.

<P> Sirve para delimitar un párrafo. Inserta una línea en blanco antes del texto.

<CENTER> </CENTER> Permite centrar todo el texto del párrafo.

<H1> ... </H1> Es la cabecera de nivel 1 y sirve para modificar de manera rápida el tamaño de alguna frase o párrafo.

Una vez hecho esto guarda el archivo con el nombre que quieras pero con extensión con extensión ***. html o *.htm** y cambiando el tipo de archivo de tipo de texto a todos los archivos.

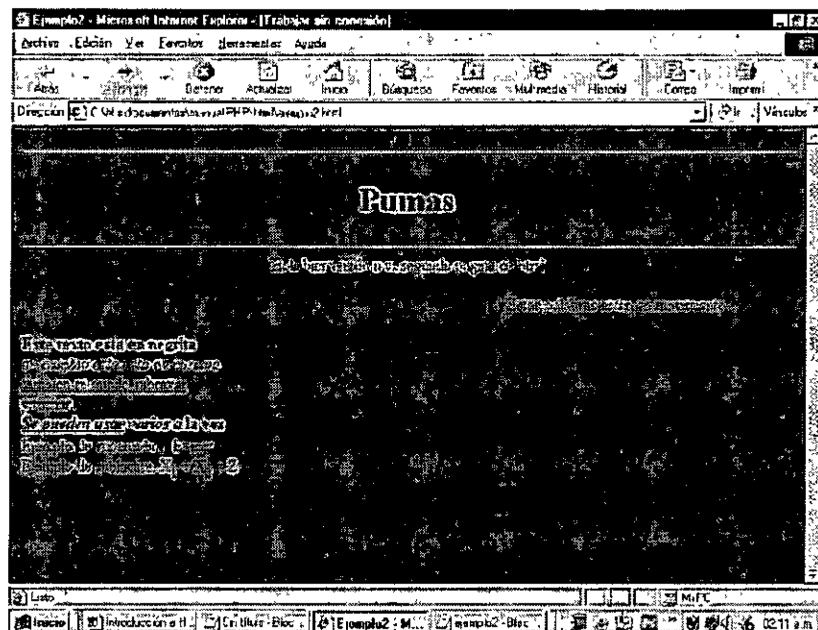
Abre cualquier explorador de Internet, Explorer o Netscape y abre tu archivo desde ahí, o si lo prefieres ve a la carpeta donde se encuentre tu primer pagina y ábrela directamente, si hiciste todo correctamente debe aparecerte con el icono del explorador que tengas instalado en tu maquina.

La salida de la primera página debe verse de esta manera:

Ahora veamos una página un poco menos plana:

```
<HTML>
<HEAD > <TITLE> Ejemplo2 </TITLE> </HEAD>
<BODY bgcolor=#345678>
<hr> <center> <h1> Pumas </h1> </center> <hr>
<center> Hola bienvenido a tu segunda pagina de html </center>
<p><p>
<marquee> Estas palabras están en movimiento</marque>
<B>Este texto esta en negrita</B>
<BR> <I>en cambio este esta en cursiva</I>
            <BR> <U>también se puede subrayar</U>
<BR> <S>y tachar</S>
<BR> <B><I><U>Se pueden usar</U> varios a</I> la vez</B><BR>
Ejemplo de superíndice: 6<SUP>2</SUP>= 36<BR>
```

```
Ejemplo de subíndice: X<SUB>1</SUB> + X<SUB>2</SUB> = Z
</BODY>
</HTML>
```



Explicación de las etiquetas para este ejemplo y algunas más:

Etiqueta	Utilidad
<B> … </B>	Pone el texto en negrita.
<I> … </I>	Representa el texto en cursiva.
<U> … </U>	Para subrayar algo.
<S> … </S>	Para tachar.
[…]	Letra superíndice.
_…	Letra subíndice.
<HR> *	Inserta una barra horizontal
 *	Salto de línea
<P>*	Inicio de un párrafo
<marquee>…</marquee>	Desplaza el texto por toda la pantalla
<BIG> … </BIG>	Incrementa el tamaño del tipo de letra.
<SMALL> … </SMALL>	Disminuye el tamaño del tipo de letra.

<P>	Sirve para delimitar un párrafo. Inserta una línea en blanco antes del texto.
<CENTER> ... </CENTER>	Permite centrar todo el texto del párrafo.
<DIV ALIGN=x> ... </DIV>	Permite justificar el texto del párrafo a la izquierda (ALIGN=LEFT), derecha (RIGHT), al centro (CENTER) o a ambos márgenes (JUSTIFY)
<BLOCKQUOTE> ... Para dejando </BLOCKQUOTE>	citar un texto ajeno. Se suele implementar márgenes tanto a izquierda como a derecha, razón por la que se usa habitualmente.

Etiqueta	Resultado
<H1> ... </H1>	Cabecera de nivel 1
<H2> ... </H2>	Cabecera de nivel 2
<H3> ... </H3>	Cabecera de nivel 3
<H4> ... </H4>	Cabecera de nivel 4
<H5> ... </H5>	Cabecera de nivel 5
<H6> ... </H6>	Cabecera de nivel 6

Las etiquetas marcadas con (*) no necesitan ser cerradas.

El "atributo" que se encuentra dentro de la etiqueta body nos permite modificar el color de fondo de pantalla

<body bgcolor=#345678>	Fondo de la página

El tipo de color puede agregarse en forma hexadecimal logrando así varias degradaciones de colores o por el nombre en ingles de dicho color (blue, green, red, gray, etc).

En html algunos caracteres como los acentos, diéresis y algunas letras especiales tiene que escribirse antes de cada letra como se muestra a continuación:

Formularios y links

Los formularios son el sistema del que nos provee el HTML para enviar información desde una página web a algún programa u otro recurso en un ordenador remoto. Esto quiere decir que un formulario no sirve de mucho si no tenemos un lugar al que enviarlo.

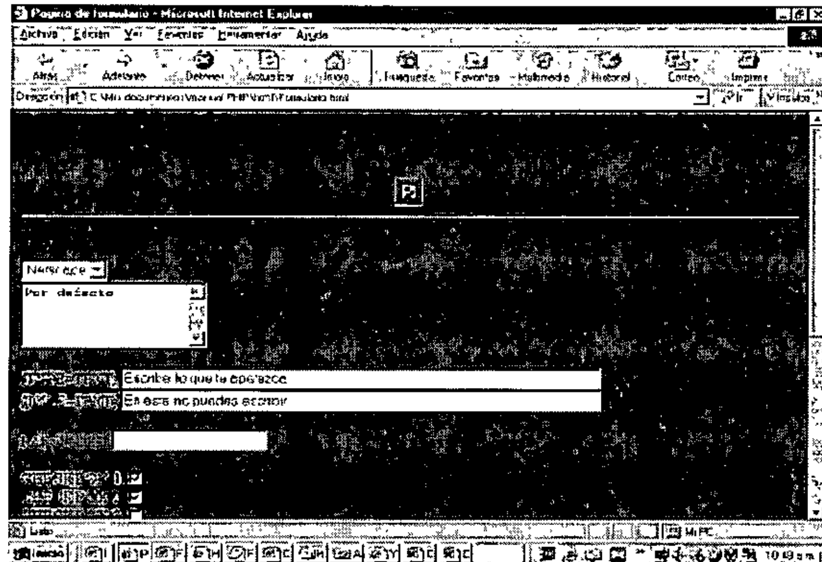
El funcionamiento de los formularios es muy simple: La persona que visita la página rellena el formulario con los datos que se quieran enviar y, al picar en un botón, estos son enviados al lugar que corresponda

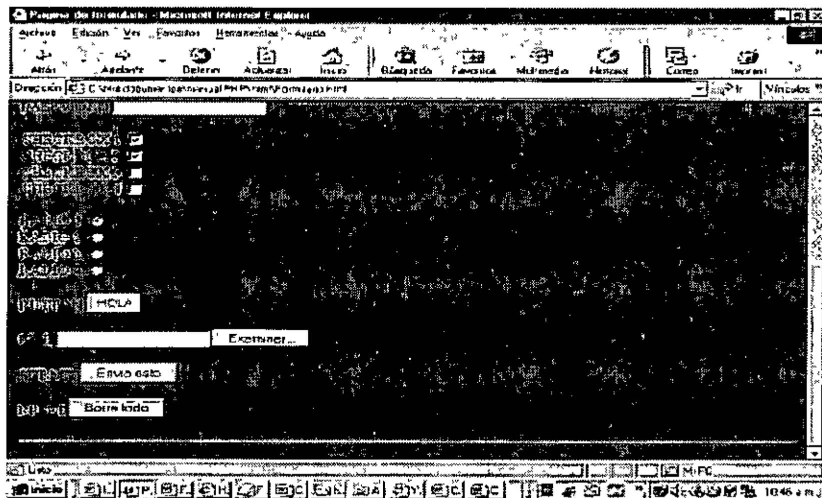
La primera etiqueta que debemos tener en cuenta es el que crea un formulario. Esta etiqueta deberá englobar en su interior a todos los elementos que formen parte de este:

```
<HTML>
<HEAD>
<TITLE>Página de formulario </TITLE>
</HEAD>
<BODY bgcolor=#234567>
<br>
<br>
<a href="ejemplo2.html"> soy un link </A>
<BR>
<CENTER>
<img scr=Dibujo.bmp>
</CENTER>
<hr>
<FORM METHOD="POST" ACTION="ejemplo1.html">
<br>
<SELECT NAME="Navegador">
<OPTION>Netscape
<OPTION>Explorer
<OPTION>Opera
<OPTION>Lynx
<OPTION>Otros
</SELECT>
<br>
<TEXTAREA rows=4 cols=20>Por defecto</TEXTAREA>
<P>(TYPE=TEXT) <INPUT TYPE="TEXT" VALUE="Escribe lo que te apetezca " SIZE="65"
NAME="Text1">
<BR>(TYPE=TEXT) <INPUT TYPE="TEXT" VALUE="En este no puedes
escribir" SIZE="65" NAME="Text2" READONLY>
<P>PASSWORD <INPUT TYPE="PASSWORD" NAME="Password">
<P>CHECKBOX 1 <INPUT TYPE="CHECKBOX" NAME="Checkbox">
<BR>CHECKBOX 2 <INPUT TYPE="CHECKBOX" CHECKED NAME="">
<BR>CHECKBOX 3 <INPUT TYPE="CHECKBOX" NAME="Checkbox">
<BR>CHECKBOX 4 <INPUT TYPE="CHECKBOX" NAME="Checkbox">
<P>RADIO 1 <INPUT TYPE="RADIO" CHECKED NAME="Radio">
<BR>RADIO 2 <INPUT TYPE="RADIO" NAME="Radio">
<BR>RADIO 3 <INPUT TYPE="RADIO" NAME="Radio">
<BR>RADIO 4 <INPUT TYPE="RADIO" NAME="Radio">
<P>BUTTON <INPUT TYPE="BUTTON" VALUE="HOLA" NAME="Boton">
<P>FILE <INPUT TYPE="FILE" NAME="File">
<P>SUBMIT <INPUT TYPE="SUBMIT" VALUE="Env&iacute;a esto">
<P>RESET <INPUT TYPE="RESET" VALUE="Borra todo">
```

```
</FORM>
<hr>
</BODY>
</HTML>
```

Se debe ver así.





Explicación de la página.

<A HREF="destino.htm">pica aquí mismo</A> La etiqueta <A> </A> indica que lo que hay en medio, en este caso la frase "Pica aquí mismo", es un Hipervínculo (Un Link) y HREF="Lo-Que-Sea" indica el destino al que te mandará cuando pique en el con el ratón, en este caso una página que está en el mismo directorio que en la que nos encontramos y con el nombre de destino.htm.

Pero, ¿y si, en vez de en el mismo, está en un subdirectorio? Pues le indicamos la ruta y en paz:

<A HREF="directorio/destino.htm">pica aquí mismo</A>

Si la página está en el directorio justamente superior (el directorio padre), se indicará la ruta de este modo:

<A HREF="../destino.htm">pica aquí mismo</A>

Y si está en el superior a este (algo así como el abuelo), se haría así:

<A HREF="../../destino.htm">pica aquí mismo</A>

Y si está en un directorio "paralelo":

<A HREF="../directoriohermano/destino.htm">pica aquí mismo</A>

<IMG> Esta etiqueta es muy simple, ni siquiera hay que cerrarla (o sea, que no existe </IMG>) y tiene una sola opción estrictamente necesaria que es:

<IMG SRC="nombre de la imagen"> donde "nombre de la imagen" es el nombre del archivo gráfico que se quiere insertar en este punto.

Todos los elementos de un formulario deben estar encerrados entre ***<FORM>*** y ***</FORM>.*** Como parámetros cabe destacar tres. ***ACTION*** define el URL que deberá gestionar el formulario. Puede ser una dirección de correo (precedida del inevitable mailto:, en cuyo caso deberemos añadir el parámetro ENCTYPE="text/plain" para que lo que recibamos resulte legible.

Por otro lado, tenemos el parámetro ***METHOD*** define la manera en que se mandará el formulario. Es recomendable utilizar ***POST***. En el caso de que estemos mandando el formulario a nuestra dirección de correo electrónico es obligado usarlo.

Ahora vamos a ver uno a uno todos los elementos que podemos incluir en un formulario. Veremos que todos ellos tienen algo en común. Como el resultado de cualquier formulario es una lista de variables y valores asignados a las mismas, todos ellos tendrán un atributo en común: el nombre de su variable. El parámetro también será común a todos: NAME.

Menú de selección.

Los parámetros que admite ***SELECT*** son las siguientes:

Parámetro	Utilidad
SIZE	El número de opciones que podremos ver. Si es mayor que 1 veremos una lista de selección y, si no, veremos una lista desplegable.
MULTIPLE	Si lo indicamos podremos elegir más de una opción.

Y ***OPTION*** estos:

Parámetro	**Utilidad**
VALUE	Este es el valor que asignará a la variable.
SELECTED	Si lo indicamos en una de las opciones esta será la seleccionada por defecto.

Campo de texto área.

<TEXTAREA>: Este comando permite al usuario meter más de una línea de texto. El campo es mostrado en caracteres de tamaño fijo y puede ser escroleado si es necesario. El texto mostrado es usado para inicializar el campo. Típicamente los valores de los atributos ROWS y COLS determinan la dimensión visible del campo de caracteres.

Cajas de texto.

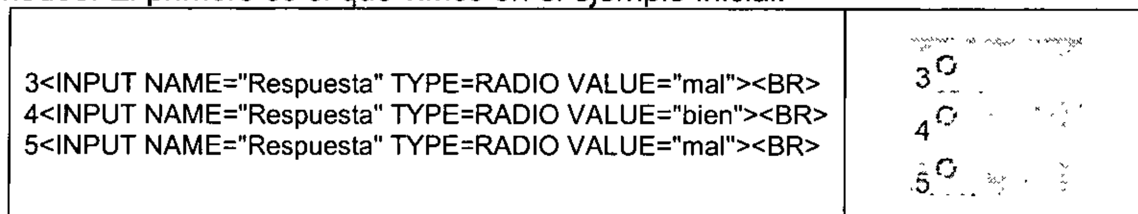
Existen tres maneras de conseguir que el usuario introduzca texto en nuestro formulario. Las dos primeras se obtienen por medio de la etiqueta **<INPUT>:**

El primero nos dibujará una caja donde escribir un texto (de una sola línea). El segundo es equivalente, pero no veremos lo que tecleemos en él. Estos son los atributos para modificarlos:

Parámetro	**Utilidad**
SIZE	Tamaño de la caja de texto.
MAXLENGTH	Número máximo de caracteres que puede introducir el usuario.
VALUE	Texto por defecto que contendrá la caja.

"PASSWORD" hace lo mismo que "TEXT", pero los caracteres que se escriban no se verán en pantalla, sino que serán sustituidos por asteriscos.

Si lo que deseamos es que el usuario decida entre varias opciones podremos hacerlo de dos modos. El primero es el que vimos en el ejemplo inicial:



```
3<INPUT NAME="Respuesta" TYPE=RADIO VALUE="mal"><BR>
4<INPUT NAME="Respuesta" TYPE=RADIO VALUE="bien"><BR>
5<INPUT NAME="Respuesta" TYPE=RADIO VALUE="mal"><BR>
```


Para asociar varios botones de radio a una misma variable les pondremos a todos ellos el mismo NAME. Aparte de esto acepta los siguientes parámetros:

Parámetro	**Utilidad**
VALUE	Este es el valor que asignará a la variable.

CHECKED	Si lo indicamos en una de las opciones esta será la que esté activada por defecto.

Puede que necesites que el usuario sencillamente nos confirme o niegue algo. Lo podremos conseguir por medio de controles de confirmación:

<INPUT NAME="Belleza" TYPE=CHECKBOX>Me considero guapo/a	☐ Me considero guapo/a

Si queremos que el control esté activado por defecto le añadiremos el parámetro CHECKED. El formulario asignará a la variable NAME el valor *on* u *off*.

Botones del formulario.

Existen dos: uno que se utiliza para mandar el formulario y otro que sirve para limpiar todo lo que haya rellenado el usuario:

<input type="submit" name="Entrar" value="Entrar" /> <input type="reset" name="Resetear" value="Resetear" />	Entrar Resetear

Podemos cambiar el texto que el navegador pone por defecto en esos botones utilizando el parámetro VALUE.

"BUTTON" crea un botón.

"FILE" sirve para crear un cuadro de diálogo mediante el que enviar un archivo desde tu disco duro. No está soportado en todos los navegadores.

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA
U. N. A. M.

2007

CURSO: CI187	DESARROLLO DE COMPONENTES WEB CON JAVA		
INSTRUCTOR:	ING. ALEJANDRO VELÁZQUEZ MENA	PERIODO	/ HORARIO
INSTITUCIÓN:	FIDEICOMISO FONDO NACIONAL DE HABITACIONES POPULARES	Lunes, 17 de Septiembre de 2007	17:00
SEDE:	INSURGENTES SUR NO. 3483	Martes, 25 de Septiembre de 2007	20:00

HOJA DE REGISTRO:

DATOS PERSONALES										
Escolaridad:	Primaria:		Secundaria:		Preparatoria:		Profesional:		Otros estudios:	
Teléfono:		Domicilio:	Calle y Número:							
Colonia:		Código Postal:		Delegación ó Municipio:						

DATOS LABORALES								
Apellido Paterno:		Apellido Materno:		Nombre(s):				
R. F. C.:				Sexo:	Femenino:		Masculino:	
Área de Adscripción Real:								
Puesto:				Antigüedad:				
Tipo de Puesto:	Base:		Confianza:		Honorarios:		Otro:	
Tipo de Personal:	Directivo:		Administrativo:		Técnico:		Secretarial:	
Teléfono:		Domicilio:	Calle y Número:					
Colonia:		Código Postal:		Delegación ó Municipio:				

____

Firma