CAPÍTULO 10

Servlets con acceso a bases de datos

10.1. Acceso a bases de datos mediante *servlets* y JDBC

Una de las tareas más importantes y más frecuentemente realizadas por los ***servlets*** es la conexión a ***bases de datos*** mediante ***JDBC***. Esto es debido a que los ***servlets*** son un componente ideal para hacer las funciones de capa media en un sistema con una arquitectura de tres capas como la mostrada en la Figura 10.1.

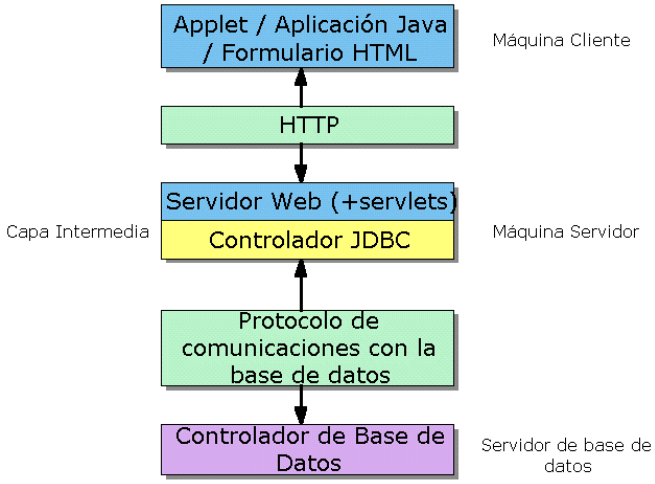

Figura 10.1. Arquitectura cliente-servidor de 3 capas.

Este modelo presenta la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan contra la base de datos, y además, está la ventaja adicional de que los *drivers* ***JDBC*** no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de *driver*. En cualquier caso, tanto el ***Servidor HTTP*** como el ***Servidor de Base de Datos*** pueden estar en la misma máquina, aunque en sistemas empresariales de cierta importancia esto no suele ocurrir con frecuencia.

Los ejemplos que se presentan en este capítulo hacen uso de ***JDBC*** y ***SQL*** de una forma muy simple, pero perfectamente válida para lo que se pretende de los ***servlets*** en este documento.

La arquitectura de los ***servlets*** hace que la escritura de aplicaciones que se ejecuten en el servidor sea relativamente sencilla y que sean aplicaciones muy robustas. La principal ventaja de utilizar ***servlets*** es que se puede programar sin dificultad la información que va a proporcionar entre peticiones del cliente. Es decir, se puede tener constancia de lo que el usuario ha hecho en peticiones anteriores. Además, cada objeto del ***servlet*** se ejecuta dentro de un ***thread*** de ***Java***, por lo que se pueden controlar las interacciones entre múltiples objetos; y al utilizar el identificador de sincronización, se puede asegurar que los ***servlets*** del mismo tipo esperan a que se produzca la misma transacción, antes de procesar la petición; esto puede ser especialmente útil cuando mucha gente intenta actualizar al mismo tiempo la base de datos, o si hay mucha gente pendiente de consultas a la base de datos cuando ésta está en pleno proceso de actualización.

En efecto, las características ***multithread*** de los ***servlets*** hacen que se adecuen perfectamente a este tipo de tareas. Si el servidor de base de datos incluye soporte para múltiples conexiones simultáneas,

incluso es posible establecer éstas una única vez e ir administrando las conexiones entre las sucesivas peticiones de servicio.

10.2. Ejemplo 1: Escribir en una base de datos tipo Access

El siguiente ejemplo muestra cómo compartir una única conexión entre todas las peticiones de servicio. Para ello, se retoma el ejemplo introductorio del capítulo anterior, pero en lugar de mostrar en pantalla la información recogida en el formulario, se introducirá en una base de datos.

Para poder ejecutar este ejemplo en los PCs de las Salas de Tecnun (o en el propio domicilio) se deben seguir los siguientes pasos:

1. Se deberá disponer de ***Microsoft Access***. Con dicho programa se debe crear una nueva base de datos, llamada por ejemplo ***ServletOpinion2.mdb***. En dicho fichero habrá que crear una tabla vacía llamada ***Opiniones_Recogidas*** cuyos campos se llamen ***nombre***, ***apellidos***, ***opinion*** y ***comentarios***, que sean respectivamente de tipo ***text***, ***text***, ***text*** y ***memo***.
2. A continuación se debe crear un Data Source Name (DSN) con el nombre "***opinion***" mediante el administrador de ODBC y asignarle la base de datos anterior.
3. Abrir una consola de ***MS-DOS*** y arrancar ***servletrunner.exe***. Se supone que la variable ***CLASSPATH*** tiene ya un valor correcto.
4. El siguiente fichero contiene la clase ***ServletOpinion2*** que se conectará con la base de datos mencionada escribiendo en ella las opiniones de los usuarios. Dichas opiniones pueden ser recogidas con el fichero ***MiServlet2.html*** que se incluye a continuación de ***ServletOpinion2.java***. Es importante prestar atención a las ***líneas en negrita***, en las que se concentran los conceptos fundamentales de este ejemplo.

```
// fichero ServletOpinion2.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class ServletOpinion2 extends HttpServlet {

  // Declaración de variables miembro
  private String nombre = null;
  private String apellidos = null;
  private String opinion = null;
  private String comentarios = null;

  // Referencia a un objeto de la interface java.sql.Connection
  Connection conn = null;

  // Este método se ejecuta una única vez (al ser inicializado el servlet
  // por primera vez)
  // Se suelen inicializar variables y ejecutar operaciones costosas en tiempo
  // de ejecución (abrir ficheros, conectar con bases de datos, etc)
  public void init (ServletConfig config) throws ServletException {

    // Llamada al método init() de la superclase (GenericServlet)
    // Así se asegura una correcta inicialización del servlet
    super.init(config);

    // dsn (Data Source Name) de la base de datos
    String dsn = new String("jdbc:odbc:opinion");

    // Carga del Driver del puente JDBC-ODBC
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch(ClassNotFoundException ex) {
      System.out.println("Error al cargar el driver");
      System.out.println(ex.getMessage());
    }
```

```
    // Establecimiento de la conexión con la base de datos
    try {
      conn = DriverManager.getConnection(dsn, "", "");
    } catch (SQLException sqlEx) {
      System.out.println("Se ha producido un error al" +
                         " establecer la conexión con: " + dsn);
      System.out.println(sqlEx.getMessage());
    }

    System.out.println("Iniciando ServletOpinion (version BD)...");
  } // fin del método init()

  // Este método es llamado por el servidor web al "apagarse" (shut down).
  // Sirve para proporcionar una correcta desconexión de una base de datos,
  // cerrar ficheros abiertos, etc.
  public void destroy () {
    super.destroy();
    System.out.println("Cerrando conexion...");
    try {
      conn.close();
    } catch(SQLException ex){
      System.out.println("No se pudo cerrar la conexion");
      System.out.println(ex.getMessage());
    }
  } // fin del método destroy()

  // Método de llamada mediante un HTTP POST
  public void doPost (HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {

    boolean hayError = false;

    // adquisición de los valores del formulario
    if(req.getParameter("nombre")!=null)
      nombre = req.getParameter("nombre");
    else
      hayError=true;

    if(req.getParameter("apellidos")!=null)
      apellidos = req.getParameter("apellidos");
    else
      hayError = true;

    if(req.getParameter("opinion")!=null)
      opinion = req.getParameter("opinion");
    else
      hayError = true;

    if(req.getParameter("comentarios")!=null)
      comentarios = req.getParameter("comentarios");
    else
      hayError = true;

    // Mandar al usuario los valores adquiridos (Si no se ha producido error)
    if(!hayError) {
      if (actualizarBaseDeDatos() == 0)
        devolverPaginaHTML(resp);
      else
        resp.sendError(500, "Se ha producido un error"+
                       " al actualizar la base de datos");
    } else
      resp.sendError(500, "Se ha producido un error"+
                     " en la adquisición de parámetros");
  } // fin doPost()
```

```
public int actualizarBaseDeDatos() {
  // crear un statement de SQL
  Statement stmt=null;
  int numeroFilasActualizadas=0;

  // Ejecución del query de actualización de la base de datos
  try {
    stmt = conn.createStatement();
    numeroFilasActualizadas = stmt.executeUpdate("INSERT INTO"+
                              " Opiniones_Recogidas VALUES"+
                             "('"+nombre+"','"+apellidos+"','"+opinion+
                             "','"+comentarios+"')");
    if(numeroFilasActualizadas!=1) return -1;
  } catch (SQLException sql) {
    System.out.println("Se produjo un error creando Statement");
    System.out.println(sql.getMessage());
    return -2;
  } finally {
    // Se cierra el Statement
    if(stmt!=null) {
      try {
        stmt.close();
      } catch(SQLException e){
        System.out.println("Error cerrando Statement");
        System.out.println(e.getMessage());
        return -3;
      }
    }
    return 0;
  } // fin finally
} // fin método actualizarBaseDeDatos()

public void devolverPaginaHTML(HttpServletResponse resp) {
  // Se obtiene un PrintWriter donde escribir (sólo para mandar texto)
  PrintWriter out=null;
  try {
    out=resp.getWriter();
  } catch (IOException io) {
    System.out.println("Se ha producido una excepcion");
  }

  // Se establece el tipo de contenido MIME de la respuesta
  resp.setContentType("text/html");

  // Se mandan los valores
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Valores recogidos en el formulario</title>");
  out.println("</head>");
  out.println("<body>");
  out.println("<b><font size=+2>Valores recogidos del");
  out.println("formulario: </font></b>");
  out.println("<p><font size=+1><b>Nombre: </b>"+nombre+"</font>");
  out.println("<br><fontsize=+1><b>Apellido : </b>"
              +apellidos+"</font><b><font size=+1></font></b>");
  out.println("<p><font size=+1><b>Opini&oacute;n: "+
              "</b><i>"+opinion+"</i></font>");
  out.println("<br><font size=+1><b>Comentarios: </b>"
              +comentarios+"</font>");
  out.println("<P><HR><CENTER><H2>Valores actualizados "+
              "con éxito</CENTER>");
  out.println("</body>");
  out.println("</html>");
```

```
    // Se fuerza la descarga del buffer y se cierra el PrintWriter
    out.flush();
    out.close();
  } // fin de devolverPaginaHTML()

  // Función que permite al servidor web obtener una pequeña descripción del
  // servlet, qué cometido tiene, nombre del autor, comentarios adicionales...
  public String getServletInfo() {
    return "Este servlet lee los datos de un formulario "+
           "y los introduce en una base da datos";
  } // fin de getServletInfo()

} // fin de la clase servletOpinion2
```

```
<!-- Fichero MiServlet2.htm -->
<HTML>
<HEAD><TITLE>Envíe su opinión</TITLE></HEAD>
<BODY>
<H2>Por favor, envíenos su opinión acerca de este sitio web</H2>

<FORM ACTION="http://localhost:8080/servlet/ServletOpinion2" METHOD="POST">
    Nombre: <INPUT TYPE="TEXT" NAME="nombre" SIZE=15><BR>
    Apellidos: <INPUT TYPE="TEXT" NAME="apellidos" SIZE=30><P>

    Opinión que le ha merecido este sitio web<BR>
    <INPUT TYPE="RADIO" CHECKED NAME="opinion" VALUE="Buena">Buena<BR>
    <INPUT TYPE="RADIO" NAME="opinion" VALUE="Regular">Regular<BR>
    <INPUT TYPE="RADIO" NAME="opinion" VALUE="Mala">Mala<P>

    Comentarios <BR>
    <TEXTAREA NAME="comentarios" ROWS=6 COLS=40>
    </TEXTAREA><P>

    <INPUT TYPE="SUBMIT" NAME="botonEnviar" VALUE="Enviar">
    <INPUT TYPE="RESET" NAME="botonLimpiar" VALUE="Limpiar">
</FORM>

</BODY>
</HTML>
```

Nota importante: En este fichero el valor del parámetro ***ACTION*** depende del ordenador con el que se esté trabajando como servidor de servlets. En el ejemplo se ha supuesto que el propio ordenador local es el servidor de servlets. Si se trata de otro ordenador se puede poner su ***nombre*** o su ***número de IP*** (suponiendo que lo tenga).

Este ***servlet*** tiene como resultado el cambio en la base de datos reflejado en la Figura 10.2 y una pantalla similar a la del ejemplo introductorio en el browser. Cada vez que se ejecuta el formulario de añade una nueva línea a la tabla ***Opiniones_Recogidas***.

Figura 10.2. Resultado obtenido en la base de datos.

En este ejemplo cabe resaltar lo siguiente:

- La conexión con la base de datos se hace por medio de un driver que convierte de ***JDBC*** a ***ODBC***. La carga de dicho *driver* ***JDBC-ODBC*** tiene lugar durante la inicialización del ***servlet***, al igual que la ***conexión*** con la base de datos. Basta pues con una única ***conexión*** a la base de datos para satisfacer todas las peticiones de los clientes (a priori, ya que puede ser necesario tener más conexiones abiertas si se espera un tráfico intenso).
- A su vez, en el método ***destroy()*** se ha implementado la ***desconexión*** de la base de datos, de forma que no quede recurso alguno ocupado una vez que el ***servlet*** haya sido descargado.
- En el método ***doPost()*** se ha comprobado que la adquisición de los parámetros del formulario era correcta, enviando un error al cliente en caso contrario. También podría haberse comprobado que ningún campo estuviera vacío.
- El método ***actualizarBaseDeDatos()*** es el que se encarga de la introducción de los valores en la base de datos, mientras que el método ***devolverPaginaHTML()*** se encarga de la presentación en pantalla de los valores leídos. En la ejecución del ***query*** en ***actualizarBaseDeDatos()*** se tienen muy en cuenta posibles excepciones que pudieran surgir y se devuelve como valor de retorno un código de error distinto de cero para esos casos.
- Asimismo, mediante el bloque ***finally*** se consigue cerrar siempre el ***Statement*** tanto si se produce el error como si no, liberando recursos del sistema de esa manera.
- Téngase especial cuidado con la sintaxis del ***query***, y en especial con las comillas simples en que debe ir envuelto cada ***String***.
- Tras mostrar en pantalla los datos recogidos, se comprueba si se ha producido alguna actualización. Si este es el caso, se informa al cliente de esta circunstancia, mostrando un mensaje de error en caso contrario.

10.3. Ejemplo 2: Consultar una base de datos Tipo Access

Este segundo ejemplo pretende mostrar la forma en que se accede a datos contenidos en bases de datos. Para ello, se ha escrito un programa empleando un ***servlet*** que recibe como parámetro el nombre de un grupo de prácticas (parámetro ***GRUPO***), y muestra la lista de los alumnos que están en ese grupo, así como algunos de sus datos personales (número de carnet, nombre, apellidos, curso).

El usuario selecciona el grupo que quiere ver en el formulario mostrado en la Figura 10.3:

Figura 10.3. Formulario del Ejemplo 2.

El código **HTML** correspondiente a la página mostrada en la Figura 10.3 es el siguiente:

```
<!-- fichero Formulario2.htm -->
<html>
<head>
    <title>Grupos de prácticas</title>
</head>

<body>
<h2 align="center">Escoja el grupo de prácticas cuya lista desea ver</h2>

<form method="GET" action="http://localhost:8080/servlet/ListaAlumnos"
      name="Formulario">
  <div align="center"><center><p>
  <input type="radio" value=1 checked name="GRUPO">Grupo1 
  <input type="radio" name="GRUPO" value=2>Grupo2 
  <input type="radio" name="GRUPO" value=3>Grupo3
  </p></center></div>
  <div align="center"><center><p>
  <input type="submit" value="Enviar" name="BotonEnviar">
  <input type="reset" value="Borrar" name="BotonBorrar">
  </p></center></div>
</form>

</body>
</html>
```

El formulario se compone de tres ***radio buttons*** con los nombres de los grupos disponibles, y de los botones ***Enviar*** y ***Borrar***. Por otra parte, el método ***HTTP*** empleado en este ejemplo ha sido ***HTTP GET***. Por esto, clicando en ***Enviar***, el ***URL*** solicitado al servidor tiene la siguiente forma:

```
http://localhost:8080/servlet/ListaAlumnos?GRUPO=1&BotonEnviar=Enviar
```

Seleccionando el ***Grupo1*** y clicando en ***Enviar***, se obtiene el resultado mostrado en la Figura 10.4.

Figura 10.4. Lista de alumnos del Grupo 1

El ***servlet*** empleado para obtener es bastante simple. He aquí el código del ***servlet***:

```
// fichero ListaAlumnos.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

public class ListaAlumnos extends HttpServlet {

  Connection conn = null;
  // Vector que contendrá los objetos Alumno
  Vector vectorAlumnos=null;

  //Este método se ejecuta una única vez, al ser inicializado por primera vez
  //el servlet.Se suelen inicializar variables y ejecutar operaciones costosas
  //en tiempo de ejecución (abrir ficheros, conectar con bases de datos, etc)
  public void init (ServletConfig config) throws ServletException {

    // Llamada al método init() de la superclase (GenericServlet)
    // Así se asegura una correcta inicialización del servlet
    super.init(config);

    // url de la base de datos
    String url=new String("jdbc:odbc:alumnos");

    // Carga del Driver
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch(ClassNotFoundException ex) {
      System.out.println("Error al cargar el driver");
      System.out.println(ex.getMessage());
    }

    // Establecimiento de la conexión
    try {
      conn=DriverManager.getConnection(url,"","");
    } catch (SQLException sqlEx) {
      System.out.println("Se ha producido un error al establecer "+
                         "la conexión con: "+url);
      System.out.println(sqlEx.getMessage());
    }

    System.out.println("Iniciando ListaAlumnos");
  } // fin del método init()

  // Este método es llamado por el servidor web al "apagarse"
  // (al hacer shut down). Sirve para proporcionar una correcta
  // desconexión de una base de datos, cerrar ficheros abiertos, etc.
  public void destroy () {
    super.destroy();
    System.out.println("Cerrando conexion...");
    try {
      conn.close();
    } catch(SQLException ex){
      System.out.println("No se pudo cerrar la conexion");
      System.out.println(ex.getMessage());
    }
  } // fin de destroy()

  // Método llamada mediante un HTTP GET
  public void doGet (HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException {
    // Obtención del grupo de prácticas
    String grupo = null;
```

```
    grupo = req.getParameter("GRUPO");

    if(grupo==null) {
      resp.sendError(500, "Se ha producido un error en la lectura " +
                     "de la solicitud");
      return;
    }

    //Consulta a la base de datos para obtener la lista de alumnos de un grupo
    if(obtenerLista(grupo)==0) {
      // Mostrar la lista de alumnos mediante una página HTML
      mostrarListaAlumnos(resp, grupo);
    }
    else if(obtenerLista(grupo)==-3) {
      resp.sendError(500, "No se ha encontrado el grupo: " +grupo);
    }
    else
      resp.sendError(500, "Se ha producido un error en el acceso " +
                     "a la base de datos");
  } // fin del método doGet()

  public int obtenerLista(String grupo) {

    Statement stmt = null;
    ResultSet rs = null;

    // Ejecución del query
    try {
      stmt=conn.createStatement();
      rs = stmt.executeQuery("SELECT DISTINCT Nombre, Apellidos, Curso, " +
           "Carnet, GrupoPractica FROM TablaAlumnos " +
           "WHERE GrupoPractica=" + grupo + " ORDER BY Carnet");

      vectorAlumnos=new Vector();

      // Lectura del ResultSet, en Java 2
      while (rs.next()) {
        Alumno temp=new Alumno();
        temp.setNombre(rs.getString("Nombre"));
        temp.setApellidos(rs.getString("Apellidos"));
        temp.setCarnet(rs.getLong("Carnet"));
        temp.setCurso(rs.getInt("Curso"));
        vectorAlumnos.addElement(temp);
      }
      // En el JDK 1.1.x hay un bug que hace que no se lean bien los valores
      // que no son Strings. Sería de la siguiente forma:
      /* while (rs.next()) {
           Alumno temp=new Alumno();
           temp.setNombre(rs.getString("Nombre"));
           temp.setApellidos(rs.getString("Apellidos"));
           temp.setCarnet(java.lang.Long.parseLong(rs.getString("Carnet")));
           temp.setCurso(java.lang.Integer.parseInt(rs.getString("Curso")));
           vectorAlumnos.addElement(temp);
         }*/

      if(vectorAlumnos.size()==0) return -3;

      return 0;

    } catch (SQLException sql) {
      System.out.println("Se produjo un error al crear el Statement");
      System.out.println(sql.getMessage());
      return -1;
    } finally {
```

```
      // se cierra el Statment
      if(stmt!=null) {
        try {
          stmt.close();
        } catch(SQLException e) {
          System.out.println("Error al cerrar el Statement");
          System.out.println(e.getMessage());
          return -2;
        }
      }
    } // fin del finally
} // fin del método obtenerLista()

public void mostrarListaAlumnos(HttpServletResponse resp, String grupo) {

    // se establece el tipo de contenido MIME de la respuesta
    resp.setContentType("text/html");

    // se obtiene un PrintWriter donde escribir (sólo para mandar texto)
    PrintWriter out=null;
    try {
      out=resp.getWriter();
    } catch (IOException io) {
      System.out.println("Se ha producido una excepcion");
    }

    // se manda la lista
    out.println("<html>");
    out.println("");
    out.println("<head>");
    out.println("<title>Lista de alumnos del grupo "+grupo+"</title>");
    out.println("<meta name=\"GENERATOR\" " +
                "content=\"Microsoft FrontPage 3.0\">");
    out.println("</head>");
    out.println("");
    out.println("<body>");
    out.println("");
    out.println("<H2 align=\"center\">Lista de alumnos del grupo " +
                grupo + "</H2>");
    out.println("<div align=\"center\"><center>");
    out.println("");
    out.println("<table border=\"1\" width=\"70%\">");
    out.println("<tr>");
    out.println("<td width=\"25%\" bgcolor=\"#808080\">"+
                "<font color=\"#FFFFFF\">Carnet</font></td>");
    out.println("<td width=\"25%\" bgcolor=\"#808080\">"+
                "<font color=\"#FFFFFF\">Nombre</font></td>");
    out.println("<td width=\"25%\" bgcolor=\"#808080\">"+
                "<font color=\"#FFFFFF\">Apellidos</font></td>");
    out.println("<td width=\"25%\" bgcolor=\"#808080\">"+
                "<font color=\"#FFFFFF\">Curso</font></td>");
    out.println("</tr>");

    // Datos del Alumno por filas
    Alumno alum=null;
    for (int i=0; i<vectorAlumnos.size();i++) {
      alum=(Alumno)vectorAlumnos.elementAt(i);
      out.println("<tr>");
      out.println("<td width=\"25%\">"+alum.getCarnet()+"</td>");
      out.println("<td width=\"25%\">"+alum.getNombre()+"</td>");
      out.println("<td width=\"25%\">"+alum.getApellidos()+"</td>");
      out.println("<td width=\"25%\">"+alum.getCurso()+"</td>");
      out.println("</tr>");
    }
```

```
    out.println("</table>");
    out.println("</center></div>");
    out.println("</body>");
    out.println("</html>");

    // se fuerza la descarga del buffer y se cierra el PrintWriter
    out.flush();
    out.close();
  } // fin del método mostrarListaAlumnos()

  // Función que permite al servidor web obtener una pequeña descripción del
  // servlet, qué cometido tiene, nombre del autor, comentarios adicionales...
  public String getServletInfo() {
   return "Servlet que muestra en pantalla la lista de un grupo de prácticas";
  }
} // fin de la clase ListaAlumnos
```

Puede observarse que este ***servlet*** es bastante parecido al del ejemplo previo, en cuanto a su forma general. Tiene como el anterior un método ***init()*** donde efectúa la conexión con la base de datos cuyo ***DSN*** es ***alumnos***, y comprueba que la conexión se ha realizado con éxito. Asimismo, tiene un método ***destroy()*** donde se cierra dicha conexión y se avisa de posibles eventualidades.

Sin embargo, a diferencia del ***servlet*** del ejemplo anterior, la petición del cliente es de tipo ***HTTP GET***, por lo que se ha redefinido el método ***doGet()*** y no el ***doPost()*** como en el caso anterior. En este método, lo primero que se hace es leer el parámetro ***GRUPO*** dentro del método ***doGet()***. En caso de que haya algún problema en la lectura de dicho parámetro, lanza un mensaje de error.

Una vez que se sabe cuál es el grupo cuya lista quiere visualizar el cliente, se llama al método ***obtenerLista(String)***, que tiene como parámetro precisamente el nombre del grupo a mostrar. En este método se realiza la consulta con la base de datos, mediante el método ***executeQuery()*** de la interface ***Statement***.

La ejecución de dicho método tiene como resultado un objeto ***ResultSet***, que es como una especie de ***tabla*** cuyas ***filas*** son cada uno de los alumnos, y cuyas ***columnas*** son los datos solicitados de ese alumno (***carnet***, ***nombre***, ***apellidos***, ***curso***, ...). Para obtener dichos datos se emplea el método ***next()*** del objeto ***ResultSet*** en conjunción con los métodos ***getXXX()*** de dicho objeto. En el ***JDK 1.1.x*** existía un ***bug*** por el cual ***Java*** fallaba en determinadas circunstancias al intentar leer valores de campos con métodos distintos del ***getString()*** (por ejemplo: ***getLong()***, ***getInt()***, etc.). Esto puede ser solventado empleando el método ***getString()*** y después convirtiendo el ***String*** resultante a los distintos tipos primitivos deseados (***int***, ***long***, etc.). En ***Java 2*** dicho ***bug*** ha sido corregido, y pueden emplearse directamente los métodos adecuados sin ningún problema.

En este ejemplo, además, al leer los valores de la base de datos, estos han sido almacenados en un ***Vector*** de objetos de la clase ***Alumno***, que ha sido creada para este ejemplo. Esta clase lo único que hace es definir las variables que van a contener los datos de cada alumno, y establecer los métodos de introducción y recuperación de dichos datos.

```
public class Alumno {
    // Definición de variables miembro
    private String nombre;
    private String apellidos;
    private long carnet;
    private int curso;
    private String grupoPractica;

    // Métodos para establecer los datos
    public void setNombre(String nom) { nombre=nom; }
    public void setApellidos(String apel) { apellidos=apel; }
    public void setCarnet(long carn) { carnet=carn; }
    public void setCurso(int cur) { curso=cur; }
    public void setGrupoPractica(String grupo) { grupoPractica=grupo; }

    // Métodos de recuperación de datos
    public String getNombre() { return nombre; }
    public String getApellidos() { return apellidos; }
```

```
    public long getCarnet() { return carnet; }
    public int getCurso() { return curso; }
    public String getGrupoPractica() { return grupoPractica; }
} // fin de la clase Alumno
```

Siguiendo con el método ***obtenerLista(String)***, su función es comprobar que se ha obtenido al menos una fila como resultado de la consulta y capturar posibles excepciones, retornando los correspondientes códigos de error si fuera necesario.

Por último, volviendo al método ***doGet()*** que se estaba describiendo, sólo queda llamar al método ***mostrarListaAlumnos(HttpServletResponse resp, String grupo)***, en caso de que no se haya producido ningún error. Este método es análogo al método ***devolverPaginaHTML()*** del ejemplo anterior. En este caso, muestra una tabla ***HTML*** que va construyendo dinámicamente, añadiendo tantas filas a la misma como alumnos estén contenidos en el vector de alumnos.