Práctica 3

PROGRAMACIÓN WEB CON SERVLETS

Tabla de contenidos

1. Introducción a los Servlets 1
   1.1. Características de los Servlets 1
   1.2. API de los Servlets 2
2. Servlets HTTP 3
   2.1. Características de los Servlets HTTP 3
   2.2. Estructura de un Servlet HTTP 4
   2.3. Requisitos para probar servlets 4
3. El primer Servlet 4
   3.1. Codificación 4
   3.2. Prueba del Servlet 6
4. Manejo de peticiones y respuestas 7
   4.1. Petición (*request*) al Servlet HTTP 7
   4.2. Respuesta (*response*) del Servlet HTTP 7
   4.3. Ejemplo de modelo petición/respuesta 7
5. Ejemplo con acceso a datos 9
   5.1. Página HTML que realiza la petición al servlet 9
   5.2. Definición de la base de datos 9
   5.3. Servlet que atiende las peticiones 10
6. Gestión de sesiones HTTP 11
   6.1. Métodos del objeto Session 12
   6.2. Ejemplo de sesiones 13
7. Ampliación de la práctica 14

1. Introducción a los Servlets

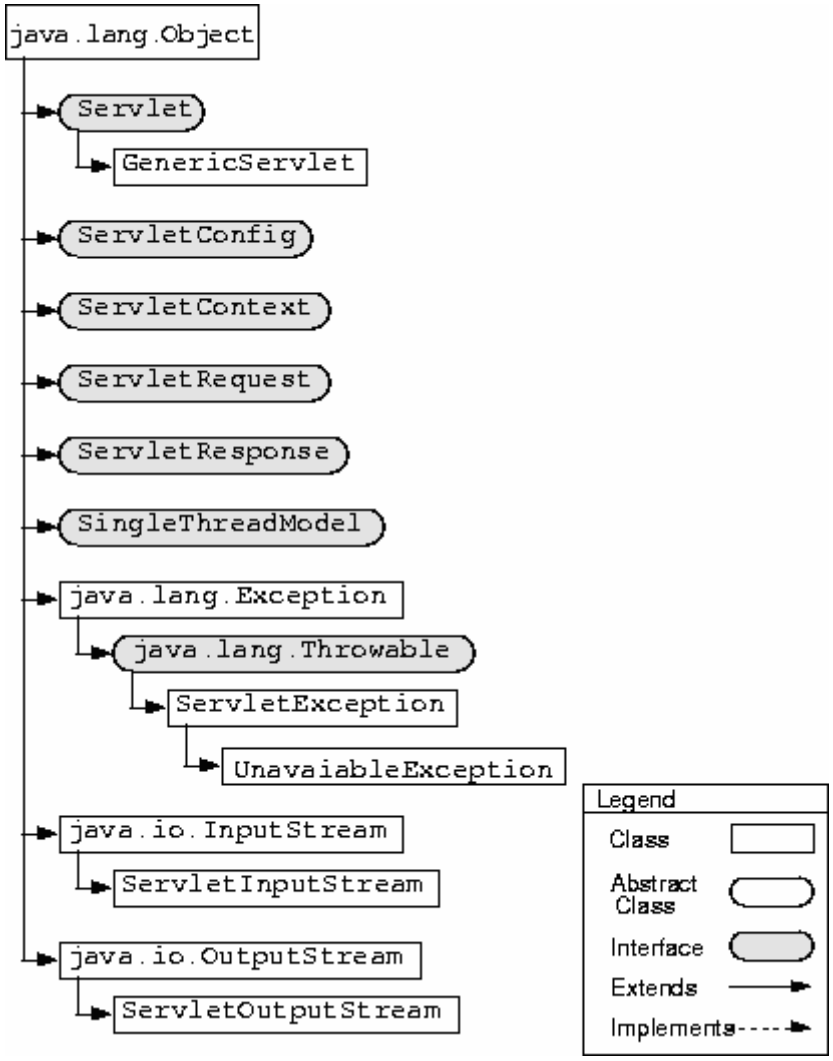
1.1. Características de los Servlets

- Los servlets se utilizan para extender o implementar funcionalidades en un servidor.
- Son módulos de software que se ejecutan dentro del entorno de un servidor y proveen servicios de tipo petición/respuesta.
- Son componentes escritos en Java, situados en los servidores e independientes de cualquier protocolo y de cualquier plataforma.
- Son los equivalentes en los servidores a los applets en el cliente. Sin embargo, a diferencia de los applets, los servlets no tienen representación gráfica.
- Los servlets pueden estar en muchos tipos de servidores. Pero su uso más común es en servidores web. Existen muchos servidores web que soportan el uso de servlets (JWS, Tomcat, JRun, Websphere, etc.).
- Un servlet se instancia la primera vez y se mantiene en memoria esperando nuevas invocaciones (el servidor web tiene una máquina virtual java que es la que ejecuta el servlet).

- La comunicación entre servlets es fácil porque ya están en memoria. El servidor web te da el mecanismo para que se comuniquen entre ellos.

1.2. API de los Servlets

Las clases relacionadas con los Servlets están en el paquete javax.servlet que puede obtenerse a través de la instalación de J2EE o ya viene incorporado en algunos servidores web.

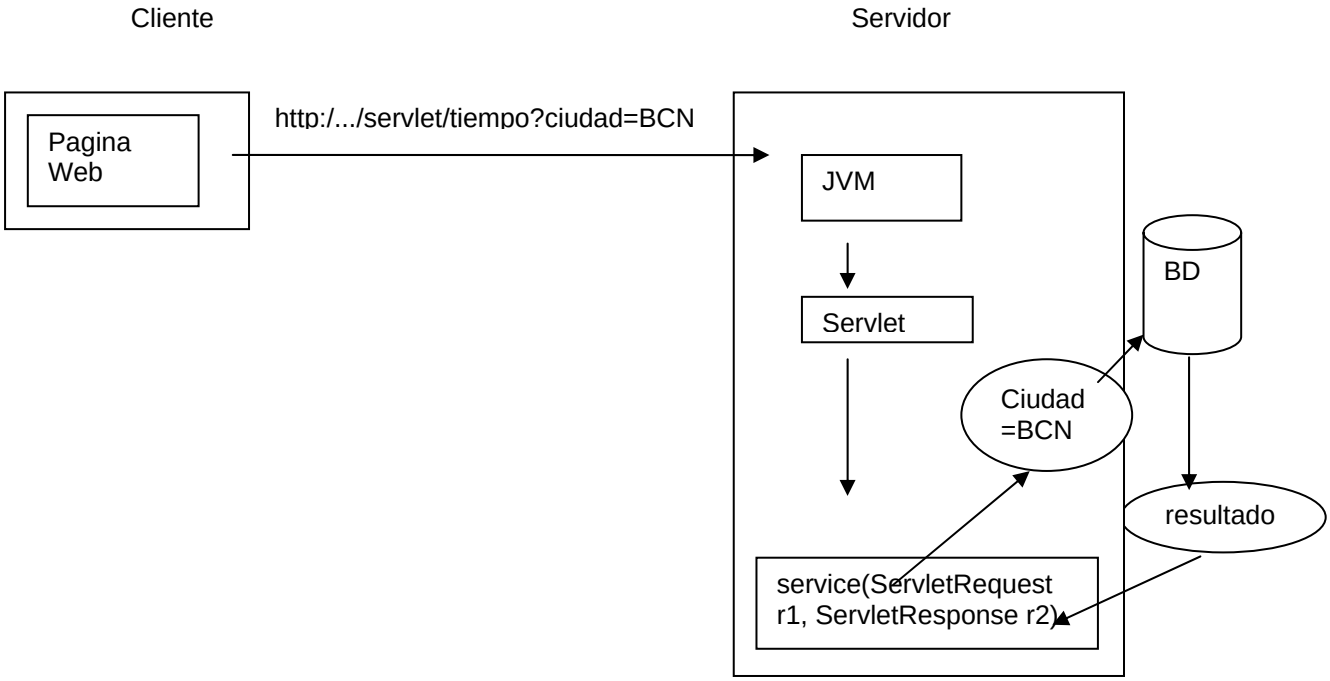


Un servlet debe implementar la interfaz Servlet, la cual declara métodos que administran el servlet y su comunicación con el cliente. También se puede extender la clase GenericServlet que implementa la interfaz Servlet de forma predeterminada.

Las clases ServletRequest y ServletResponse se encargan de recibir la información de petición del cliente y enviar la respuesta al mismo.

Las clases ServletInputStream y ServletOutputStream se encargan de manejar los flujos de entrada de la petición y de salida de la respuesta respectivamente.

A continuación mostramos el funcionamiento de un servlet genérico solicitado a través de un navegador.



2. Servlets HTTP

2.1. Características de los Servlets HTTP

- Los servlets http se cargan directamente en el servidor web y ofrecen acceso interactivo a bases de datos y sistemas propietarios.
- API de los Servlets http (paquete *javax.servlet.http*)
    - *HttpServlet* (clase que hereda de *GenericServlet*)
    - *HttpServletRequest* (hereda de *ServletRequest*)
    - *HttpServletResponse* (hereda de *ServletResponse*)
- Un servlet http puede extender la clase *HttpServlet*. El método service no hay que programarlo directamente, pues éste en su implementación llama a los métodos doGet o doPost para las dos formas de enviar información a través de http.
- El método service de HttpServlet haría lo siguiente:

```
service(req, res) {
Si es Get
   doGet()
Si es Post
   doPost()
```

- GET y POST son formas de enviar información a través del protocolo http.
- GET envía dentro de la URL los parámetros para el servlet. GET permite que se llame al servlet directamente pasándole los parámetros a través de la URL.
- POST encapsula los parámetros en la trama que se envía por el protocolo http. Para enviar datos a través de POST es necesario tener una página web con un formulario que se encargue de enviar la información vía POST. Esta forma también se podría utilizar para GET.
- Existen otras peticiones que se pueden hacer: PUT (permite al cliente ubicar un fichero en el servidor, es similar a enviar un fichero por ftp) y DELETE (permite al cliente eliminar un documento o paquete del servidor web).
- A través de las clases *HttpServletRequest* derivada de *ServletRequest* y *HttpServletResponse* derivada de *ServletResponse* se manipula la información recibida del cliente y la respuesta que se le enviará vía HTTP.

2.2. Estructura de un Servlet HTTP

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Prueba extends HttpServlet
{
   public void doGet(  HttpServletRequest request,
                       HttpServletResponse response )
      throws ServletException, IOException
   {
      ...
   }
}
```

2.3. Requisitos para probar servlets

- Cualquier servidor web que soporte Servlets (Apache Tomcat 5.X, se encuentra en la dirección http://tomcat.apache.org/)
- Instalar Web Tools Platform (WTP) plugin a eclipse, o descargar la versión de eclipse que lo incluye. [1]
- Poner el API de los servlets y de los JSP en el CLASSPATH. Si estamos trabajando con Eclipse, es necesario ponerlos como jars externos del proyecto (verificar que jsp-api y servlet-api están incluidos como jar externos).
- Para instalar el Apache Tomcat 5.5
    - Bajarse el fichero de http://tomcat.apache.org/download-55.cgi
    - Instalación normal (Todo por defecto)
        - Instale también los ejemplos
    - Probar que funciona mediante la siguiente ruta: http://127.0.0.1:8080
        - Puede visualizar los ejemplos desde: http://localhost:8080/examples/
    - Para publicar en TomCat un proyecto, es necesario exportar el fichero WAR a la carpeta: `dir_instalación\webapps\`
    - Reiniciar el Tomcat (normalmente ya esta en ejecución)

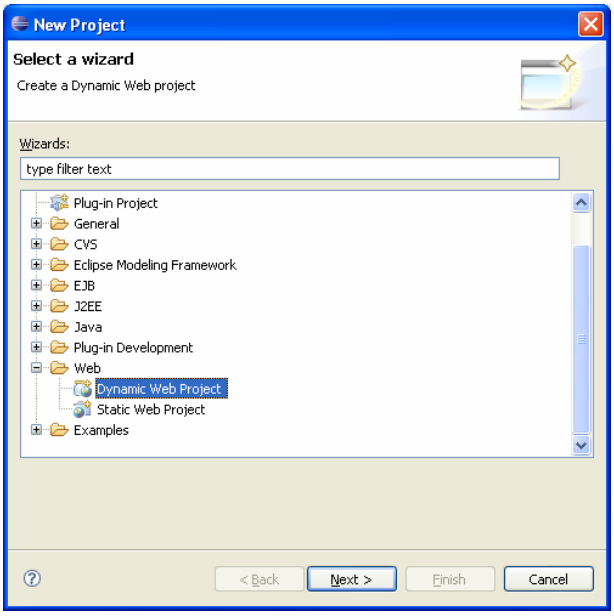
3. El primer Servlet

3.1. Codificación

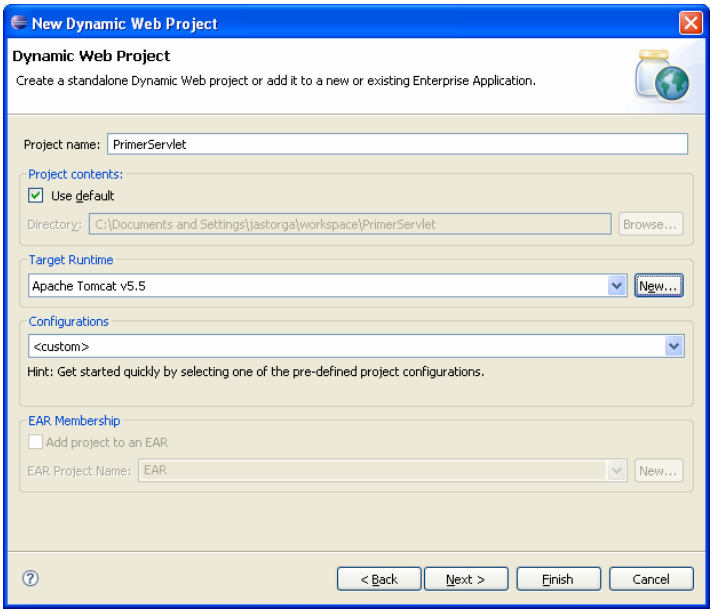
Este servlet sería el ejemplo más sencillo en el cual se solicita el servlet directamente desde el cliente y una vez invocado, el servlet devuelve una respuesta al cliente en forma de una página HTML.

- Lo primero que tenemos que hacer es abrir en Eclipse un nuevo proyecto *Web* (*Dinamic Web Project*).

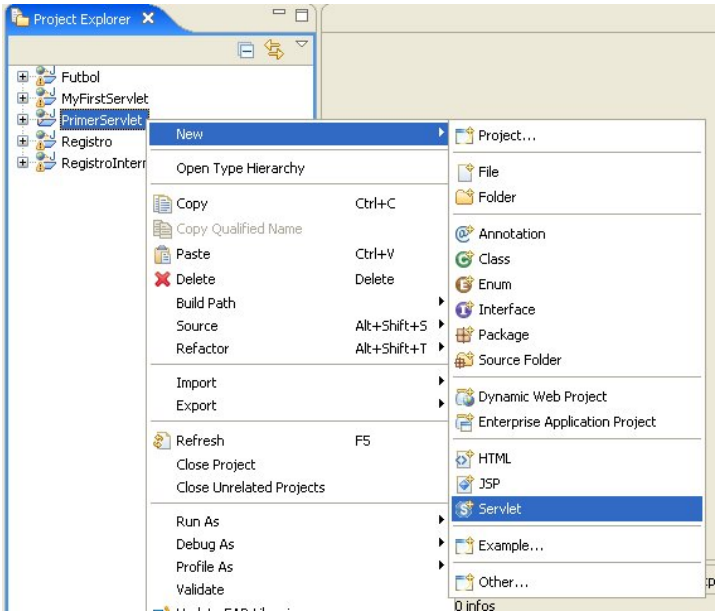
[1] http://www.eclipse.org/webtools/main.php

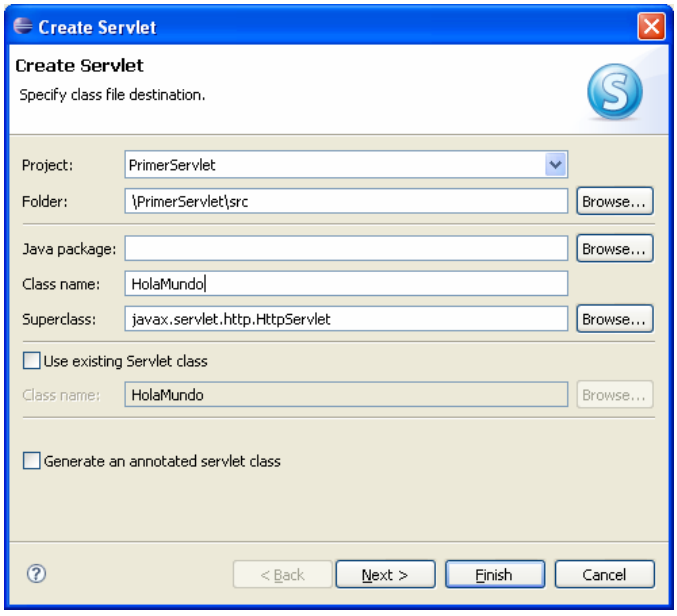


- El proyecto lo llamaremos PrimerServlet.
- En la opción *Target Runtime* elegimos Apache Tomcat v5.5
    - Si no aparece, seleccionar *New* y elegirlo de la lista.



- Las opciones siguientes, dejarlas con los valores por defecto.
- Una vez en el entorno del nuevo proyecto, agregar un Servlet que llamaremos HolaMundo y que NO pertenece a un paquete.





- Escribimos el siguiente código en la clase HolaMundo:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HolaMundo extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        PrintWriter out; //variable de la clase
        //PrintWriter=flujo de caracteres

        res.setContentType("text/html"); //MIMEtype
        out=res.getWriter(); //flujo de salida que va al cliente
        out.println("<html><head><title>Hola Mundo</title></head>");
        out.println("<body><h1>Mi Primer Servlet</h1></body>");
        out.println("</html>");
    }
}
```

- Con esto ya tenemos el primer Servlet codificado.

3.2. Prueba del Servlet

Para probar un servlet lo primero que se necesita es publicarlo en el servidor web. En el caso de utilizar Apache Tomcat y Eclipse, esto se haría exportando el proyecto como Web como un archivo WAR dentro del directorio `dir_instalación\webapps\` del servidor.

Para que el servidor convierta el WAR en una carpeta a servir, es necesario reiniciar el servicio.

Una vez publicado y estando en funcionamiento el servidor web, se puede invocar el servlet directamente desde un navegador. La sintaxis general es:

```
http://servidor:puerto/servlet/NombreServlet
```

En el caso de nuestro ejemplo, si el servidor lo tenemos en local y el puerto de comunicación con el mismo es el 8080, sería:

```
http://localhost:8080/PrimerServlet/HolaMundo
```

Descriptor de despliegue

El contenedor de servlets recibe la petición a `/servlet/HolaMundo` y despliega una instancia de HolaMundo.class, dando inicio a su ciclo de vida. En los *descriptores de despliegue* de Tomcat (que son ficheros llamados `web.xml`) es donde se indica el servlet que recibirá la petición cuando llegue una petición http con URL acabada en `/servlet/HolaMundo`. Suele haber un descriptor de despliegue por cada aplicación web alojada un mismo contenedor, además de uno global situado en `dir_instalación\conf\web.xml`.

4. Manejo de peticiones y respuestas

4.1. Petición (*request*) al Servlet HTTP

Cuando se implementa la interfaz *HttpServletRequest* se obtiene acceso a la siguiente información específica de http:

- Encabezado http
- Información de sesión

Formas de leer datos de formularios enviados por el cliente

- Métodos *getParameter* de la interfaz *ServletRequest*. Si el parámetro tiene más de un valor se utiliza *getParameterValues*. Para conocer el nombre de los parámetros tenemos *getParameterNames*.
- Método *getReader* que retorna un *BufferedReader* para leer el cuerpo de la petición (sólo para POST).
- Procesando una referencia a un *stream* de entrada con los datos binarios, a través del método *getInputStream* de la clase *ServletInputStream* (sólo para POST).
- Método *getQueryString* que retorna un String con datos del cliente. Habría que analizar la cadena para identificar los parámetros (sólo para GET).

4.2. Respuesta (*response*) del Servlet HTTP

La interfaz *HttpServletResponse* proporciona métodos para dar formato a *streams* HTML:

- Constantes para los códigos de error http.
- Métodos para añadir campos a un encabezado HTML. Por ejemplo el método *setContentType* sirve para especificar el tipo del contenido.
- Para enviar datos a un cliente web, el objeto Response proporciona un stream. Antes de obtener el *stream* de escritura, se debe llamar al método *setContentType* para especificar el tipo MIME (*Multipurpose Internet Mail Extension*) que indica el formato del dato que será pasado (texto de página HTML, gif, audio, etc).
- Un método que puede utilizarse para mandar información al cliente es *getWriter* que retorna un objeto de tipo *PrintWriter* para escribir textos al cliente.
- Otro método es *getOutputStream* que retorna un *ServletOutputStream* para enviar datos binarios al cliente.
- Cerrar el *Writer* o el *ServletOutputStream* después de enviar la respuesta permite que el servidor web sepa que la respuesta ha sido completada.

4.3. Ejemplo de modelo petición/respuesta

Para poder enviar información al servlet desde el cliente a través del protocolo HTTP, existen dos formas: GET y POST. GET envía dentro de la URL los parámetros para el servlet. GET permite que se llame al servlet directamente pasándole los parámetros a través de la URL.

POST encapsula los parámetros en la trama que se envía por el protocolo http. Para enviar datos a través de POST es necesario tener una página web con un formulario que se encargue de enviar la información vía POST. Esta forma también se podría utilizar para GET.

A continuación mostraremos un ejemplo muy sencillo de una página html que envía información a un servlet y este devuelve otra página como respuesta. La página se llamará Registro.html.

```
<html>
    <head><title>Registro</title></head>
<body>
    <form METHOD="POST" ACTION="./Registro">
       <hr>
       <h2>
       <center>Registro de cliente<br>
       Nombre<INPUT NAME="Nombre" VALUE=""><br>
       Correo<INPUT NAME="Correo" VALUE="" ><br>
       <INPUT  TYPE=Submit Value = "Enviar">
       </center>
       </h2>
       <br>
</body>
</html>
```

El servlet que recibiría la petición de esta página estará programado en Registro.java dentro de un proyecto llamado Registro y tendría el siguiente código:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Registro extends HttpServlet
{
    static final String PARAMETRO_NOMBRE = "Nombre";
    static final String PARAMETRO_CORREO = "Correo";

    public void doPost(HttpServletRequest req, HttpServletResponse res)
       throws ServletException,IOException
    {
       res.setContentType("text/html");
       PrintWriter out=res.getWriter();

       // Leer los parámetros de la petición
       String nomb = req.getParameter(PARAMETRO_NOMBRE);
       String correo = req.getParameter(PARAMETRO_CORREO);

       // Mandar página de respuesta
       out.println("<html>");
       out.println("<head>");
       out.println("<title>Registro</title>");
       out.println("</head>");
       out.println("<body>");
       out.println("Datos del cliente " );
       out.println("<h2>Nombre: "+nomb+"</h2>");
       out.println("<h2>Correo: "+correo+"</h2>");
       out.println("</body>");
       out.println("</html>");
       out.close(); // indica al servidor que se ha terminado de
                    // mandar la información
       }
}
```

Para probar este ejemplo habría que:

1. Exportar el proyecto como se explicó anteriormente
2. Publicar la página html dentro de la carpeta creada para el proyecto en:
   `dir_instalación\webapps\Registro`
3. Cambiar el nombre a la página html por index.html
4. Invocar la página html desde el navegador con:
   `http://localhost:8080/Registro/`

5. Ejemplo con acceso a datos

A continuación desarrollaremos un ejemplo en el que utilizaremos los conceptos mencionados anteriormente y además el acceso a bases de datos con JDBC desde un servlet.

Primeramente tendremos una página HTML futbol.html que se encargará de enviar datos a un servlet TablaVotos.java. Luego desarrollaremos el servlet que se encargará de visualizar información de una base de datos BDJugadores y de su tabla Jugadores. Todo esto se realizará en un nuevo proyecto llamado Futbol.

5.1. Página HTML que realiza la petición al servlet

La página HTML *futbol.html* permitirá que un cliente entre su nombre y correo y solicite las estadística de los votos recibidos hasta el momento por jugadores de fútbol en la Base de Datos de MySQL *BDJugadores* que tendrá una tabla *Jugadores* y otra *RegistroClientes*.

Se debe crear esta BD y sus tablas en *MySQL* con la siguiente estructura:

5.2. Definición de la base de datos

Nombre de la BD: BDJugadores

Tabla Jugadores

Campo	Tipo
Nombre	Varchar (50)
Votos	Integer

Tabla Registro

Campo	Tipo
Nombre	Varchar (50)
Correo	Varchar (30)
Visitas	Integer

La tabla Jugadores se rellenará inicialmente con datos de algunos jugadores y se pondrán a 0 los votos de cada uno. Por ejemplo:

Nombre	Votos
Roberto Carlos	0
Raul	0
Torres	0
Ronaldinho	0
Ronaldo	0
Figo	0
Zidane	0

La página HTML tendría el siguiente código, en el cual se envía la petición al servlet *TablaVotos* a través de un formulario con el método POST y además se envían los datos del cliente que se encuentra visitando la página en esos momentos:

```
<html>
<head>
    <title>Estadísticas de futbol</title>
</head>
<body>
    <center><H1>Estadísticas de Jugadores de Futbol</H1></center>

    <form action="./TablaVotos" method="POST" >
    <p align="left">
       Nombre del Visitante: <input type="text" size="20" name="txtNombre">
       eMail: <input type="text" size="20" name="txtMail">
    </p>
    <p align="left"><input type="submit" name="B1"
                    value="Mostrar Estadisticas">
     <input type="reset" name="B2" value="Reset">
    </p>
    </form>
</body>
</html>
```

5.3. Servlet que atiende las peticiones

A continuación mostramos el código del servlet que atenderá la petición, conectándose a la Base de Datos *BDJugadores* y consultando la tabla *Jugadores* para enviar una tabla con sus datos como página dinámica de respuesta y dándole las gracias al cliente por su visita.

Para que el servlet sea capaz de acceder a las bases de datos, es necesario poner a su alcance la librería de clases del driver JDBC. Se puede copiar el paquete mysql-connector.jar o alguno equivalente en **`dir_instalación\common\lib`**.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class TablaVotos extends HttpServlet
{
	private Connection con;
	private Statement set;
	private ResultSet rs;

	public void init(ServletConfig cfg) throws ServletException
	{
		String URL="jdbc:mysql://localhost/BDJugadores";
		String userName="root";
		String password="admin";

		try
		{
			Class.forName("com.mysql.jdbc.Driver").newInstance();
			con = DriverManager.getConnection(URL, userName,
password);
			System.out.println("Se ha conectado");
		}
		catch(Exception e)
		{
			System.out.println("No se ha conectado");
		}
	}

	public void doPost(HttpServletRequest req, HttpServletResponse res)
		throws ServletException, IOException
	{
		String nombreP=(String)req.getParameter("txtNombre");
```

```
        PrintWriter out = res.getWriter();
                res.setContentType("text/html");
                out.println("<html><head><title>JDBC</title></head>");
                out.println("<body><font size=10>");
                out.println("<h1>Ejemplo de JDBC</h1>");
                out.println("<table border=1>");
                out.println("<tr><td><b>Jugador</b></td><td><b>Votos"
                        + "</b></td></tr>");
                try
                {
                        String nombre;
                        int votos;

                        set = con.createStatement();
                        rs = set.executeQuery("SELECT * FROM Jugadores");
                        while (rs.next())
                        {
                                nombre = rs.getString("Nombre");
                                votos = rs.getInt("Votos");
                                out.println("<tr><td>"+ nombre + "</td><td>"
                                                + votos + "</td></tr>");
                        }
                        rs.close();
                        set.close();
                }
                catch(Exception e)
                {
                        System.out.println("No lee de la tabla");
                }

                out.println("</table>");
                out.println("<h3>Muchas gracias " + nombreP + " por su
visita</h3>");
                out.println("</form></font></body></html>");
                out.close();
        }

        public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException
        {
                try
                {
                        doPost(req,res);
                }
                catch (Exception e){}
        }

        public void destroy()
        {
                try
                {
                        con.close();
                }
                catch (Exception e){}

                super.destroy();
        }
}
```

6. Gestión de sesiones HTTP

HTTP no es persistente. Si queremos utilizar los servlets para administrar datos para usuarios particulares entre múltiples peticiones http es necesario el uso de sesiones.

De modo general, una *session* comienza cuando un cliente abre el navegador y finaliza cuando lo cierra, aunque este comportamiento puede modificarse por programa.

El servidor debe ser capaz de distinguir clientes entre una y otra visita a una página en una misma sesión, para poder almacenar datos por cada cliente.

En el rastreo de una sesión hay que identificar de forma única cada cliente de la página. Esto se logra gestionando cada visita del cliente como una sesión, la cual tiene un ID único y en la cual se pueden almacenar datos asociados con ese cliente.

No podemos construir aplicaciones, tales como compras en línea, sin sesiones, puesto que se debe mantener información del usuario a través de múltiples solicitudes de páginas.

La implementación de la interfaz Session, permite manejar sesiones proporcionando métodos para identificar sesiones y para almacenar y recuperar datos asociados con ellas.

- El método *getSession* de *HttpServerRequest* retorna la sesión actual para el usuario. El parámetro a true indica si se debe crear una nueva sesión si ésta no existe aún.

```
HttpSession s=req.getSession(true);
```

- Todos los datos de la sesión son mantenidos a nivel del motor del servlet y pueden ser compartidos entre servlets. De esta forma se puede tener un grupo de servlets trabajando juntos para servir al mismo cliente de la sesión.
- Una vez que tenemos un objeto Session, éste trabaja como cómo una *HashTable* o diccionario de Java. Podemos poner cualquier objeto arbitrario en la sesión mediante una clave única.

```
s.putValue(CLAVE, objeto);
var = s.getValue(CLAVE);     //CLAVE es una cadena
                             //que identifica al valor
```

- Además de almacenar datos de la aplicación, el objeto sesión contiene muchos métodos para acceder a propiedades de la misma, como por ejemplo el método *getId()* para obtener el identificador de la sesión.

6.1. Métodos del objeto Session

Método	Funcionalidad
public void putValue(string nombre, Object valor)	Guardar valor en la session
public void setAttribute(string nombre, Object valor)	Lo mismo que el anterior, pero a partir de versión 2.2 de API de los servlets
public Object getValue(String nombre)	Obtener valor de la sesion
public Object getAttribute(String nombre)	Lo mismo que el anterior, pero a partir de versión 2.2 de API de los servlets
public void removeValue(String nombre)	Elimina los valores asociados al nombre pasado como argumento.
public void removeAttribute(String nombre)	Lo mismo que el anterior, pero a partir de versión 2.2 de API de los servlets
public String getValueNames()	Devuelve los nombres de las claves asociadas a la sesion.
public Enumeration getAttributeNames()	Lo mismo que el anterior, pero a partir de versión 2.2 de API de los servlets
public boolean isNew()	Devuelve true si el cliente no se encontraba en una sesión. Devuelve false si la sesion ya existía.
public void invalidate()	Invalida la sesión y la desliga de todos los objetos asociados con ella.

6.2. Ejemplo de sesiones

Si entre la página *Registro.html* vista anteriormente y el servlet *Registro.java*, se invocara otro servlet *RegIntermedio.java*, entonces los datos de *Registro.html* no serían accesibles desde *Registro.java*, a menos que se pasaran a través de una sesión. Crearemos un nuevo proyecto llamado RegistroIntermedio. En *Registro.html* sería necesario cambiar el servlet al que se envía la petición y poner *RegIntermedio.class*. El servlet *RegIntermedio.java* tendría que ser codificado de la siguiente forma:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class RegIntermedio extends HttpServlet
{
	static final String CLAVE_NOMBRE="CLAVE.Nombre";
	static final String CLAVE_CORREO="CLAVE.Correo";
	static final String PARAMETRO_NOMBRE = "Nombre";
	static final String PARAMETRO_CORREO = "Correo";

	public void doPost(HttpServletRequest req, HttpServletResponse res)
		throws ServletException,IOException
	{
	// Acceso a la sesion
	HttpSession s = req.getSession(true);
	res.setContentType("text/html");
	PrintWriter out=res.getWriter();

	// Lectura de parámetros del formulario
	String nomb = req.getParameter(PARAMETRO_NOMBRE);
	String correo = req.getParameter(PARAMETRO_CORREO);

	// Si no había nombre, buscamos valores en la sesión
	if (nomb == null)
	{
		// getValue para versiones del API de los servlets anterior a
2.2
		nomb = (String)s.getAttribute(CLAVE_NOMBRE);
		correo = (String)s.getAttribute(CLAVE_CORREO);
	}

	// Guardar elementos en la sesion. putValue para version < 2.2
	s.setAttribute( (CLAVE_NOMBRE), nomb);
	s.setAttribute( (CLAVE_CORREO), correo);

	out.println("<html>");
	out.println("<head>");
	out.println("<title>Registro</title>");
	out.println("</head>");
	out.println("<body>");
	out.println("Datos del cliente " );
	out.println("<h2>Nombre " + nomb + "</h2>");
	out.println("<h2>Correo " + correo + "</h2>");
	out.println("<P><A HREF='./Registro'> Despedida </A></P>");
	out.println("</body>");
	out.println("</html>");
	out.close();
	}
}
```

El servlet *Registro.java*, que ahora sería invocado desde el servlet *RegIntermedio.java* quedaría como:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Registro extends HttpServlet
{
	static final String CLAVE_NOMBRE = "CLAVE.Nombre";
	static final String CLAVE_CORREP = "CLAVE.Correo";

   public void doPost(HttpServletRequest req, HttpServletResponse res)
```

```
            throws ServletException,IOException
      {
            HttpSession s=req.getSession(true);
       res.setContentType("text/html");
       PrintWriter out=res.getWriter();

            String nomb=(String)s.getValue(CLAVE_NOMBRE);

            out.println("<html>");
       out.println("<head>");
       out.println("<title>Registro</title>");
       out.println("</head>");
       out.println("<body>");
       out.println("Contestacion al registro " );
       out.println("<h2>Gracias por visitar esta pagina " + nomb + "</h2>");
out.println("</body>");
       out.println("</html>");
       out.close();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
    {
        try
        {
            doPost(req,res);
        }
        catch (ServletException e)
        {
        }
        catch (IOException e)
        {
        }
    }
}
```

Por últomo, la páginal *Registro.html*, quedaría como:

```
<html>
    <head><title>Registro</title></head>
    <body>
        <form METHOD="POST" ACTION="./RegIntermedio">
            <hr>
             <h2> <center>Registro de cliente<br>
             Nombre<INPUT NAME="Nombre" VALUE=""><br>
             Correo<INPUT NAME="Correo" VALUE="" ><br>
             <INPUT  TYPE=Submit Value = "Enviar">
             </center>
             </h2>
             <br>
    </body>
</html>
```

7. Ampliación de la práctica

Para culminar la práctica se debe desarrollar por parte de los estudiantes las siguientes tareas:

- Modificar *TablaVotos* y la página correspondiente para que una vez mostradas las estadísticas, puedan votar por uno de los jugadores.
    - Una vez que ejerce un voto, volver a mostrar las estadísticas.
- Modificar la página web para que permita mostrar el listado de Jugadores y sus votos por orden alfabético o de votos, y si es por orden de votos, entonces en orden ascendente o descendente, según seleccione el cliente.
- Modificar el servlet *TablaVotos* para que pueda dar respuesta apropiadamente a la petición especificada en la página web.