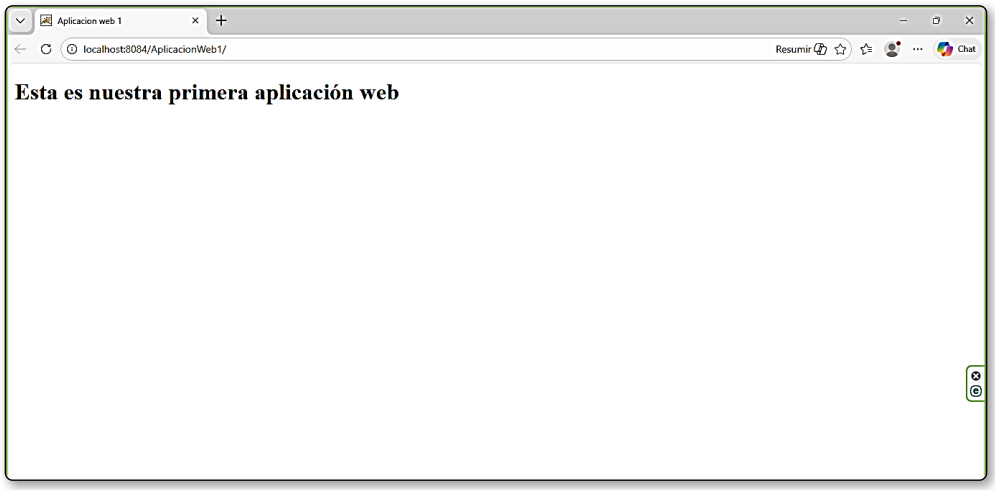
1

INTRODUCCIÓN

1.1 AplicacionWeb1 – VISUALIZACIÓN CONTENIDO HTML

Toda aplicación web necesita estar alojada en un servidor de aplicaciones que responda a las invocaciones a la citada aplicación. Obviamente, en un contexto de producción, dichas invocaciones serán remotas desde navegadores en máquinas cliente. Concretamente, para la redacción de este libro se ha utilizado Tomcat como contenedor de Servlets. Centraremos la atención en dos componentes de Java EE que se ejecutan en la parte de servidor, hablamos de las JSP y sobre todo de los Servlets. A las primeras les daremos un tratamiento accesorio, un compromiso de disponer de un componente para la interacción con el usuario, minimizando la carga de proceso que implementemos en ellas. En Java EE podemos encontrar otras alternativas que cumplan el mismo cometido. No es el caso de los Servlets, que son el "corazón", el núcleo de una aplicación Java EE, y su forma de funcionar es lo que le confiere a Java EE una destacada eficacia en relación con el resto de tecnologías competidoras. Los Servlets serán objeto de tratamiento en el capítulo siguiente. Abordemos en este primer capítulo una aproximación a las JSP.

En primer lugar, antes de entrar en más consideraciones, procedamos a la ejecución de esta primera aplicación ejemplo del libro:



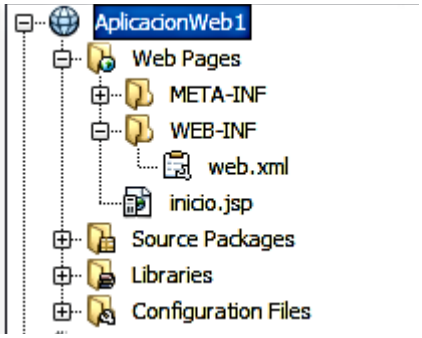
Observamos que su ejecución lleva inherente, por tratarse de una aplicación web, la apertura de una sesión en un navegador de la máquina cliente desde la que es invocada. Dicho navegador permitirá al usuario la interacción con la aplicación. Si centramos nuestra atención en la barra de navegación observaremos que en la URL de invocación a la aplicación

http://localhost:8084/AplicacionWeb1/

nos encontramos:

- **localhost**. Identificación de la máquina donde está desplegada la aplicación. En este caso concreto, por tratarse de un contexto didáctico, coinciden la máquina cliente, desde la que se invoca a la aplicación, y el servidor, donde se encuentra alojada y desplegada la aplicación web. No es necesario matizar que en un contexto real en producción esta coincidencia nunca se recrea. En aplicaciones ejemplo más avanzadas de este libro, aportaremos capturas de pantalla de invocaciones a aplicación desde otra máquina distinta a la que actúa como servidor. Comprobaremos que **localhost** deberá ser sustituido por la IP de dicho servidor.
- **8084**. Puerto desde el que el servidor web, en nuestro caso Apache Tomcat, atiende las peticiones de ejecución de aplicaciones. El puerto habitual para Tomcat es el 8080. Se utiliza 8084 como alternativa cuando, vinculado a un IDE, se puedan producir colisiones con otros productos.
- **AplicacionWeb1**. Es el nombre de la aplicación invocada.

En la siguiente captura de pantalla mostramos el despliegue de componentes de esta aplicación en el IDE empleado.



El código de dichos componentes es:

inicio.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Aplicacion web 1</title>
    </head>
    <body>
        <h1>Esta es nuestra primera aplicación web</h1>
    </body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/
javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>inicio.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Como se observa en la captura de pantalla, la única acción que percibimos de la ejecución de la aplicación es la visualización del texto:

Esta es nuestra primera aplicación web

Del análisis del código de **inicio.jsp** el lector puede llegar a la conclusión errónea de que una JSP viene a ser una HTML, de hecho, su contenido viene a coincidir prácticamente en su totalidad con una HTML. Ello es debido a que el cometido de esta primera aplicación es disponer del ejemplo más sencillo posible de aplicación Java EE. En el próximo apartado ofreceremos al lector la posibilidad de adquirir noción de lo que es una JSP en su verdadera magnitud.

El otro componente que hemos presentado es el fichero **web.xml**. Se trata de un fichero de configuración que contiene información relacionada con el despliegue de la aplicación. En este caso estamos configurando:

- Tiempo de inactividad de la sesión expresado en minutos. Lo establecemos en la etiqueta < *session-timeout* >
- Primer componente que se ejecuta al ser invocada la aplicación. Se establece en la etiqueta <*welcome-file-list*>. Se trata ésta de una etiqueta de grupo a la que están anidadas posibles <welcome-file> en las que se especifica el componente en concreto. Esto nos permite definir varios componentes alternativos, de modo que, si no se encuentra el primero, pasa a buscar el segundo, y así sucesivamente. En este libro solamente se ha especificado uno para cada aplicación.

Más definiciones que nos encontraremos en el **web.xml** en las aplicaciones de este libro será la información de mapeo de los Servlets, circunstancia que será abordada en el siguiente capítulo.

1.2 AplicacionWeb2 - SCRIPTLET

Mantenemos la misma estructura de despliegue que en la anterior. La aportación de esta aplicación consiste en contemplar y presentar la utilización de scriptlets en una JSP, en este caso la inclusión de código Java puro. Podríamos acometer una primera definición simplista de una página JSP diciendo que se trata de un componente, con una estructura HTML, con todo lo que ello comporta, pero que tiene la capacidad de incluir código Java puro mediante un scriptlet, que se ejecuta en el servidor, previo el envío a la máquina cliente, donde se interpretará la parte HTML (etiquetas, javascript, hojas de estilo, …). La ejecución del código Java del scriptlet, además de ejecutar tareas en servidor, puede contribuir a acabar de conformar el contenido HTML puro que es lo que realmente se envía a la máquina cliente con la que está interactuando el servidor. Realmente podríamos incluir en las JSP, si no todo sí la

mayor parte del código Java de una aplicación. El fragmento de código Java puro correspondiente al scriptlet viene delimitado por:

```
<%  SCRIPTLET   %>
```

inicio.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Aplicacion web 2</title>
    </head>
    <body>
        <%  for (int i=1; i<=10; i++)
           {
        %>
             hola <br>
        <% } %>
    </body>
</html>
```

En la siguiente captura de pantalla mostramos lo que estamos tratando:

En la captura de pantalla comprobamos que la visualización producida en el navegador cliente ya no coincide, como en el caso anterior, con la interpretación exacta del contenido HTML. El texto **hola** aparece una sola vez en la JSP. En cambio, en la visualización dicho texto aparece repetido 10 veces. Ello es consecuencia de la repetición a que se ve sometido el texto en cuestión dentro

de la estructura repetitiva *for*, tratándose de código Java puro. Observamos que "salimos" del scriptlet al "mundo" HTML puro inmediatamente después del primer delimitador (*{*). En ese contexto HTML ubicamos el texto susceptible de repetición, y tenemos que "entrar" otra vez al contexto Java puro mediante otro scriptlet para poder ubicar el delimitador de cierre del bloque que se repite (*}*), manteniéndose una total continuidad del código Java implementado de un scriptlet a otro.

1.3 AplicacionWeb3 – INCRUSTACIÓN VALOR EXPRESIONES JAVA

Esta aplicación ejemplo viene a ser una réplica de la anterior. La modificación aplicada consiste en sustituir el literal "hola", sometido a repetición por la estructura repetitiva *for*, por el valor que va adquiriendo en cada repetición la variable de contador *i*. Utilizamos para ello lo que podríamos llamar un "scriptlet de incrustación". Esta variante de scriptlet consiste en "incrustar" en el contenido HTML a enviar a la máquina cliente el resultado de la evaluación de una expresión Java, evaluación que se ejecuta en servidor. La sintaxis a utilizar para ello, es, dentro del scriptlet, y previo a la expresión Java, el símbolo "=":

```
<%=expresión Java%>
```

inicio.jsp

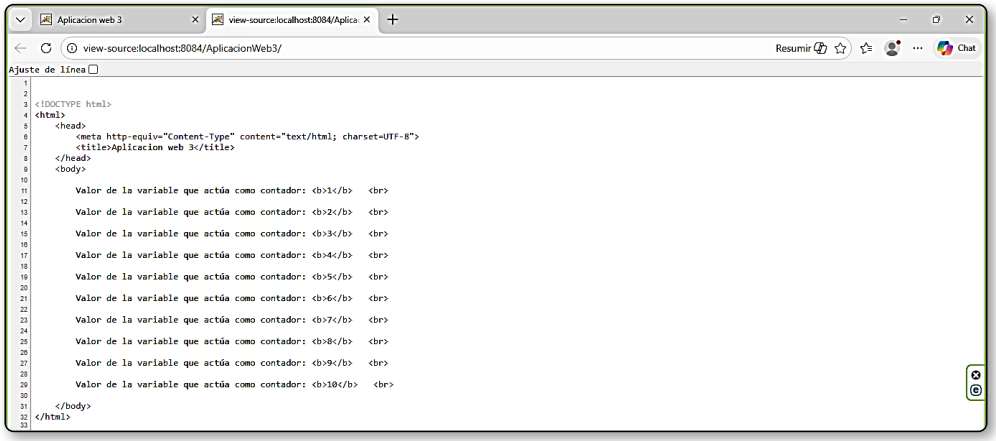
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Aplicacion web 3</title>
    </head>
    <body>
        <%  for (int i=1; i<=10; i++)
           {
        %>
             Valor de la variable que actúa como contador: <b><%=i%></b>   <br
        <% } %>
    </body>
</html>
```

La captura de pantalla que muestra lo descrito:

Con el fin de reforzar el concepto que venimos exponiendo de que:

1. El código Java incluido en el scriptlet se ejecuta en servidor.
2. En el servidor se ejecuta también la evaluación de las expresiones Java de los scriptlets "de incrustación", colaborando también en servidor, en la conformación de la totalidad del HTML.
3. Lo que se envía a la máquina cliente receptora del HTML, es solamente eso, HTML y otros elementos vinculados a ello (javascript, hojas de estilo,). Es decir, solamente elementos que serán objeto de interpretación en el navegador de la máquina cliente.

Presentamos a continuación captura de pantalla obtenida del navegador cliente mediante la opción del mismo "Ver código fuente de página", correspondiente a la anterior:



De modo que en la "parte cliente" pasa inadvertido si los 10 valores que va tomando la variable de control "*i*" de la estructura repetitiva *for* proceden de la ejecución del scriptlet en servidor o si su origen es el resultado de la edición pura de HTML.

1.4 AplicacionWeb4 - DESPROPÓSITO

Realmente podríamos incluir en las JSP, si no todo sí la mayor parte del código Java de una aplicación, objetivo a demostrar en esta aplicación ejemplo, cuya estructura viene siendo análoga a la de las anteriores. Esta actuación es un verdadero despropósito. Todas las recomendaciones y el buen estilo van encaminados a minimizar el código Java a incrustar mediante scriptlets en una JSP. Uno de los objetivos esenciales del mundo HTML es la interacción con el usuario en formularios, menús, etc. Adicionalmente a este cometido, utilizaremos generalmente los scriptlets tan sólo para aportar, en la conformación del HTML, información cuyo origen procede de la ejecución de clases Java en servidor. Esta circunstancia se evidenciará en todas las aplicaciones objeto de tratamiento en el resto del libro. Presentemos a continuación el contenido de esa única JSP, en la que concentramos, de forma totalmente inapropiada como ya hemos mencionado, toda la implementación de código:

- Conexión con la Base de Datos.
- Lanzamiento de la SELECT a la Base de Datos.
- Recepción del resultado en el *ResultSet*.
- Recorrido del *ResultSet*.
- Visualización del contenido del *ResultSet*. Es ésta la única acción que debería ser apropiado realizar en la JSP.

inicio.jsp

```
<%@page import="java.sql.ResultSet"%>
<%@page import="java.sql.Statement"%>
<%@page import="java.sql.DriverManager"%>
<%@page import="java.sql.Connection"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Aplicacion web 4</title>
    </head>
    <body>
```

```
        <%
              //      CONEXION MySQL
              Class.forName("com.mysql.jdbc.Driver");
              Connection connection = DriverManager.getConnection("jdbc:mysql://
localhost:3306/inmobiliaria", "root", "");

              //      CONEXION Oracle

              //  Class.forName("oracle.jdbc.driver.OracleDriver");
              //  Connection connection = DriverManager.getConnection("jdbc:oracle:th
in:@localhost:1521:ORCL2", "scott", "tigre");

              String sql = "SELECT * FROM inmuebles";
              Statement statement = connection.createStatement();
              ResultSet resultSet = statement.executeQuery(sql);
              while (resultSet.next()) {
        %>
                  <b>referencia :          </b><%=resultSet.getString(1)%>   <br>
                  <b>tipo :                </b>
        <%        switch(resultSet.getString(2).charAt(0))
                    { case 'P':%>
                                Piso
        <%                      break;
                    case 'C':%>
                                Casa
        <%                      break;
                    case 'B':%>
                                Bajo comercial
        <%                      break;
                    case 'A':%>
                                Adosado
        <%                      break;
                    case 'G':%>
                                Garaje
        <%                      break;
                    }%>
                  <br>
                  <b>ubicacion :           </b><%=resultSet.getString(3)%>   <br>
                  <b>poblacion :           </b><%=resultSet.getString(4)%>   <br>
                  <b>número dormitorios :  </b><%=resultSet.getInt(5)%>      <br>
                  <b>precio :              </b><%=resultSet.getInt(6)%>      <br>
                  <b>estado :              </b>
        <%        switch(resultSet.getString(7).charAt(0))
                    { case 'D':%>
                                Disponible
        <%                      break;
                    case 'R':%>
                                Reservado
        <%                      break;
                    case 'V':%>
                                Vendido
        <%                      break;
                    }%>
                  <br>
```

```
                    ------------------------------------------------------     <br>
        <%
                }

                connection.close();
        %>
    </body>
</html>
```

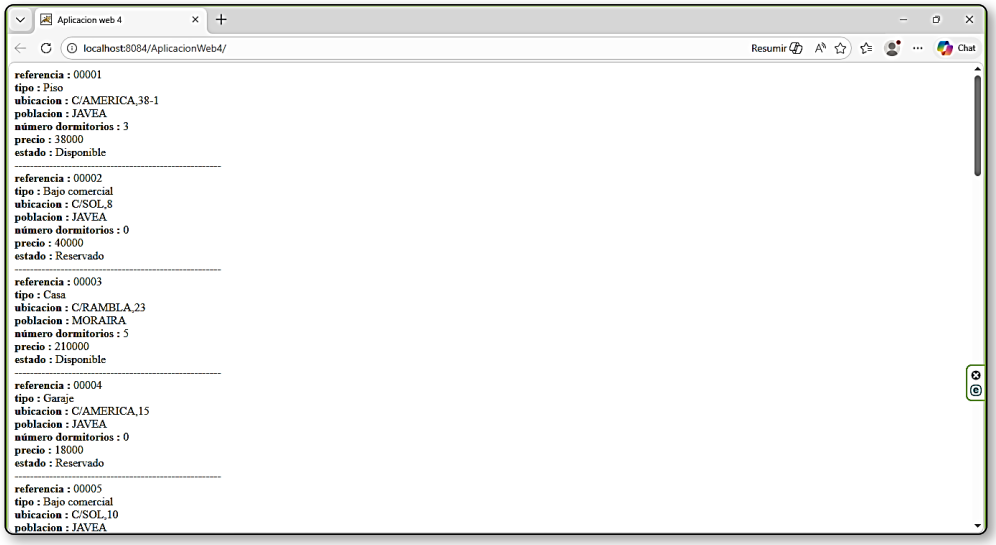
Esta aplicación accede a Bases de Datos. Nos remitimos al apartado del Anexo en que hacemos referencia a detalles a tener en cuenta en las aplicaciones de este libro que están en esta circunstancia.

Podemos observar que aparecen en esta aplicación ejemplo una serie de líneas con el siguiente formato:

```
<%@page import="java.sql.Connection"%>
```

En primer lugar, manifestar que este libro no pretende erigirse en un manual de referencia de los diferentes componentes de que podamos disponer en una aplicación Java EE. Pero sí que nos vemos forzados a comentar que en una página JSP pueden aparecer determinadas directivas, las cuales establecen directrices sobre al modo de procesamiento de la página. Entre ellas nos podemos encontrar con la directiva *page*. En este caso nos aparece con uno de sus posibles atributos, *import*. Fácilmente podrá deducir el lector que la misión es análoga a la de *import* en las clases Java.

Mostramos a continuación la captura de pantalla correspondiente a la ejecución de esta aplicación ejemplo:



Por tratarse de una presentación en modo prueba, ésta tiene un carácter "monolítico", de modo que para ver la totalidad es necesario interactuar con la barra de desplazamiento vertical del propio navegador. Admisible en este caso por tratarse, como hemos dicho, de una presentación de prueba, y porque el volumen de los resultados es mínimo. En la captura de pantalla solamente se observa la parte inicial de dichos resultados. Se trata ésta de una circunstancia totalmente impropia e inadecuada en una aplicación real destinada a ser desplegada en producción, en que además nos podríamos encontrar con la necesidad de visualizar la información de centenares o millares de filas. En esos casos, inexcusablemente deberemos aplicar técnicas de paginación en la presentación de los resultados. Se trata éste de un aspecto cuyo tratamiento no vamos a eludir, sino que será objetivo monográfico de un capítulo posterior de este libro.

2

SERVLETS

2.1 INTRODUCCIÓN AL CONCEPTO DE SERVLET

El Servlet es una clase Java, específica para la arquitectura Java EE, y es lo que confiere a esta arquitectura superioridad en eficiencia en relación al resto de tecnologías competidoras. Se ejecuta en servidor.

Iniciemos el tratamiento de los Servlets con la aplicación ejemplo **AplicacionWeb5**. En esta primera aproximación al Servlet, a efectos de mostrar el resultado de su ejecución, mantenemos la misma estructura de despliegue que en las aplicaciones ejemplo del capítulo anterior, pero ampliada:

- **web.xml.-** Fichero que centraliza la información de despliegue de la aplicación y otra información neurálgica.
- **inicio.jsp.-** Página "de bienvenida" a la aplicación.
- ***package* presentacion.-** Novedosamente incorporado a la aplicación con la misión de aglutinar las clases Java que formarán parte de la capa de presentación. Para una mejor comprensión de este concepto, el lector puede remitirse al capítulo 3 del libro "Java: JDBC y Swing" de este autor, en que fue ampliamente tratado el modelo de desarrollo de software arquitectura a tres capas. Este modelo de desarrollo será aplicado en su verdadera esencia y extensión en las aplicaciones ejemplo más avanzadas de este libro. No es el caso de estas primeras aplicaciones, pero dado que el Servlet formará parte de la capa de presentación empezaremos a integrarlo, desde un inicio, en el lugar que le corresponde.
- **Servlets.-** Objeto de tratamiento en este capítulo, ubicados, como ya se ha mencionado en el *package* presentación.

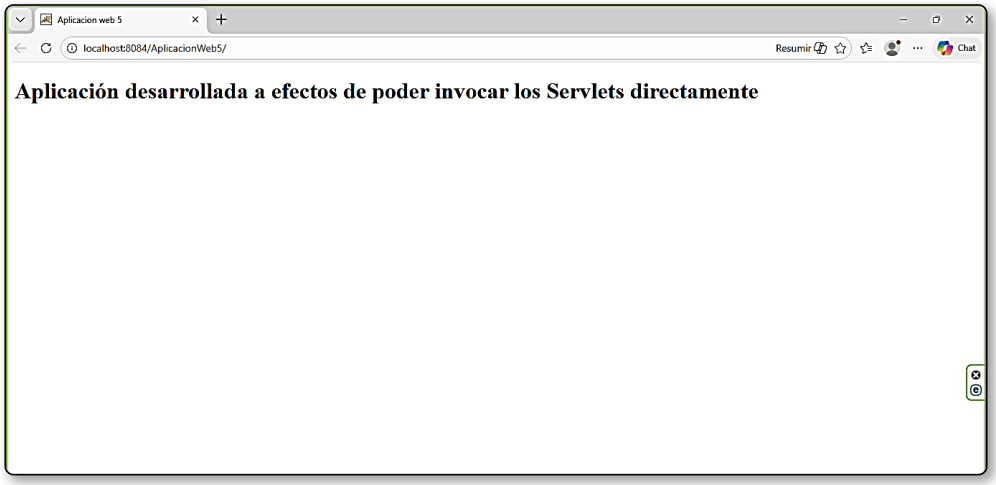
Para invocar al Servlet explícitamente (situación muy poco frecuente), o implícitamente desde otros componentes de la aplicación (es lo habitual), es necesario que exista información de mapeo del mismo en el centro neurálgico de la aplicación, el **web.xml**. Vamos a mostrar el correspondiente fichero de esta aplicación, en que apreciaremos como queda registrada la información de mapeo de los Servlets que formarán parte de la aplicación:

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/
javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>Servlet1</servlet-name>
        <servlet-class>presentacion.Servlet1</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>Servlet2</servlet-name>
        <servlet-class>presentacion.Servlet2</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>Servlet3</servlet-name>
        <servlet-class>presentacion.Servlet3</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Servlet1</servlet-name>
        <url-pattern>/Servlet1</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Servlet2</servlet-name>
        <url-pattern>/Servlet2</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Servlet3</servlet-name>
        <url-pattern>/Servlet3</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>inicio.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Una alternativa para definir la citada información de mapeo del Servlet es mediante la anotación *@WebServlet(...)*.

Ejecutemos la aplicación y observaremos como se manifiestan los Servlets de la misma:



Observamos que el punto de entrada a la misma sigue siendo, como lo fue en los casos del capítulo anterior la página **inicio.jsp**. Pero si a la URL de invocación a la aplicación se le añade un recurso de la misma, concretamente el Servlet:

http://localhost:8084/AplicacionWeb5/Servlet1

se produce, como podemos observar una ejecución directa de Servlet1. Para la ejecución del resto de los Servlets de la aplicación procederemos exactamente igual que con este, añadiendo a la URL de invocación, el nombre del Servlet correspondiente. Maticemos que invocar a ejecución a los Servlets de esta manera es algo inusual.

Detallamos a continuación el código de esta clase Servlet:

Servlet1

```
package presentacion;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Servlet1 extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Servlet1</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Visualizado en la ejecución de Servlet1</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    public String getServletInfo() {
        return "Short description";
    }

}
```

Toda petición al Servlet se inicia por el método *service()*, encadenando su ejecución con uno de los dos siguientes métodos:

- *doGet(HttpServletRequest request, HttpServletResponse response)*
- *doPost(HttpServletRequest request, HttpServletResponse response)*

Uno u otro en función de que la petición haya sido tipo GET o el tipo POST. Ambos canalizan la ejecución al método:

```
processRequest(HttpServletRequest request, HttpServletResponse response)
```

en el que habitualmente implementaremos todo el código que hayamos destinado al Servlet.

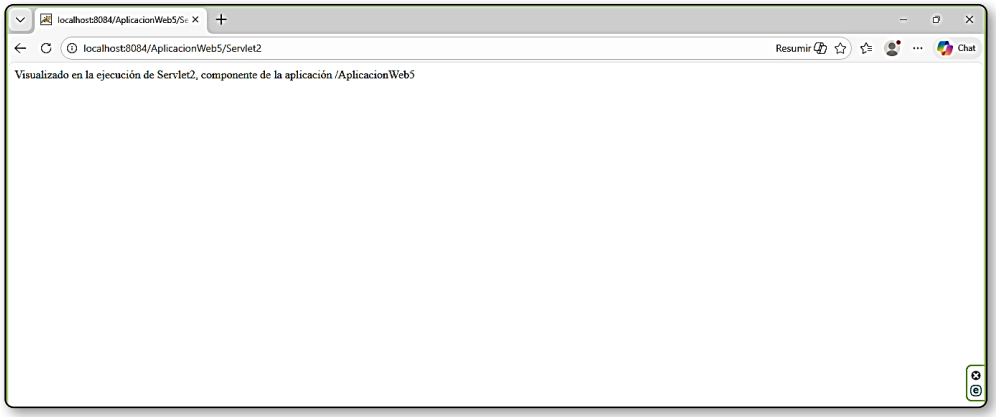
Habremos observado que el resultado de la ejecución de Servlet1 consiste en enviar al cliente la página HTML conformada en la ejecución del método *processRequest*. El parámetro recibido

```
HttpServletResponse response
```

nos permite acceder a métodos enfocados a la manipulación de la respuesta del Servlet. En este ejemplo:

- *response.setContentType("text/html;charset=UTF-8").* Establecemos el tipo de contenido de la respuesta.
- *PrintWriter out = response.getWriter().* Obtenemos el flujo de salida para la respuesta. Podremos construir dicha respuesta invocando al correspondiente método *println()* del citado flujo de salida. En el caso del ejemplo, construimos, como ya hemos mencionado una página HTML.

Nos encontramos también en esta aplicación ejemplo **Servlet2**, manifestándose su ejecución en



Procedamos a mostrar el código de este **Servlet2**. A partir de aquí y en lo sucesivo, en pro de abreviar contenido en el libro, en el caso de los Servlets solamente mostraremos el contenido del método *processRequest*, dado que generalmente el contenido del resto de métodos es invariable.

Servlet2

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("Visualizado en la ejecución de Servlet2, componente de la
aplicación " + request.getContextPath());
        }
```

A diferencia del caso anterior, no enviamos la estructura completa de una página HTML, sino solamente una cadena conformada por un literal al que le concatena lo devuelto por el método

```
request.getContextPath()
```

En el Servlet anterior hemos tenido oportunidad de mostrar la utilización de

```
HttpServletResponse response
```

Análogamente, en este otro Servlet actuaremos con

```
HttpServletRequest request
```

objeto que nos permite acceder a la información enviada al Servlet., que generalmente se tratará información manipulada por el usuario. En esta ocasión, dada la simplicidad del ejemplo utilizaremos dicho objeto para obtener información relacionada con el contexto en que está ubicado el Servlet.

Es necesario comentar que el uso de que se le está dando a los Servlets de esta aplicación ejemplo, de enviar salida a visualizar la máquina cliente es algo inusual. Lo estamos haciendo aquí como primera aproximación de contacto al Servlet y mostrar sus posibilidades. No obstante, como veremos en aplicaciones más avanzadas de este libro, el uso que se le dará al Servlet será muy distinto: actuar como nexo de unión y comunicación entre los componentes visuales (en este libro páginas JSP) de la aplicación y las clases Java de las capas “más profundas” que se ejecutan en servidor.

Otro Servlet de la aplicación:

Servlet3

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        System.out.println("Visualizado en la ejecución de Servlet3 en la consola del
servidor");
}
```

En relación a la ejecución de este otro Servlet, no mostramos captura de pantalla de la visualización en el navegador de la máquina cliente, pues no aparece nada. En su lugar mostraremos captura de pantalla de la consola del servidor:

Observamos que se visualiza el literal String parámetro del método:

```
System.out.println("Visualizado en la ejecución de Servlet3 en la consola del
servidor");
```

Este recurso nos puede ser de utilidad a efectos de depuración de código o notificación de incidencias en la consola del servidor.

2.2 CICLO DE VIDA DEL SERVLET

El ciclo de vida de un Servlet se inicia con la carga en memoria del mismo. Esta actuación puede producirse con la primera petición de ejecución, o cuando arranca el contenedor web, dependiendo de la información de despliegue del Servlet registrada en la aplicación de que forma parte. Una vez cargado en memoria un Servlet, la correspondiente instancia está ubicada en memoria, ocupando constantemente el correspondiente espacio. Esta circunstancia es extensiva a todos los Servlets de todas las aplicaciones desplegadas en el contenedor web. Razón por la que es conveniente minimizar el número de Servlets de una aplicación, lo cual derivará en la minimización del número total de Servlets cargados en memoria. Uno de los beneficios que conllevará la aplicación del patrón de diseño Front Controller, a tratar en capítulos posteriores, será reducir a uno el número de Servlets de toda aplicación, con independencia de la envergadura de la misma.

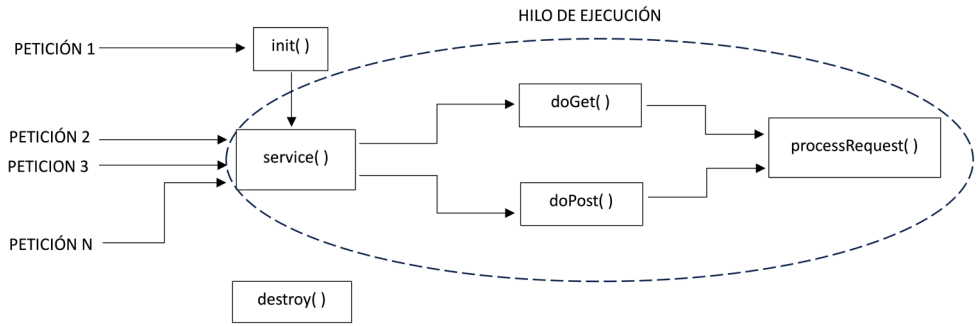
Dicha instancia se encuentra, como ya hemos dicho, constantemente cargada en memoria en condiciones de atender todas las demandas de ejecución que se produzcan desde las diferentes sesiones que puedan abrirse desde máquinas cliente. De modo que para cada una de las invocaciones que se produzcan, se crea un hilo de ejecución, concurrente con los demás que pudieran producirse, para los métodos

```
service() → doGet() ó doPost() → processRequest()
```

tal y como se ha expuesto anteriormente. Con la primera petición, adicionalmente a este encadenamiento de métodos, también se ejecuta previamente el método *init()*.

Cuando el Servlet entra en un proceso de finalización, bien por actuación explícita sobre el mismo, bien por finalización del contenedor web en que está desplegado, se ejecuta el método *destroy()* en el que implementaremos la liberación de recursos.

El siguiente gráfico ilustra el ciclo de vida del Servlet:



En la siguiente aplicación ejemplo mostraremos como en la clase Servlet podemos:

- Declarar variables globales, cuyo acceso será común y compartido desde todos los hilos de ejecución que pudieran producirse durante toda la vida del Servlet.
- Sobrescribir el método

```
public void init(ServletConfig servletConfig)
```

  Se trata de un método predefinido subyacente que se ejecuta una sola vez, en la primera invocación al Servlet. Con esta sobreescritura podremos implementar acciones de inicialización, sin olvidar tomar siempre la precaución de invocar al método predefinido que se sobrescribe, pues con la ejecución del mismo se realizan acciones ineludibles.

En la siguiente aplicación ejemplo, AplicacionWeb6, hemos implementado Servlet4 con el fin de evidenciar todo ello.

Servlet4

```
package presentacion;

import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Servlet4 extends HttpServlet {

         byte contadorSesiones;
         String cadenaFechas;

    public void init(ServletConfig servletConfig) throws ServletException {
         super.init(servletConfig);
         contadorSesiones = 0;
         cadenaFechas = "";
       }

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
           contadorSesiones ++;
           cadenaFechas = cadenaFechas + "Sesion: "+ String.
valueOf(contadorSesiones)+" Tiempo: "+new SimpleDateFormat("dd-MM-yyyy HH:mm:ss").
format(new Date())+" ----- ";
           out.println(cadenaFechas);
        }
    }
         …
}
```

inicio.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
        <title>Aplicacion web 6</title>
    </head>
    <body>
        <a href="Servlet4">Servlet4</a>
    </body>
</html>
```

Analicemos la aplicación. Novedosamente en relación a los ejemplos anteriores, implementamos un enlace en la página "de bienvenida" *inicio.jsp*

```
<a href="Servlet4">Servlet4</a>
```

que nos permitirá invocar al Servlet implícitamente sin necesidad de añadirlo a la URL en el navegador. Ya en el Servlet observamos las innovaciones anunciadas:

- La declaración de los datos globales del Servlet

```
byte contadorSesiones;
String cadenaFechas;
```

  que serán accesibles y compartidos desde todos los hilos de ejecución del método *processRequest*. Hemos declarado la variable **contadorSesiones**, que actúa como contador, tipo byte. Es este el más reducido en tamaño de todos los tipos numéricos enteros. Ello se hecho testimonialmente, con la intención de evidenciar la conveniencia de minimizar el uso de memoria en datos que van a estar permanentemente ocupando el correspondiente espacio, como es el caso de esta variable, dado el ámbito global al Servlet en que ha sido declarada.

- La implementación del método

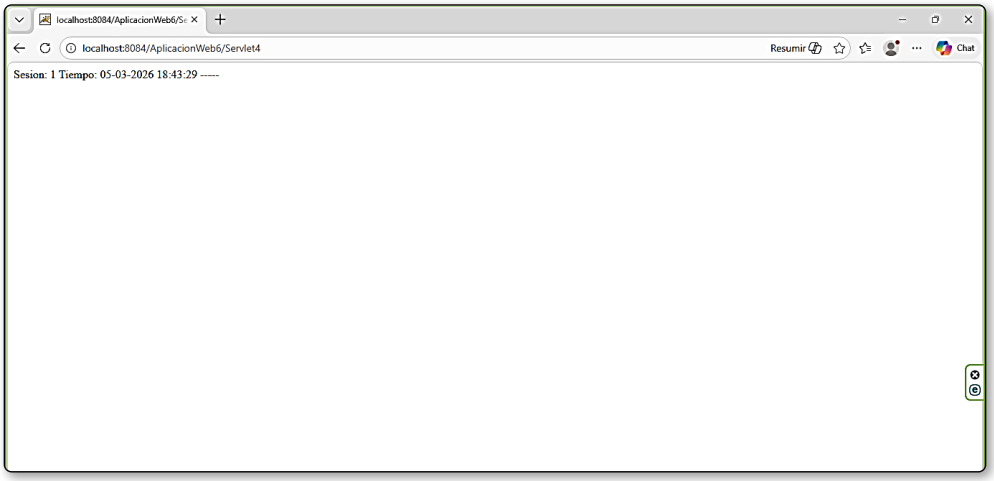
```
    public void init(ServletConfig servletConfig) throws ServletException {
        super.init(servletConfig);
        contadorSesiones = 0;
        cadenaFechas = "";
    }
```

  en que observamos
  - La invocación explícita al método que se sobrescribe.
  - La inicialización de los datos globales. Esta actuación se podría haber simultaneado con la declaración, pero la implementamos aquí testimonialmente y en representación de otro tipo de inicializaciones que ineludiblemente debería realizarse en este método.

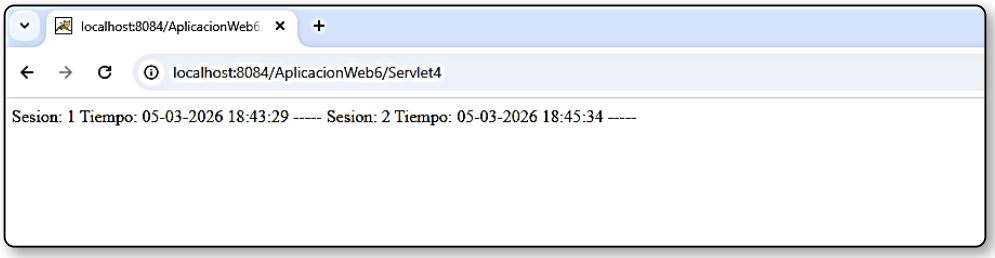
- En el método *processRequest*, con cada hilo de ejecución del mismo:
  - incrementamos el valor de la variable que actúa como contador de sesiones.
  - Concatenamos al String global al Servlet el número de sesión y el momento del tiempo del correspondiente hilo de ejecución.

En las siguientes capturas de pantalla podremos comprobar, con las sucesivas invocaciones a ejecutar la aplicación que conllevan las sucesivas concatenaciones al String global, como va incrementándose el valor de la variable global que actúa como contador de sesiones y el momento del tiempo en que ello se produce. El lector puede afianzar esta comprobación invocando la aplicación directamente con la URL aplicada a otros navegadores. Incluso mejor, si se dispone de otras máquinas conectadas en red, invocando directamente la aplicación desde el navegador de dichas máquinas, pero sustituyendo localhost por la dirección IP de la máquina donde está ubicado el servidor que aloja la aplicación.

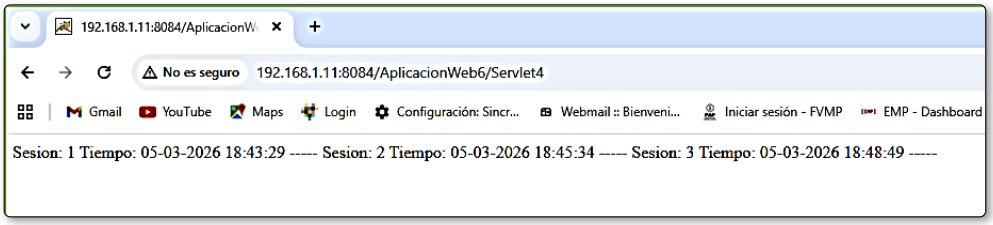
Iniciamos en primer navegador:



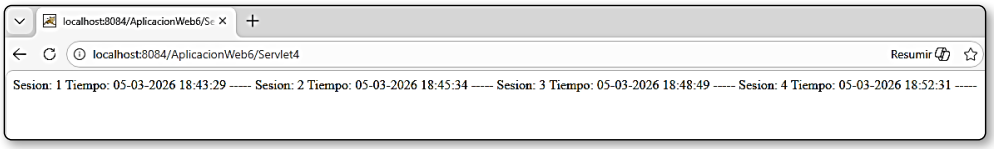
A continuación, ejecutamos la aplicación desde otro navegador en el mismo ordenador. Prescindimos de la **jsp** de inicio con el enlace al Servlet. Las sucesivas capturas de pantalla serán expuestas mediante visualización parcial, circunstancia que nos permitirá disponer de una perspectiva ampliada del contenido.



Vamos ahora a un segundo ordenador. Se puede verificar que es así por la URL de invocación a la aplicación, en que se ha sustituido localhost por la IP del que actúa como servidor.



Y ahora proseguimos, invocando la aplicación desde el primer navegador:



Podríamos considerar el mecanismo de acceso compartido a datos globales del Servlet, en este caso la variable que actúa como contador de sesiones y el String sometido a sucesivas concatenaciones, como un preludio de la aplicación ejemplo Chat, que será presentada en el siguiente capítulo. En dicha aplicación, los datos accedidos desde todas las sesiones serán una variable que actúa como contador de mensajes, totalmente análoga a la de esta aplicación, y un *ArrayList*, que tendrá la misión de aglutinar objetos encapsuladores de la clase **Mensaje**.

2.3 TEMPORIZACIÓN DE TAREAS

Expondremos en este apartado cómo establecer tareas temporizadas mediante una aplicación Java EE, en este caso, desde un Servlet. Disponemos para ello de la siguiente aplicación ejemplo:

AplicacionWeb12 – COMPONENTES DE LA APLICACIÓN

- **web pages**
  - inicio.jsp
- *package* **presentacion**
  - Servlet10
  - TareaPeriodica

Servlet10

```
package presentacion;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Servlet10 extends HttpServlet {

        private void ejecutarTareaPeriodica(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
           IOException {
            response.addHeader("Refresh", "3");
            new TareaPeriodica().ejecutarPeriodicamente();
   }


    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // processRequest(request, response);
        ejecutarTareaPeriodica(request, response);
    }


    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // processRequest(request, response);
        ejecutarTareaPeriodica(request, response);
    }


    @Override
    public String getServletInfo() {
        return "Short description";
    }

}
```

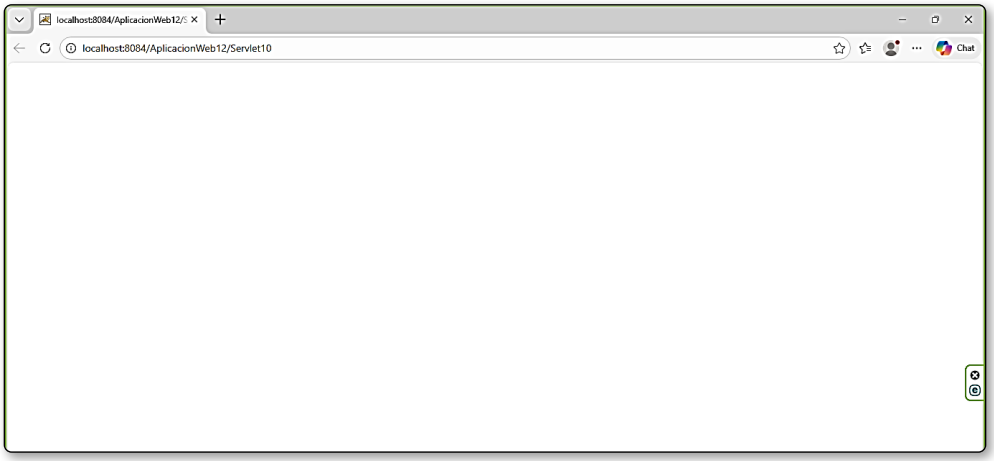
TareaPeriodica

```
package presentacion;

import java.text.SimpleDateFormat;
import java.util.Date;

public class TareaPeriodica {
    void ejecutarPeriodicamente() {
        System.out.println("Tiempo: " + new SimpleDateFormat("dd-MM-yyyy HH:mm:ss").
format(new Date()));
        System.out.println("La visualización de esta cadena representa lo que sería
la ejecución de una tarea periodica");
    }
}
```

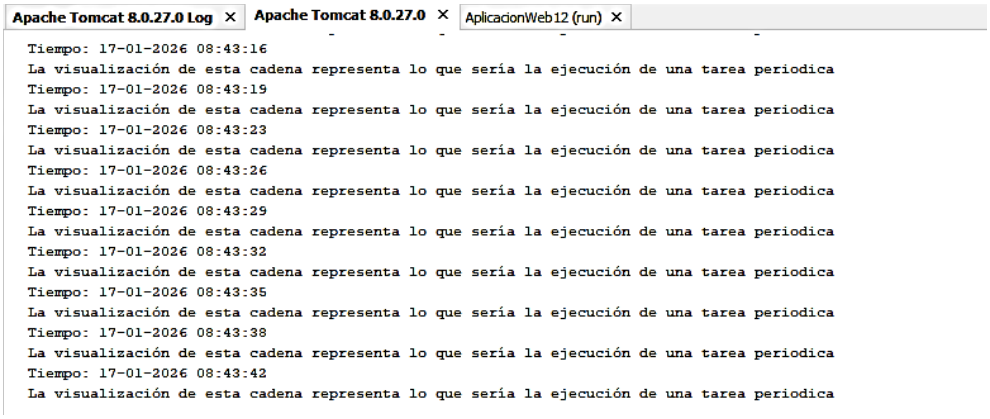
inicio.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Aplicacion Web 12</title>
    </head>
    <body>
        <% response.sendRedirect("Servlet10"); %>
    </body>
</html>
```

Procedamos a ejecutar la aplicación para posteriormente analizar el código que ha generado dicha ejecución. Cuando la aplicación se inicia, aparece un navegador con la siguiente apariencia:



Es decir, no se visualiza nada en el navegador. El propósito de esta aplicación es iniciar una tarea de temporización periódica, de modo que la constancia de que realmente se ejecutan acciones periódicas temporizadas queda reflejada en la consola del servidor, tal y como muestra la siguiente captura de pantalla:



Analicemos el código de la aplicación. El primer componente que toma el control de ejecución es **inicio.jsp**, cuya única acción consiste en redirigir automáticamente el control de ejecución a **Servlet10** mediante

```
<% response.sendRedirect("Servlet10"); %>
```

que formará parte de los contenidos a tratar en el siguiente capítulo. Observaremos que insólitamente en relación al resto de aplicaciones ejemplo de este libro, los dos métodos de entrada al Servlet, *doGet()* y *doPost()*, en lugar de redirigir el control de

ejecución al método *processRequest()*, lo hacen a un nuevo método implementado en el Servlet, concretamente *ejecutarTareaPeriodica()* que es el que concentra todo lo relativo a la tarea temporizada. El método *ejecutarTareaPeriodica()*, solamente implementa las siguientes dos líneas:

```
            response.addHeader("Refresh", "3");
            new TareaPeriodica().ejecutarPeriodicamente();
```

La primera es la que se encarga de activar la temporización, estableciéndose la duración del periodo de refresco mediante el segundo parámetro que viene expresado en segundos, en este caso concreto 3 segundos. Observamos en la captura de pantalla de la consola del servidor que efectivamente la tarea se ejecuta cada 3 segundos. La segunda línea ejecuta el método *ejecutarPeriodicamente()* de la *class TareaPeriodica* que es donde implementamos las tareas que en concreto deben ejecutarse periódicamente:

```
 void ejecutarPeriodicamente() {
        System.out.println("Tiempo: " + new SimpleDateFormat("dd-MM-yyyy HH:mm:ss").
format(new Date()));
        System.out.println("La visualización de esta cadena representa lo que sería
la ejecución de una tarea periodica");
    }
```

La primera de las dos líneas permite visualizar el momento del tiempo en que se produce cada ejecución periódica. Dados los fines didácticos que nos guían, solamente provocamos que se ejecute algo testimonial representativo que nos permita comprobar que realmente se ejecutaría cualquier tarea de mayor envergadura si este mecanismo se aplicase en un entorno en producción. En ese caso, este método sería el encadenante de toda una jerarquía de arquitectura a 3 capas en que quedaría organizada la tarea periódica a ejecutar.

3

INTERACCIÓN ENTRE SERVLET Y OTROS COMPONENTES

3.1 INTERCAMBIO DE INFORMACIÓN MEDIANTE ATRIBUTOS

Mostraremos este mecanismo en **AplicacionWeb7**. Esta modalidad de intercambio de información entre componentes consiste en que el componente "generador" de la información, en el caso de este ejemplo **Servlet5**, "deposita" la referencia del objeto a transmitir en un atributo mediate el método *setAttribute()* del objeto *request* ya presentado en el capítulo anterior:

```
request.setAttribute("personaEncapsuladaEnRequest", persona);
```

Generalizando:

```
request.setAttribute("NOMBRE_ATRIBUTO", REFERENCIA_A_OBJETO);
```

Referencia que extraerá para su utilización el componente receptor, en este caso **visualizaPersona.jsp**, mediante su opuesto, el método *getAttribute()* del citado objeto *request*:

```
Persona persona = (Persona)request.getAttribute("personaEncapsuladaEnRequest");
```

Que generalizando:

```
CLASE REFERENCIA_A_OBJETO = (CLASE)request.getAttribute("NOMBRE_ATRIBUTO");
```

Novedosamente aparece en esta aplicación: