Tema 5
Introducción a la plataforma JakartaEE. Servlets

¿Qué es un servidor de aplicaciones?

Un **servidor web** (Apache) almacena y entrega contenido o servicios a usuarios en respuesta a peticiones, normalmente http. Por sí mismo, no es capaz de procesar contenido dinámico.

Un **servidor de aplicaciones** es un software que, además de funcionar como servidor web,

- Proporciona contenido dinámico
- Proporciona una serie de servicios a aplicaciones que se ejecutan en su interior. Por ejemplo: pools de conexiones con BBDDs, gestión de transacciones, clustering, reparto de la carga de la aplicación web entre varios servidores, etc.

El término "servidor de aplicaciones" usualmente hace referencia a un servidor de aplicaciones Java EE.
Algunos servidores de aplicación Java EE son: WebLogic, Jboss, GlassFish
Mucha gente confunde Tomcat como un servidor de aplicaciones; sin embargo, es solamente un contenedor de servlets.

Para una app web Java

- El servidor web recibe la petición http.
- Analiza dicha petición y almacena su información en forma de objetos Java.
- Busca si existe algún programa registrado para gestionar la petición (un servlet) e invoca dicho programa, pasándole como parámetros objetos Java con toda la información de la petición.
- Empleando el API (clases de Java) adecuado, nuestro programa podrá examinar la petición, procesarla, y generar la respuesta

¿Qué son los Servlets?

Los Servlets son objetos Java (de clases que implementan la interface Servlet) que se ejecutan en un servidor de aplicaciones o en un contenedor web y permanecen a la espera de peticiones web de clientes, normalmente peticiones http, y les dan una respuesta.
Actúan como una capa intermediaría entre la petición web y las aplicaciones, BBDDS o recursos del servidor Web.
Los Servlets no se encuentran limitados a un protocolo de comunicaciones específico, pero en la práctica se utilizan únicamente con el protocolo HTTP.

Lugar de los Servlets en la jerarquía de clases Java

Un servlet, al ser una clase de Java, incorpora todas las ventajas del lenguaje Java en cuanto a portabilidad y seguridad. Los 2 paquetes principales de la API Java con clases para programación de servlets son:

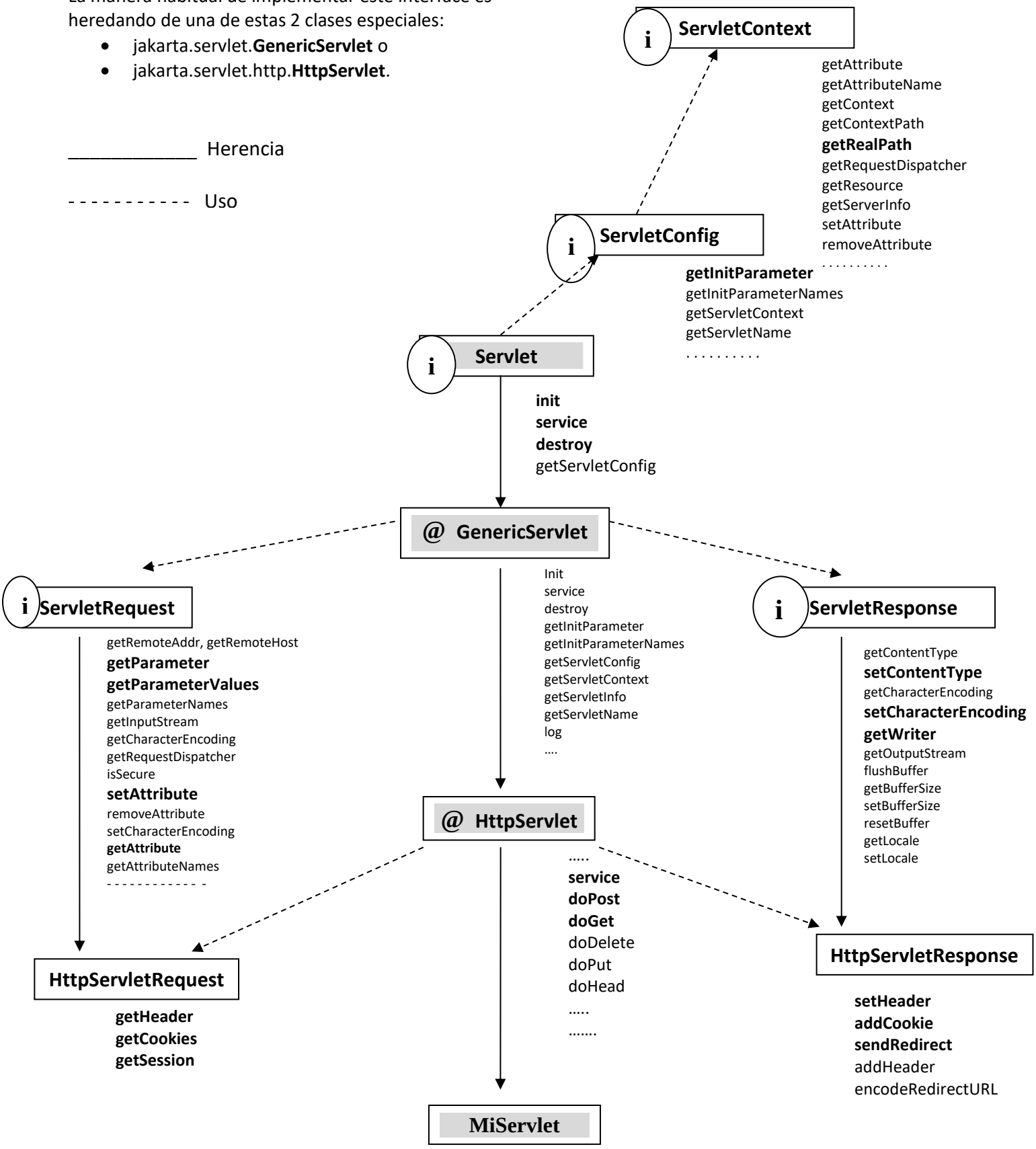
- **jakarta.servlet** → Contiene clases e interfaces para los servlets genéricos, independientes del protocolo
- **jakarta.servlet.http** →Clases que extienden las del paquete anterior, añadiendo funcionalidad específica para el protocolo http

Los servlets son clases que deben implementar la interface jakarta.servlet.**Servlet**
La manera habitual de implementar este interface es heredando de una de estas 2 clases especiales:

- jakarta.servlet.**GenericServlet** o
- jakarta.servlet.http.**HttpServlet**.

_____ Herencia

- - - - - - - - - - - Uso



Tareas de un servlet HTTP básico

Este es la secuencia de pasos más habitual que realiza un servlet http, cuando le llega una petición

- Crear un hilo de control para atender dicha petición
- Leer los datos enviados por la petición: parámetros enviados desde formularios, información de la sesión abierta, cookies, datos del navegador del cliente, etc
- Aceptar la petición o bien delegarla a otro servlet
- Buscar información asociada a la petición y procesarla. Este proceso suele requerir operaciones sobre bases de datos u otros recursos
- Generar la respuesta, formateando los resultados en un flujo de salida, normalmente HTML, pero también formato binario: imágenes, ficheros zip, etc
- Enviar la respuesta al cliente

Principales clases/métodos de la API de Jakarta servlets:

Interface **Servlet**

Es la **interface** principal. Todos los servlets que creemos deben implementarla, directamente o más comúnmente, extendiendo de una clase que lo implemente como HttpServlet. Métodos:

init (*ServletConfig* config)	Invocado por el servidor al instanciar el servlet. Puede ser sobreescrito para realizar tareas al comienzo de la vida del servlet, por ejemplo, poner un contador de visitas a 0
getServletConfig ( )	Retorna la configuración dada para la inicialización del servlet
service (*ServletRequest* request, *ServletResponse* response)	Invocado por el servidor cada vez que se recibe una petición En su implementación para HTTP verifica el tipo de solicitud GET, POST, etc. y la redirige a los métodos doPost, doGet. No es recomendable sobrescribirlo.
destroy( )	Invocado por el servidor antes de que el servlet sea destruido.

Clase abstracta **GenericServlet**: **public abstract class GenericServlet** implements **Servlet, ServletConfig**

Usada para programar servlets genéricos, independientes del protocolo
Implementa el método service para establecer el comportamiento mínimo de todos los servlets.

Clase abstracta **HttpServlet**: **public abstract class HttpServlet extends GenericServlet**:

Utilizada para programar servlets que respondan a peticiones http. Todo Servlet http debe heredar de ella y sobrescribir alguno de sus métodos. Sus métodos más comunes son:

doGet (*HttpServletRequest* request, *HttpServletResponse* response)	Invocado cuando se solicita el servlet por GET
doPost (*HttpServletRequest* request, *HttpServletResponse* response)	Invocado cuando se solicita el servlet por POST

Clase **HttpServletRequest**: Implementa la interface ServletRequest

Al recibir una petición, el contenedor de Servlets creará un objeto HttpServletRequest que la represente y se lo pasará como parámetro a doGet(), doPost() , etc

Este objeto representa la petición del servlet que hace el cliente. Tiene estos métodos para obtener la información que envía el usuario en la petición:

String **getHeader** (String name)	Devuelve el contenido de la cabecera http cuyo nombre se le pa
Cookie[] **getCookies**()	Devuelve 1 array con las cookies del cliente actual
HttpSession **getSession**(boolean create) y HttpSession **getSession**():	Devuelven la sesión en la que se encuentra el cliente y crean una sesión nueva si no existe. ***getSesssion(true)*** y ***getSession()*** son equivalentes.
String **getParameter**(String name)	Devuelve el valor del parámetro cuyo nombre se pasa o null
String[] **getParameterValues**(String name)	Equivalente a getParameter para campos multiselect
void **setAttribute**(String name, Object object) Object **getAttribute**(String name) Object **removeAttribute**(String name)	Asigna, recupera y borrar el valor de un atributo de la sesión, respectivamente
String **getContextPath**()	Devuelve la URL que indica el contexto de la petición. En un mismo servidor de aplicaciones puede haber desplegadas varias aplicaciones Java. El servidor se encarga de aislarlas, teniendo cada una su contexto. Ese contexto es el 1º fragmento de la URL después del nombre del servidor. Es básicamente el nombre de la aplicación: /WebFruteria
String **getMethod**()	Devuelve el método http empleado en la petición; por ejemplo GET, POST, o PUT.

Clase **HttpServletResponse:** Implementa la interface ServletResponse

Representa la respuesta que se dará al usuario tras servir su petición, normalmente en formato html

void **setContentType**(String type)	Establece el tipo de respuesta a devolver al cliente: "text/html", "text/plain","application/pdf","image/gif", etc
PrintWriter **getWriter**() *Método de ServletResponse*	Devuelve un objeto *PrintWriter* asociado con la respuesta que será enviada al cliente, si esta es textual. Normalmente, escribiremos texto HTML en este flujo.
ServletOutputStream **getOutputStream**() *Método de ServletResponse*	Devuelve un objeto *ServletOutputStream*. Se usa para construir respuestas binarias (no textuales), por ejemplo, una imagen.
void **addCookie**(Cookie cookie)	Añade una cookie a la respuesta
void **addHeader**(String name, String value)	Añade a la respuesta una nueva cabecera
void **sendRedirect**(String location)	Envía un mensaje al cliente para que éste redireccione la respuesta a la dirección señalada. Es importante notar que se genera **una nueva petición** desde el cliente
String **encodeRedirectURL**(String url)	Prepara una URL para ser usada por el método sendRedirect

Interface **ServletConfig**: Para que el contenedor del servlet, pase al servlet información de configuración: parámetros de inicialización, contexto del servlet, etc

String **getInitParameter**(String name)	Devuelve el valor de 1 de parámetro inicialización del servlet
String[] **getInitParameterNames**()	Devuelve en un array los nombres de todos los parámetros de inicialización del servlet
ServletContext **getServletContext()**	Devuelve 1 objeto ServletContext con la información referente al contexto del servlet, es decir, a la aplicación.

Interface **ServletContext**: Se refiere al contexto del servlet o dicho de otra manera, a la "Aplicación web". En el contexto reside aquella información que es común a todos los Servlets desplegados en la aplicación actual.

Existe un único contexto por cada aplicación web y por cada instancia de la máquina virtual Java. Por eso, el contexto es un mecanismo para comunicar información entre todos los Servlet que corran en una misma aplicación web, siempre y cuando esa aplicación web corra en un mismo servidor (si tenemos varios servidores, y por tanto varias máquinas virtuales, el contexto de la aplicación no es un sitio adecuado para guardar información a la que queramos acceder globalmente; en estos escenarios suele recurrirse a una base de datos).

Métodos

void **setAttribute**(String name, Object object) Object **getAttribute**(String name) Object **removeAttribute**(String name)	Asigna, recupera y borrar el valor de un atributo del contexto, respectivamente.
String **getContextPath**()	Devuelve la ruta del contexto de la aplicación web
RequestDispatcher **getRequestDispatcher**(String)	Devuelve un objeto RequestDispatcher que suele emplearse para redirigir la petición a otro recurso. Ejemplo: request. getRequestDispatcher("recurso"). forward(request,response);
URL **getResource**(String path)	Devuelve una URL que se corresponde con la ruta del recurso que se le pasa como parámetro. La ruta del recurso debe comenzar como un "/" y se interpreta como una ruta relativa a la raíz del contexto de la aplicación, o relativa al directorio /META-INF/resources de un archivo jar que esté contenido en /WEB-INF/lib
InputStream **getResourceAsStream**(String path)	Similar al anterior pero devuelve el recurso como un flujo de entrada
String **getRealPath**(String path):	Devuelve la URL absoluta para nuestra URL relativa

Funcionamiento de un servlet (NUESTRO PRIMER SERVLET)

Los Servlets objetos Java (clases Java compiladas -.class-) que se ejecutan dentro de un servidor de aplicaciones JakartaEE, en concreto, en un contenedor de Servlets, como Tomcat

Los servlets deben estar **registrados** para poder comenzar a atender peticiones de clientes.

Cuando al contenedor le llega una petición http, comprueba si hay algún Servlet registrado para responder a dicha petición; en caso afirmativo, este servlet atenderá la petición.

Crearemos nuestros servlets heredando de la clase abastracta **HttpServlet.**

Esta clase ofrece, entre otros, los métodos **doGet()** y **doPost()** que podrá sobrescribir nuestro servlet.

1. Si la petición que ha llegado al Servlet era de tipo GET se invocará doGet()
   (url, enlaces, redirección con sendRedirect….)
2. si era tipo POST invocará doPost()
   (envío de formularios POST, forward desde doPost de otro servlet)

Tanto **doPost()** como **doGet()** reciben 2 parámetros: 1 objeto **HttpServletRequest** y 1 objeto **HttpServletResponse.**
HttpServletRequest y **HttpServletResponse** que representan una petición y una respuesta del protocolo HTTP.

El objeto **HttpServletRequest** ofrece métodos para obtener información relativa a la petición HTTP realizada, como puede ser información incluida en formularios HTML, cabeceras de petición HTTP, etc

El objeto **HttpServletResponse** nos permite elaborar la respuesta HTTP a enviar al cliente. Esta información incluye cabeceras de respuesta del protocolo HTTP, códigos de estado, y lo más utilizado: nos permite obtener un objeto PrintWriter, donde escribir texto HTML.

Al reescribir **doGet** y **doPost**, hay que tener en cuenta que lanzan las excepciones ServletException e IOException

Ejemplo 1: Servlet HolaMundoServlet

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class HolaMundoServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                throws ServletException, IOException
        {
                response.setContentType("text/html;charset=UTF-8");
                PrintWriter out;
                out = response.getWriter();
                try {
                        out.println("<html>");
                        out.println("<head>");
                        out.println("<title>Hola mundo</title>");
                        out.println("</head>");
                        out.println("<body>");
                        out.println("<h1>Hola mundo!</h1>");
                        out.print("<img src='falta.png'/>");
                        out.println("</body>");
                        out.println("</html>");
                }
                finally {
                        out.close();
                }
        }
}
```

Servlet que genera una salida html estática.

Este servlet genera como respuesta una página web estática que incluye un texto y una imagen. La imagen debe de estar situada en la raíz de la carpeta web para poder acceder a ella empleando la ruta indicada en el código. Si desplegamos este Servlet en el servidor y hacemos una petición, veremos:

Observa la URL que se muestra en el navegador.

- La máquina es localhost.
- El protocolo (aunque Chrome lo omite) http.
- El puerto en el cual está el servidor es el 8080
- La aplicación está corriendo dentro del contexto "Servlet1" del servidor de aplicaciones.
- Y el recurso de la aplicación al cual estamos accediendo es HolaMundo.

Registrar Servlets

Debemos identificar nuestros servlets e indicar qué urls pueden atender. Existen 2 maneras:

Mediante **anotación @WebServlet** dentro de la propia clase

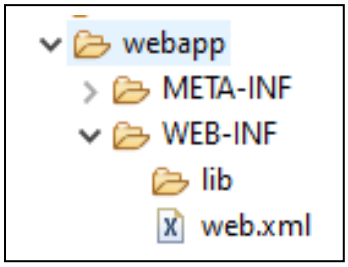
En la anotación indicamos un nombre interno para el servlet y los "patrones URL" a los que deseamos que el servlet responda (los patrones pueden usar el carácter comodín *)

Ejemplo:

```
@WebServlet(
        name="HolaMundoServlet",
        urlPatterns={"/HolaMundo/*","*.saludo"})
```

- Da al servlet el nombre interno HolaMundoServlet (coincide con el nombre de la clase, aunque no tendría por qué)
- El Servlet se registra para responder a URL's como las siguientes:
  http:// máquina:puerto/contexto/HolaMundo/synopsis
  http:// máquina:puerto/contexto/HolaMundo/complete?date=today
  http:// máquina:puerto/contexto/HolaMundo
  http:// máquina:puerto/contexto/Hola.saludo
  http:// máquina:puerto/contexto/directorio/fichero.saludo

Mediante el descriptor de despliegue: archivo **web.xml**



El descriptor de despliegue es un fichero de nombre **web.xml** que debe de situarse en la raíz del directorio WEB-INF. Algunos de sus usos son:

- registrar servlets e informar de correspondencias de URLs con servlets
- definir parámetros de inicialización de servlets y aplicaciones web
- configurar la sesión
- configurar la seguridad
- …….

Contenido de web.xml equiparable a la siguiente anotación en el fichero ***HolaMundoServlet.java*** :

```
@WebServlet(name="HolaMundoServlet", urlPatterns={"/HolaMundo"})
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ……>
        <servlet>
                <servlet-name>HolaMundoServlet</servlet-name>
                <servlet-class>paquete.HolaMundoServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>HolaMundoServlet</servlet-name>
                <url-pattern>/HolaMundo</url-pattern>
        </servlet-mapping>
</web-app>
```

<servlet> registra un Servlet en el contenedor. Incluye las subetiquetas:

- **<servlet-name>:** nombre interno del Servlet
- **<servlet-class>:** clase a la que pertenece (nombre completo con el paquete).

<servlet-mapping> permite especificar a qué URLs va a responder cada Servlet. Dentro de ella debemos indicar el nombre interno del Servlet y los patrones de URL al cual queremos que éste responda.
Son direcciones virtuales, no corresponden a ninguna ubicación.

El descriptor de despliegue tiene prioridad sobre las anotaciones.

Métodos **doGet** y **doPost**

La solicitud de un recurso web por parte de un cliente al servidor, se materializa en una petición http que puede ser de tipo GET o POST

Éste es el aspecto que tienen este tipo de peticiones a su llegada al servidor:

Petición GET	Petición POST
GET /servlet/MyServlet?nombre=Juan&pais=es HTTP/1.1 Connection: Keep-Alive User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows NT) Host: www.datsi.fi.upm.es Accept: image/gif, image/x-xbitmap, image/jpeg,	POST /servlet/MyServlet HTTP/1.1 User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows NT) Host: www.datsi.fi.upm.es Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */ Content-type: application/x-www-form-urlencoded Content-length: 30 nombre=Juan&pais=es

Una petición GET de un servlet (es el tipo de petición URL por defecto) llamará al método **doGet** que habremos codificado en dicho servlet y una petición HTTP llamará al método **doPost** del servlet.

Volviendo al ejemplo de **HolaMundoServlet**, el método doGet tiene 2 parámetros:

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

- El primero (request) es un objeto Java que envuelve la petición que ha llegado al servidor.
  Empleando métodos de este objeto, podremos acceder a información como a las cabeceras de la petición, parámetros de un formulario, sesión, cookies, etc
  En nuestro ejemplo con ***HolaMundoServlet***, este objeto no se usa para nada.

- El segundo objeto (response) se utiliza para generar la respuesta.
  Se puede obtener de él un flujo de tipo *PrintWriter* (salida de texto), donde se escribe html (aunque este cometido, los servlets lo delegarán a las vistas)

Es habitual implementar sólo un método (o bien doGet o bien doPost) y hacer que el otro lo referencie.

Ejemplo 2

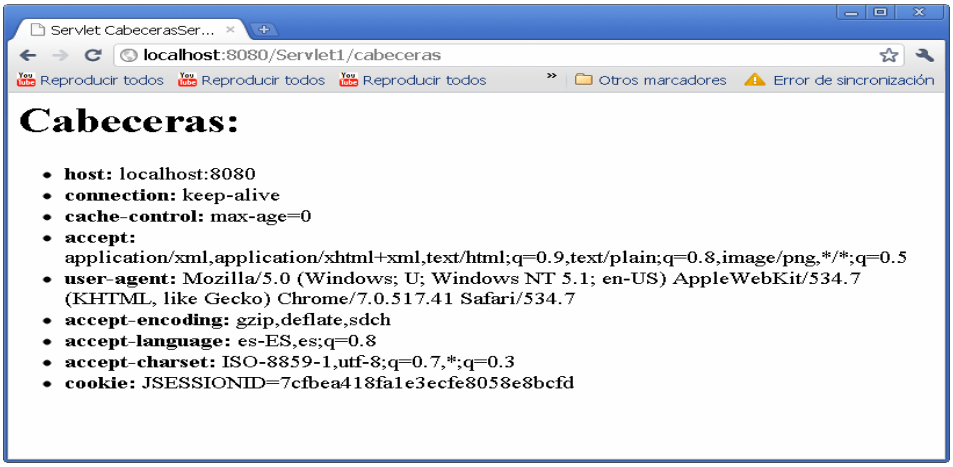
Servlet que genera una respuesta al usuario donde se muestran todas las cabeceras de la petición http recibida y el primero de sus valores (ya que es posible que 1 cabecera tenga múltiples valores).

El patrón de URL asociado con el ejemplo es /cabeceras.
Este Servlet codifica los métodos doGet y doPost de tal modo que ambos hacen lo mismo: invocan al método processRequest pasándole los parámetros con los que han sido invocados.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException
{
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
                out.println("<html>");
                out.println("<head>");
                out.println("<title>Servlet CabecerasServlet</title>");
                out.println("</head>");
                out.println("<body>");
                out.println("<h1>Cabeceras: </h1>");
                out.println("<ul>");
                Enumeration<String> nombresDeCabeceras =request.getHeaderNames();
                while (nombresDeCabeceras.hasMoreElements()) {
                        String cabecera = nombresDeCabeceras.nextElement();
                        out.println("<li><b>" + cabecera + ": </b>"+ request.getHeader(cabecera) + "</li>");
                }
                out.println("</ul>");
                out.println("</body>");
                out.println("</html>");
        } finally {
                out.close();
        }
}

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

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

}
```

Salida generada en Chrome

En este ejemplo, la salida que genera es dinámica: texto estático combinado con variables Java

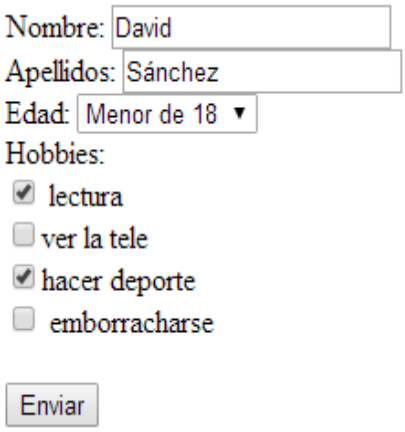


Ejemplo 3

Servlet que procesa formulario

Es habitual usar Servlets para procesar información que el cliente nos manda, normalmente desde formularios.

Supongamos que tenemos el siguiente formulario:




```
<form method="get" action="ServletLeeDatos" >
        Nombre: <input name="nombre"><br>
        Apellidos: <input name="apellidos"><br>
        Edad:
        <select name="edad">
                <option value="Menor">Menor de 18</option>
                <option value="Joven">De 18 a 30</option>
                <option value="Adulto">De 30 a 55</option>
                <option value="Mayor">Mayor de 55</option>
        </select>
        <br>
        Hobbies:<br>
        <input name="hobbies" value="lectura" type="checkbox"> lectura</br>
        <input name="hobbies" value="tele" type="checkbox">ver la tele</br>
        <input name="hobbies" value="deporte" type="checkbox">
                                                Hacer deporte</br>
        <input name="hobbies" value="emborracharse" type="checkbox">
                                                emborracharse</br></br>
        <input type="submit" value="Enviar"/>
</form>
```

** Atención, un conjunto de checkboxes relacionados, si va a ser procesado por un servlet Java, no necesita que le asignemos un nombre de array (a diferencia de en php). Basta con que el **name** sea el mismo para todos ellos*
(Lo mismo para cualquier entrada multiselect de formulario)

Observa que el formulario va a enviar sus datos a la URL formada al retirar el nombre del formulario de la URL de la página web donde se aloja este, y añadir "/ServletLeeDatos".
Ejemplo, si el formulario estaba en ***http://localhost:8080/contexto/formulariohobbies.html***, la URL a la cual se enviará la petición es: ***http://localhost:8080/contexto/ServletLeeDatos*** empleando el método GET.
En esa URL debe haber un Servlet preparado para procesar la información. Este formulario envía sus parámetros como parte de la URL, ya que emplea el método GET; obtendríamos la URL:

http://localhost:8080/contexto/ServletLeeDatos?nombre=David&apellidos=S%C3%A1nchez&edad=Menor&hobbies=lectura&hobbies=hacer%20deporte

Si hubiésemos indicado que el formulario se enviase empleando el método POST sólo veríamos la URL ***http://localhost:8080/contexto/ServletLeeDatos***, y el resto de la información se enviaría en el cuerpo del mensaje.

Cuando la petición llega al servidor, tanto si se envía por GET como por POST, el contenedor de Servlets recoges los parámetros del formulario y los guarda en un objeto HttpServletRequest. Podemos acceder a ellos a través de los métodos **getParameter(String)** o **getParameterValues(String)**. El parámetro de estos métodos es el atributo "name" de cada input de formulario.

Servlet que procesa el formulario anterior:

```
public class ServletLeeDatos extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                                throws ServletException, IOException {
                request.setCharacterEncoding("utf-8");
                String nombre = request.getParameter("nombre");
                String apellidos = request.getParameter("apellidos");
                String edad = request.getParameter("edad");
                String[] hobbies = request.getParameterValues("hobbies");

                response.setContentType("text/html;charset=UTF-8");
                PrintWriter out = response.getWriter();
                try {
                        out.println("<html>");
                        out.println("<head>("<title>Servlet que procesa un formulario basico</title></head>");
                        out.println("<body>");
                        out.println("<h1>" + "Hola " + nombre + " " + apellidos+"</h1>");
                        out.println("Eres " + edad + " );
                        if (hobbies!=null){
                                out.println ("Tus hobbies son:");
                                out.println("<ul>");
                                for (String hobby : hobbies) {
                                        out.println("<li>" + hobby + "</li>");
                                }
                                out.println("</ul>");
                        }
                        out.println("Este formulario ha sido invocado con Los siguientes parámetros:<br/>");
                        out.println(request.getQueryString());
                        out.println("</body>");
                        out.println("</html>");
                } finally {
                        out.close();
                }

}
```

Como hasta ahora:
- Los campos de texto se envían siempre
- Los radios, checkboxes, etc no se envían si no se selecciona ninguno. En ese caso, en Java, el parámetro contiene null y tienes que evitar posibles NullPointerException's

En el caso de los hobbies, hemos usado el método ***String [] getParameterValues (String)*** ya que el campo de formulario 'hobbies' es de selección múltiple y puede contener varios valores.
El método **getQueryString** se emplea para mostrar, con propósitos didácticos, los parámetros enviados al servidor. Ejemplo de la salida producida por el Servlet:

Hola David Sánchez

Eres "Menor" y tus hobbies son:

- lectura
- deporte

Este formulario ha sido invocado con Los siguientes parametros:
nombre=David&apellidos=S%C3%A1nchez&edad=%E2%80%9DMenor%E2%80%9D&hobbies=lectura&hobbies=deporte

Llamadas a servlets

Un servlet es un recurso web y puede ser accedido de diversos modos:

- Al enviarle un formulario
- Usando etiquetas HTML (, <img src=referencia_servlet….>, etc
- Redireccionando desde otro servlet
- Escribiendo su URL en un navegador
- …

Cuando queramos referenciar un servlet de forma absoluta, lo haremos precediendo su URL del contexto de la aplicación. Ejemplo:

Para un servlet: **@WebServlet("/ServletSaludo")** class ServletSaludo { ……..}

Y un jsp que incluya SALUDAR

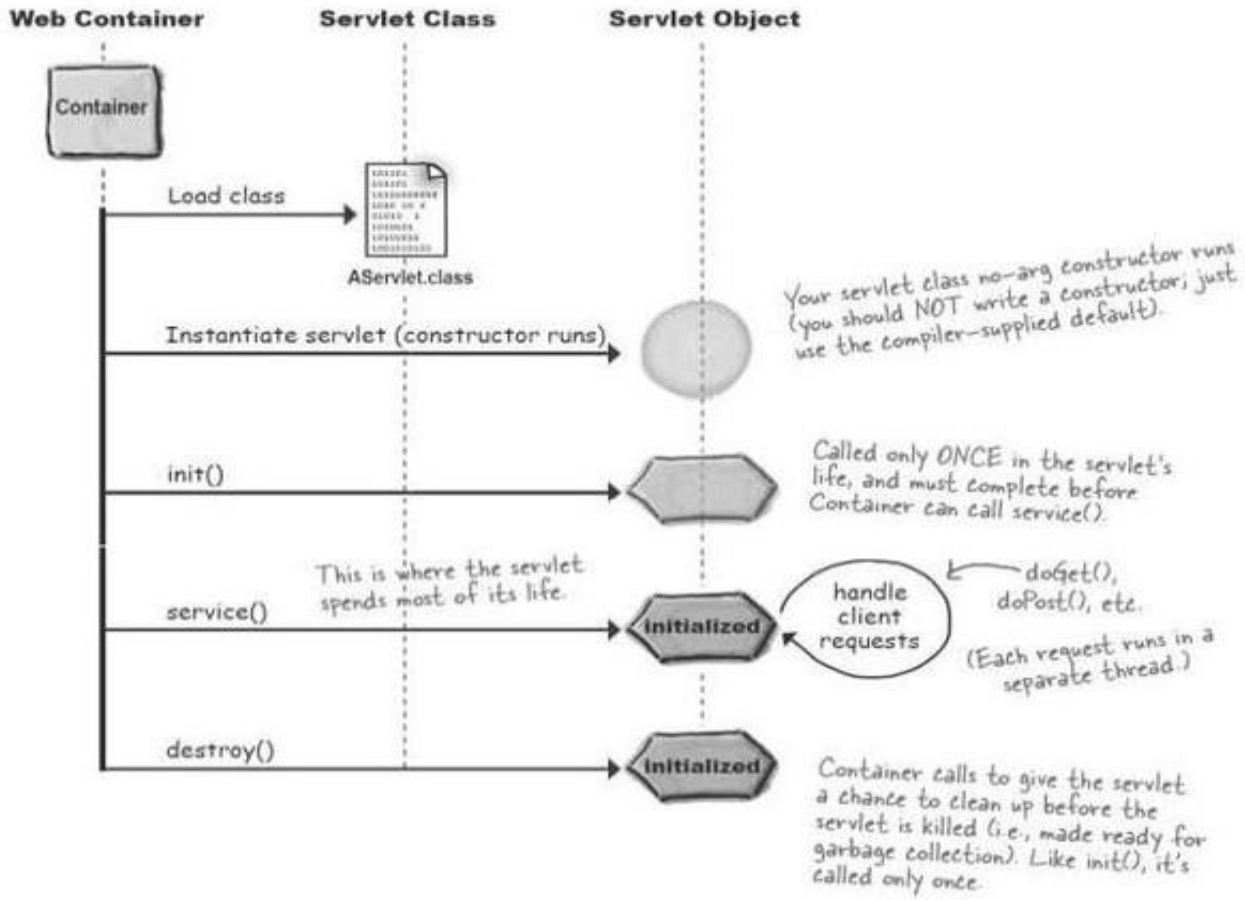
, este enlace solo funcionaría si la página jsp está en la raíz de la zona web (en webapp) y dejaría de hacerlo si reside en una subcarpeta. Para que el enlace se dirigiera al Servlet independientemente de la ubicación del jsp, tendríamos que sustituirlo por

<a href="<%= **request.getContextPath() + "/ServletSaludo"** %>"> SALUDAR

Ciclo de vida de un servlet: init, service, destroy

Los Servlets son gestionados por un contenedor de servlets (Tomcat), que controla su ciclo de vida: cuándo se crea, cuándo sirve peticiones y cuándo se destruye.
Un mismo servlet http sirve para atender peticiones web simultáneas, mediante distintos hilos.
Los servlets, una vez iniciados, quedan activos y en espera mientras la aplicación web esté en marcha.

La vida de un servlet se resume en el siguiente gráfico



1. Los servlets se **instancian** 1 única vez, normalmente en la 1ª petición que reciben (aunque podemos configurarlos para que se instancien al desplegarse la aplicación web).

2. Tras instanciarse se invoca automáticamente su método **init**, que se usa para labores de inicialización tediosas (conexiones con BD, otros equipos, etc), que haya que ejecutar una sóla vez.

   El método init ya está implementado en GenericServlet, pero si queremos personalizar la inicialización de nuestro servlet, lo podemos redefinir así:

```
public class EjemploInit extends HttpServlet {

   private int contadorVisitas;
   private MiObjeto ob;

   public void init(ServletConfig config) throws ServletException {

         //Inicializa el objeto ServletConfig y registra la inicialización
         super.init( config );

         //Inicializa las variables contados y ob
         contadorVisitas=0;
         ob=new MiObjeto();
   }
……
```

Si definimos atributos en 1 servlet, éstos tienen ámbito de "aplicación", es decir, se comparten por todas las peticiones/clientes

3. Peticiones y respuestas: método **service**

Una vez el servlet en marcha, cada petición del cliente genera una llamada al método ***service****(ServletRequest request, ServletResponse response)* del servlet.

El método service() de HttpServlet facilita la programación de servlets, llamando a otros métodos (doPost, doGet, ….) que son los que redefinirá el programador.

Ya que los servlets son multithread, distintos hilos del servlet estarán ejecutando el método service, atendiendo así múltiples peticiones simultáneas.
Para hacer que el servlet sirva un solo cliente a la vez, podría implementar el interface *SingleThreadModel*

4. Finalización ordenada: método **destroy**

Los servlets se ejecutan indefinidamente hasta que el servidor los destruye (por apagado del servidor, por petición del administrador….)
El contenedor de servlets, antes de destruir nuestro Servlet, invocará al método **destroy**(), donde podremos cerrar recursos de modo ordenado: cerrar ficheros abiertos, conexiones con bases de datos, etc. Nuestros servlets pueden redefinir ***destroy*** con tal propósito.

Configurando servlets. Clase **ServletConfig**

La clase ServletConfig es empleada por el contenedor, principalmente, para pasarle parámetros al Servlet durante su inicialización. La cabecera del método init es:

*public void init (**ServletConfig** config) throws ServletException*

¿Cómo podemos especificar los parámetros de inicialización de un Servlet?.

Añadiéndolos a la anotación @WebServlet:

Cada parámetro se indica con una anotación **@WebInitParam** y tiene 2 atributos: nombre y valor

```
@WebServlet(name="ConfigurableServlet",
urlPatterns={"/ConfigurableServlet"}, initParams={   @WebInitParam(name="parametro1", value="Valor1"),
                                                     @WebInitParam(name="parametro2", value="Valor2")  })
```

Indicándolos en el descriptor de despliegue (web.xml):

```
<servlet>
    <servlet-name>ConfigurableServlet</servlet-name>
    <servlet-class>paquete.ConfigurableServlet</servlet-class>
    <init-param>
        <param-name>parametro1</param-name>
        <param-value>Valor1</param-value>
    </init-param>
    <init-param>
        <param-name>parametro2</param-name>
        <param-value>Valor2</param-value>
    </init-param>
</servlet>
```

Si definimos el mismo parámetro con valores distintos en una anotación y en el descriptor de despliegue, el descriptor de despliegue tiene prioridad

Estos son algunos métodos de ServletConfig:

String **getInitParameter**(String name)	Devuelve el parámetro de inicialización asociado con el nombre que se le pasa como argumento.
Enumeration<String> **getInitParameterNames**()	Devuelve los nombres de los parámetros de inicialización. Si no hay ningún parámetro, la enumeración estará vacía.
ServletContext **getServletContext**()	Devuelve una referencia al contexto del Servlet
String **getServletName**()	Devuelve el nombre de esta instancia de Servlet.

sendRedirect Vs Forward

Veremos la diferencia entre estas dos formas de redireccionar:

sendRedirect

Es un método de la respuesta y pide al navegador del cliente que realice una nueva petición (mediante URL) de un recurso. En definitiva, es la máquina del cliente (el navegador) quien realiza, sin intervención del usuario, una nueva petición del recurso indicado. Esto implica:

1. Se pierden los objetos request y response de la petición precedente.
2. En la URL veremos la URL del nuevo recurso solicitado.
3. Mediante esta operación no podemos tener acceso a los ficheros contenidos en el WEB-CONTENT.

response.sendRedirect ("URL del recurso");

forward

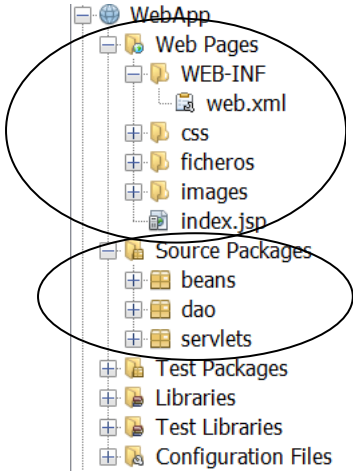
En este caso, es **el servlet quien pasa esa misma petición** contra un nuevo recurso. Esto implica:

1. **Se conservan los objetos request y response** de la petición actual (por ejemplo, los parámetros introducidos por el cliente en un formulario en la petición anterior)
2. En la URL no se verá cuál es el nuevo recurso cargado
3. Mediante esta operación podemos acceder a los ficheros contenidos en el WEB-CONTENT

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException
{
request.getRequestDispatcher("OtroServlet").forward(request, response);

 }
```

Se resume en que: *sendRedirect* es un método del objeto response y *forward* es un método del objeto request.

Estructura de una aplicación Java Web



Zona WEB:
páginas hmtl, jsp, recursos estáticos: estilos, imágenes, etc

Clases Java:
servlets, beans, clases dao (de acceso a BBDD)

- En la **zona Web**, está, entre otros el directorio **WEB-INF**, que es privado y contiene todos los recursos que no deben servirse directamente al cliente.
  Aquí está el archivo web.xml donde se establece la configuración de la aplicación web y las librerías. (También almacena las clases y librerías de Java compiladas una vez se despliegue el proyecto)

- La zona "Source Packages", almacena clases Java.
  Señalar que estas clases no pueden acceder a recursos de la zona Web directamente por su ruta.

Para que un servlet acceda a un archivo de la zona web, podemos elaborar su "ruta" desde el contexto, utilizando:

….getServletContext().getRealPath(archivo);

Ejemplo: acceso desde 1 servlet a fichero de texto de la zona web

```
class MiServlet extends HttpServlet(……..,……..){

   void doPost (…..){

     FileReader fw=new FileReader("ficheros/fich.txt");  //MAL
     FileReader fw=new FileReader                        //BIEN
           (this. getServletContext().getRealPath("ficheros/fich.txt"));
   }
```

El método getRealPath() no es seguro ni útil para una aplicación real :

- Expone una ruta absoluta dentro del disco del servidor y
- Puede fallar en algunos entornos de producción: por ejemplo, si el contenedor del servlet no expande el archivo WAR al desplegarlo (como Tomcat). En esos casos, es mejor utilizar:

URL recurso=….getServletContext().getResource("archivo");

O si necesitamos un flujo de entrada a ese archivo:

InputStream contenido=….getServletContext().getResourceAsStream("archivo");

Sesiones en Java

El seguimiento de sesión es un mecanismo que los servlets utilizan para mantener el estado sobre una serie de peticiones durante un periodo de tiempo, que pertenecen al mismo navegador .
La sesión es compartida por los servlets a los que accede el cliente. Así, una aplicación compuesta por varios servlet es capaz de mantener el estado (compartir datos entre ellos) mediante el uso de sesiones
Para utilizar el seguimiento de sesión debemos:

- Obtener una sesión (un objeto **HttpSession**) para un usuario.
- Almacenar u obtener datos desde el objeto **HttpSession**.
- Invalidar la sesión (si así se desea).

1. **Obtener una Sesión** desde la petición

HttpSession ***getSession***(boolean create)

El método **getSession** del objeto **HttpServletRequest** devuelve una sesión de usuario.
Tiene un argumento de entrada para indicar si crear la sesión (en caso de no existir)

- Cuando el argumento es **true**, "inicia una nueva sesión en caso de no existir ya". Es el valor por defecto.
- Si pasamos false como argumento y no existe una sesión, getSession simplemente devolverá null

Atención:
Si el Servlet responde utilizando un **PrintWriter**, debemos llamar a **getSession** antes de acceder al **PrintWriter**

```
public class CatalogoServlet extends HttpServlet {
        public void doGet (HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException,IOException{
                HttpSession session = request.getSession(true);
                out = response.getWriter();
                ...
}}
```

2. Manejar Atributos de sesión

Un atributo de sesión es un objeto Java, identificado por un nombre(String), que está asociado a la sesión.
HttpSession dispone de los métodos setAttribute y getAttribute para añadir y consultar atributos de la sesión.

void **setAttribute** (String nombre, Object valor);

Object **getAttribute**(String nombre);

Ejemplo:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ……. {
        String nombre = request.getParameter("nombre");
        String password = request.getParameter("password");
        DbUser user = buscarEnBD(nombre, password);
        HttpSession sess = request.getSession(true);
        sess.setAttribute("USER", user);
        if (user == null) {
                // Mensaje "Error usuario-password"
        } else {
                // Mensaje "Login OK"
        }
}
```

En este ejemplo, suponemos que buscarEnDB() devuelve un objeto de la clase DbUser, que encapsula datos de un usuario determinado recuperados de una base de datos. Asumimos que este método devuelve null si el usuario no existe o si la contraseña no es la correcta.
En cualquier caso, añadimos a la sesión un atributo de nombre "USER" que contiene como valor un objeto DbUser con los datos de un usuario
Este código borraría toda la información de un usuario previo, cuando se realiza un login erróneo

Siguiendo el ejemplo, una vez que un usuario se ha logueado, podríamos realizar un seguimiento del mismo (conocer información del usuario actualmente conectado) utilizando el método getAttribute( )

```
private DbUser getCurrentUser(HttpServletRequest request) {
        HttpSession sess = request.getSession(false);
        if (sess == null)
                return null;
        return (DBUser) sess.getAttribute("USER");  //Tener en cuenta que un casting de null dará error
}
```

3. Invalidar la Sesión

Una sesión de usuario puede ser invalidada manual o automáticamente, dependiendo de dónde se esté ejecutando el servlet. (Las aplicaciones Java suelen invalidar una sesión cuando no hay peticiones de página por un periodo de tiempo de 30 minutos).
Invalidar una sesión significa eliminar el objeto **HttpSession** y todos sus atributos del sistema.

Para eliminar todo rastro de una sesión del servidor, llamaremos al método invalidate() de la sesión

```
private void doLogout(HttpServletRequest req) {
        HttpSession sess = req.getSession(false);
        if (sess != null) {
                sess.invalidate();
        }
}
```

Reescritura de URL

Por defecto, el seguimiento de sesión utiliza cookies para asociar un identificador de sesión con un usuario. Si queremos dar soporte a usuarios que accedan a un servlet con un navegador que no soporta cookies, se debe utilizar **reescritura de URL.** (como en php)

La reescritura de la URL consiste en que el servidor añada en las URLs presentes en la respuesta HTTP transmitida al cliente, un parámetro correspondiente al identificador de sesión.
Este mecanismo no es muy complejo de implementar gracias a los métodos

String **encodeURL** (String url)
y
String **encodeRedirectURL**(String url)

disponibles en el objeto HttpServletResponse.

Estos 2 métodos transforman la URL recibida, añadiéndole un parámetro correspondiente al identificador de sesión. Además, contienen un mecanismo que permite verificar automáticamente si no hay otra solución (las cookies) para transmitir el id de sesión al cliente. Si detecta que el cliente tiene cookies activadas, no modifica la cadena de caracteres que se les ha pasado por parámetro.

Ejemplo 1

- Caso de un formulario HTML generado por un servlet.

```
  out.println("<FORM action=\"" +
response.encodeURL("/riJEE/Seguir") +"\">");
```

el código HTML construido por esta instrucción

```
<FORM
action="/riJEE/Seguir;jsessionid=43A9FBA967BBA4B68AE3114B7E4745CA">
```

Ejemplo 2

```
String originalURL = someURL;
String encodedURL = response.encodeRedirectURL(originalURL);
response.sendRedirect(encodedURL);
```

Métodos de HttpSession

Object **getAttribute**(String nombreAtributo)	Devuelve el objeto de sesión identificado por el nombre pasado como parámetro. Ya que este método devuelve un objeto de la clase genérica Object, a la hora de recuperarlo se deberá realizar el cast correspondiente. Devuelve null si el objeto indicado no existe.
Enumeration **getAttributeNames()**	Devuelve una Enumeration con los nombres de todos los objetos almacenados en la sesión actual.
long **getCreationTime()**	Devuelve la fecha y hora en la que fue creada la sesión, como timestamp (milisegundos desde el 1 de enero de 1970)
String **getId()**	Devuelve el identificador único asignado a la sesión. Este valor se corresponde con el valor de el cookie.
JSESSIONID	utilizada para poder realizar el mantenimiento de la sesión. En caso de no utilizar cookies, se corresponde con la información que se añade al final de cada enlace cuando se utiliza el mecanismo de reescritura de URLs.
long **getLastAccesesedTime()**	Devuelve como timestamp el momento de la última petición realizada en la sesión actual
int **getMaxInactiveInterval()**	Devuelve el máximo intervalo de tiempo, en segundos, en el que una sesión puede permanecer activa, sin que el cliente realice ninguna petición. Suele ser de 1800segundos. Una vez transcurrido este tiempo el contenedor de destruye la sesión
void **invalidate()**	Destruye la sesión de forma explícita, y libera de memoria todos sus objetos (atributos)
boolean **isNew()**	Devuelve si la sesión se acaba de crear en la petición actual.
void **removeAttribute(**String nombreAtributo**)**	Elimina el objeto almacenado en la sesión cuyo nombre recibe. Si el nombre no se corresponde a un atributo de sesión, no realizará ninguna acción.
void **setAttribute(**String nombre, Object valor**)**	Almacena un objeto en la sesión con el nombre indicado como primer parámetro
void **setMaxInactiveInterval(**int intervalo**)**	Establece el máximo tiempo que una sesión puede permanecer inactiva antes de ser destruida por el contenedor de servlets, en segundos.

Cookies en Java

1. Crear una Cookie: new Cookie (nombre, valor)

   Cookie libro = **new Cookie**("comprar", "301");

2. Asignar atributos

   - Asignar su duración en segundos: setMaxAge (num_segundos)
     libro.**setMaxAge**(60*60*24);
     libro.setMaxAge(0); //se borra

   Las cookies duran hasta el cierre del navegador, si no les damos mayor duración, con este método.

   - Asignar un valor nuevo: setValue(String)
     libro.**setValue**("Madame Bovary");

   - Asignar un comentario: setComment(String)
     libro.setComment("Almacena el código del libro");

3. Envío de la cookie: Método addCookie (Cookie)
   Se añaden con el metodo addCookie de la clase HttpServletResponse, antes de llamar al metodo getWriter()

   response.addCookie(libro);
   PrintWriter out = response.getWriter();

4. Leer, de vuelta, valores de cookies: Método getCookies() de HttpServletRequest

   Cookie[ ] **getCookies**( ) → null si el navegador cliente no ha enviado cookies

```
Cookie[] cookies = request.getCookies();
if (cookies != null)
    for (Cookie ck : cookies) {
                if ("prefResultsPerPage".equals(ck.getName())) {
                    String prefValue = ck.getValue();
                ..
   }
}
```

Atributos: ámbitos

Los atributos usados por los servlets, son OBJETOS (de cualquier tipo) accesibles por un nombre.

Así, para establecer, borrar y acceder a atributos, utilizaremos los métodos:

- …..**setAttribute**("nombre", objeto);
- …..**removeAttribute**("nombre");
- Objeto objeto= (Objeto)....**getAttribute**("nombre");

respectivamente.

Es importante diferenciarlos de los parámetros, que ya conoces.
Los parámetros son cadenas de texto asociadas a un nombre, que nos llegan bien desde campos de formulario o bien desde la URL (……?param1=valor1&param2=valor2&param3=valor3….).
Un parámetro no puede contener un objeto, sólo un String

Un atributo, por el contrario, ES UN OBJETO DE CUALQUIER TIPO y puede ser definido en distintos ámbitos (scopes), dependiendo de su alcance o duración (de cuánto queremos que dure el atributo)

En este momento, distinguiremos estos 3 ámbitos para 1 atributo:

- Ámbito de petición o request:
  El atributo tiene vigencia en la petición http actual

  Ejemplo: Atributo para guardar un error puntual que se visualizará en la siguiente página

- Ámbito de sesión o session: El atributo tiene vigencia durante la sesión de navegación de un cliente.

  Ejemplo: Atributo para guardar el carro de compra del usuario actual (que podría ser un HashMap o cualquier colección compleja)

- Ámbito de contexto o application: El atributo tiene vigencia mientras el servlet esté atendiendo peticiones. Para compartir datos entre varias sesiones.
  - Estableceremos los atributos de ámbito “aplicación” en 1 lugar que se ejecute una única vez, durante el tiempo de vida de la aplicación web, por ejemplo, en el método init de un servlet.
  - Desde un servlet, accederemos a un atributo de ámbito “aplicación” mediante la sentencia:
    this.getServletContext().getAttribute(“nombreatributo”);

  Ejemplo: Atributo para guardar en forma de HashMap una tabla de nuestra BD, muy accedida por todos los clientes y que no cambia (con el objetivo de reducir el nº de accesos a la BD)