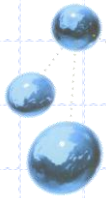
CC5406 Aplicaciones Empresariales con JEE



Servlets

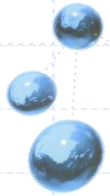
Aplicaciones Servidor Web en **Java**

Profesores: Andrés Farías

v2.0

Objetivos: aprender a...

- Servlets.
- Envío de parámetros a un Servlet.
- Session Tracking.
- Servlets Listeners.
- Servlets Filters.

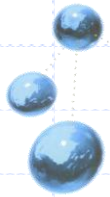


Truck Factor

- Definición
  - ✓ Corresponde al número de personas en el equipo que tienen que ser atropelladas por un camión (truck) antes que el proyecto se encuentre en serios problemas.
- Lógica
  - ✓ También puede ser interpretado como el número de integrantes que toman vacaciones de manera simultánea…
  - ✓ Es un factor a considerar cuando el proyecto se basa en tecnologías no muy populares.
  - ✓ También se traduce en la importancia de las personas.
    - Todo proyecto tiene héroes, por ejemplo, que conocen componentes críticas del sistema.
    - Cuando ellos se van hay que estar preparados para lo que sigue.
- Métrica
  - ✓ Permite determinar cuán caro será reemplazar una persona específica.
  - ✓ Si se da por un tema de tecnologías, entonces se pueden tomar otras medidas:
    - Tener una lista de posibles candidatos para el reemplazo,
    - Entrenar un equipo en paralelo.
    - Preocuparse de la estabilidad de ellos respecto a su medio de trabajo.
  - ✓ Si se trata de un héroe, entonces hay que tratar de reasignarlo a otra parte del sistema, así aun está disponible para dar soporte y el conocimiento puede ser transmitido a otros.

Serulets

1. Información general
2. Ciclo de vida
3. Recuperación de parámetros.
4. Contexto del Servidor.
5. Clases Java Notables.
6. Ejemplos.

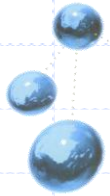


¿Qué son los Serulets?

- **Serulets** son módulos que extienden las funcionalidades de un servidor "java-enabled".
- Piedra angular del desarrollo de aplicaciones web en Java.
- Pensados para reemplazar a los CGIs
  - ✓ Cada petición de un cliente hacía al servidor lanzar un nuevo proceso (pesado) con el programa CGI
- Normalmente generan código HTML dinámicamente, el cual se manda al browser, responsable de su render (dibujado).
- Por ejemplo, pueden mandar una consulta a una base de datos, la que puede estar basada en parámetros que mandó el browser al servidor. El resultado se manda en formato HTML.

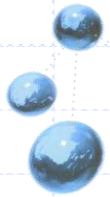
Ventajas

- Rendimiento.
  - ✓ Cada petición es procesada por un único proceso en el contenedor de servlets.
- Portabilidad.
  - ✓ Heredado de Java
- Rápido desarrollo.
  - ✓ Acceso a las riquísimas librerías de Java.
- Robustez.
  - ✓ Son gestionados por la máquina virtual de Java (garbage collection)
- Amplio soporte.
  - ✓ Muchos desarrolladores y compañías utilizan esta tecnología



Ciclo de Vida

Servlets

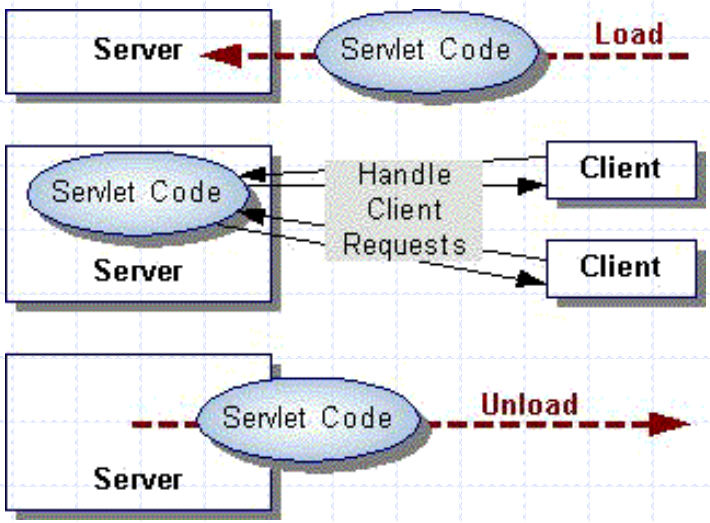


Respecto al servidor

1. Carga del Servlet.
   - ✓ El servidor es cargado (como clase Java) en el servidor.
   - ✓ La carga se realiza a través de un **Class Loader**.

2. Atención de peticiones.
   - ✓ Los clientes envían peticiones al servidor de aplicaciones.
   - ✓ El servidor de aplicaciones determina el Servlet asociado al recurso solicitado por la petición
   - ✓ Se transmite la petición al Servlet.
   - ✓ El Servlet la responde.

3. Descarga del Servlet.
   - ✓ El Servlet es descargado del Servidor de aplicaciones.



Ciclo de vida de un Serulet

- Viene dado por tres métodos: **`init()`**, **`service()`** y **`destroy()`**.

- Inicialización:
  - ✓ Una única llamada al método **`init()`** por parte del servidor.
  - ✓ Incluso se pueden recoger unos parámetros concretos con **`getInitParameter()`** de la clase **`ServletConfig`**.

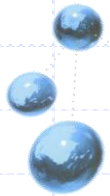
- Servicio:
  - ✓ Se realiza una llamada a **`service()`** por cada invocación al Servlet.
  - ✓ Atención El contenedor es multi-hilo (multi-thread). 

- Destrucción:
  - ✓ Cuando todas las llamadas desde el cliente cesen o un temporizador del servidor así lo indique. Se usa el invoca el método **`destroy()`**.

Reuisar documentación de la clase
`javax.servlet.Servlet`

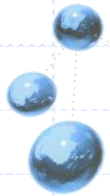
Un ejemplo

```
public class CicloVidaServlet implements Servlet {

  public void init(ServletConfig config) throws ServletException {
    System.out.println("init");
  }

  public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException {
    System.out.println("service");
  }

  public void destroy() { System.out.println("destroy"); }

  public String getServletInfo() { return null; }

  public ServletConfig getServletConfig() { return null; }
}
```



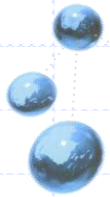
Declaración en el archivo web.xml

```
<weapp>
    ...

     <servlet>
        <servlet-name>
            LifeCycle
        </servlet-name>

        <servlet-class>
          cl.dcc.uchile.cc5604.materias.servlets.LifeCycleServlet
       </servlet-class>

       <servlet-mapping>
          <servlet-name>LifeCycle</servlet-name>
          <url-pattern>begin</url-pattern>
       </servlet-mapping>

     ...
     </servlet>
</weapp>
```



Recuperación de parámetros

Servlets

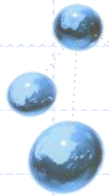


Parámetros en el archivo web.xml

```
<weapp>
    ...

     <servlet>
        <servlet-name>
             ConfigDemoServletExample
        </servlet-name>

        <servlet-class>ConfigDemoServlet</servlet-class>

        <init-param>
            <param-name>adminEmail</param-name>
            <param-value>dipina@ilargi.org</param-value>
        </init-param>

        <init-param>
            <param-name>adminContactNumber</param-name>
            <param-value>6542214213</param-value>
        </init-param>
     </servlet>
</weapp>
```

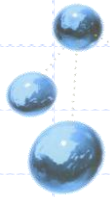
Recuperación desde el Serulet

```
public class ConfigDemoServlet implements Servlet {

   ServletConfig cfg;

   public void init(ServletConfig config) throws ServletException {

      this.cfg = config;

      Enumeration parameters = config.getInitParameterNames();

      while (parameters.hasMoreElements()) {
        String parameter = (String) parameters.nextElement();
        System.out.println("Parameter name: " + parameter);
        System.out.println("Parameter value:" + config.getInitParameter(parameter));
       }
    }

   ...
}
```



Contexto del Servidor

Servlets



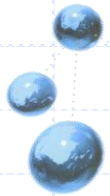
Conceptos generales

- `ServletContext` representa el entorno donde se ejecuta el servidor, es decir, el Serulet Container.

- Se puede obtener una referencia a él mediante el método `ServletConfig.getServletContext()`.

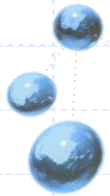
- Algunos métodos que se pueden utilizar son:
  - ✓ `getMajorVersion()` de la Servlet API soportada por el contenedor
  - ✓ `getMinorVersion()`.
  - ✓ `setAttribute(String, String)`. Guarda un objeto en el `ServletContext`.
  - ✓ `getAttributeNames()`.
  - ✓ `getAttribute()`.
  - ✓ `removeAttribute(String)`.

- A través del contexto de un Servlet se puede compartir un estado entre varios Servlets.



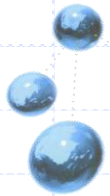
Recuperando los atributos del contexto

```
ServletConfig servletConfig;

public void init(ServletConfig config) throws ServletException {
  servletConfig = config;
}

public void service(ServletRequest request, ServletResponse response)
  throws ServletException, IOException {

  ServletContext servletContext = servletConfig.getServletContext();
  Enumeration attributes = servletContext.getAttributeNames();

  while (attributes.hasMoreElements()) {
    String attribute = (String) attributes.nextElement();
    System.out.println("Attribute name : " + attribute);
    System.out.println("Attribute value : " + servletContext.getAttribute(attribute));
  }

  System.out.println("Major version : " + servletContext.getMajorVersion());
  System.out.println("Minor version : " + servletContext.getMinorVersion());
  System.out.println("Server info : " + servletContext.getServerInfo());
}
```



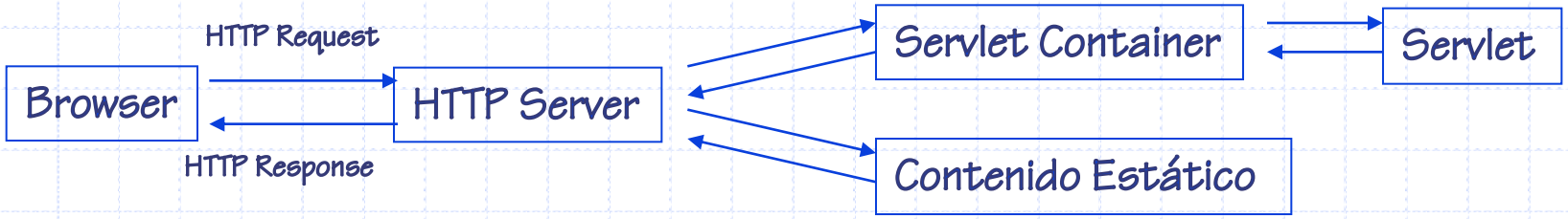
Clases Java notables

Servlets



Arquitectura

- Si un Servidor de Aplicaciones Web posee un **Serulet Container**, quiere decir que el servidor web es capaz de ejecutar Servlets.
- Cuando una petición es enviada desde el cliente (browser), el servidor la transfiere al **Serulet Container**, transfiriéndoselo a su vez al Servlet que debe atender la petición.
- Para transferir la petición al Servlet correspondiente le contenedor obtiene esta información a partir de un mapeo obtenido desde el archivo de configuración **`web.xml`**.
- El Servlet atiende la petición y genera la respuesta, creando y configurando el objeto que representa esta respuesta (**`response`**).



La clase HttpSerulet: Anatomía

- La mejor manera de escribir un Servlet es extendiendo (heredando) desde la clase `HttpServlet`. Esta clase:
  - ✓ Posee variables de entorno relativas a la sesión entre el servidor y el cliente.
  - ✓ Posee una implementación adecuada de los métodos del ciclo de vida: `init()`, `service()` y `destroy()`.
  - ✓ Tiene métodos listos para sobre-escribir de acuerdo a los verbos/métodos más importantes: `doGet()`, `doPost()`, `doDelete()`, etc.

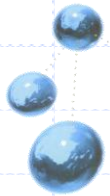
- Método `doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException`
  - ✓ Es invocado cuando el Servlet recibe una petición con el método `GET`.

- Método `doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException`
  - ✓ Es invocado cuando el Servlet una petición con el método `POST`.

- Método `doDelete(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException`
  - ✓ Es invocado cuando el Servlet una petición con el método `DELETE`.

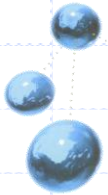
La clase HttpServlet Ejemplo

```
public class MyServlet extends HttpServlet {

    public void init() {
        // Sobre-escribir para que haga lo que queramos
    }

    public void doGet ( HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException
    {
        //Sobreescribir para que haga lo que queramos
    }

    public void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException
    {
        //Sobreescribir para que haga lo que queramos
    }
}
```



La clase HttpSeruletRequest

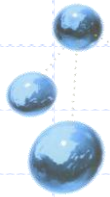
- **`HttpServletRequest`** es la clase que representa una petición procedente de un cliente.
- Esta clase provee acceso a:
  - ✓ Información acerca del cliente
    - Los parámetros que envió,
    - La versión del protocolo que está usando,
    - La IP del cliente,
    - etc.
  - ✓ Una variable asociada a la petición de tipo **`ServletInputStream`** que puede ser usada por el servidor para recibir información adicional, como por ejemplo:
    - Un archivo que el cliente quiere "uplodear" cuando se ha usado el método POST o PUT.
- Esta clase es usada para encapsular la petición entrante y entregada como argumento de los métodos de esta clase, tales como **`doGet()`**, y **`doPost()`**.



La clase HttpSeruletResponse

- **`HttpServletResponse`** es la clase que representa la respuesta que será enviada al cliente cuando el Servlet termine de atender la petición.

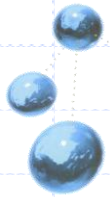
- Esta clase provee métodos para :
  - ✓ Decirle al browser cuál es el tipo MIME de la respuesta que se le va a dar al cliente.
  - ✓ Obtener un objeto **`PrintWriter`**, a partir del objeto **`ServletOutputStream`**, a través del cual se puede especifcar/construir el código HTML dinámicamente al cliente.
  - ✓ Mandar otras informaciones importantes (cookies, redireccionamineto, refresco, etc...)



Ejemplos

Servlets

1. Enviando la fecha del sistema.
2. Contador.



Enviando la fecha del sistema

```
public class SimpleServlet extends HttpServlet   {

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

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    //send data
    out.println("<HTML>");
    out.println("<H1>  Mi Primer Servlet </H1>");
    out.println("<BR> <H2>Fecha y hora: " + (new Date()) + "</H2>");
    out.println("</HTML>");
    out.close();
  }
}
```

Se indica el tipo de contenido

Se abre un canal para la respuesta.

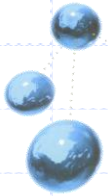
Se incluye la fecha del sistema.

Requerimientos para un contador

- Implementar un contador web.

- Requerimientos funcionales:
  - ✓ Cuenta cuántas veces el servlet recibe peticiones (a través del método **GET**).
  - ✓ También le mostrará al cliente su dirección IP.

- Requerimientos no funcionales:
  - ✓ Nombre del contexto: **counter**.



Implementando un contador

```
public class Counter extends HttpServlet {

    private int count = 0;

    public void doGet(HttpServletRequest req, HttpServletResponse res)
          throws ServletException, IOException {

          count++;
          res.setContentType("text/html");
          PrintWriter out = res.getWriter();

          out.println("<H1>Contador de accesos </H1>");
          out.println("<HR>");
          out.println("Este servlet ha sido accedido " + count + " veces");
          out.println("<HR>");
          out.println("La IP de su computador es: " + req.getRemoteHost());
          out.println("<HR>");
          out.close();
    }
}
```

Contador iniciado en cero

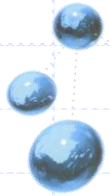
Incrementa el contador cada vez que se recibe una petición con el método get

Se imprime el contenido al response

Preguntas...

- ¿Qué pasa si el servidor se cae y se reinicia?
  - ✓ Contador volverá a 0
  - ✓ Para recordar el valor que tenía esta variable incluso si se cae el sistema la iremos escribiendo en un archivo cada vez que cambia su valor.

- Si el servlet **Counter** está atendiendo peticiones, y el valor desplegado la última vez fue 10 ¿Qué valor desplegará si otro cliente hace una petición a la misma URL?
  - ✓ Desplegará el valor 11.



Variables estáticas

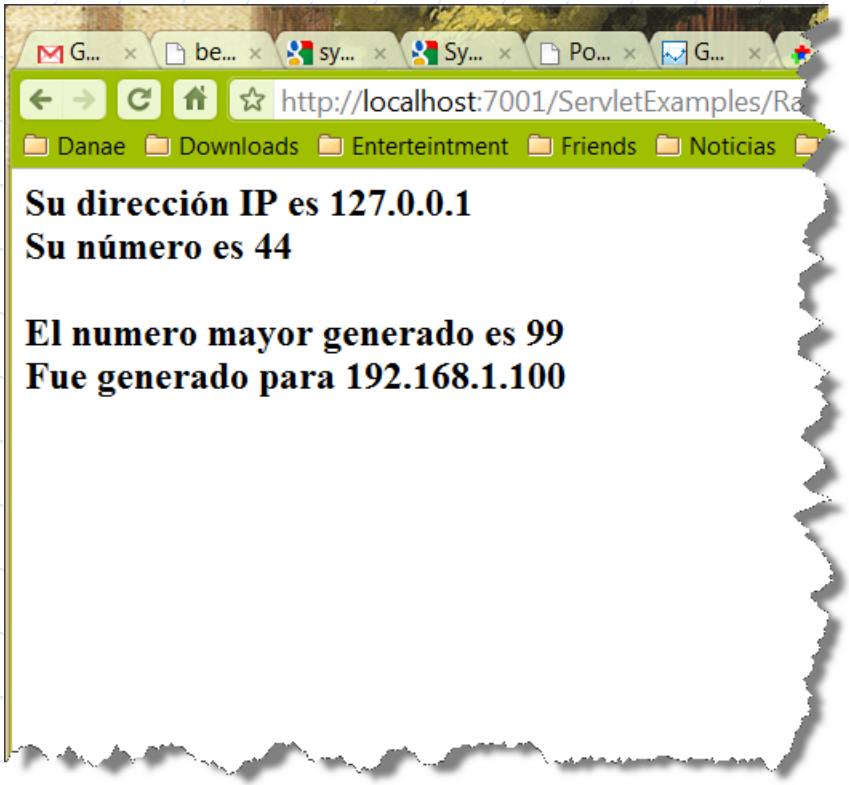
- Requerimientos Funcionales:
  - ✓ Escriba un servlet que genere un número aleatorio entre 1 y 100 cada vez que es contactado y lo muestra al cliente junto con la IP del computador del cliente.
  - ✓ Además muestra el número más grande generado hasta ahora y la IP del computador para el cual lo generó.

- Tips:
  - ✓ La dirección del computador del cliente puede ser obtenida de la siguiente manera

```
String s = request.getRemoteHost();
    Retorna el número IP (133.8.109.158)
String s = request.getRemoteAddress();
    Retorna el nombre si puede ser recuperado (www.waseda.jp)
```

  - ✓ Un número aleatorio entre 1 y 100 se genera con la siguiente instrucción en Java
    - `int numero = (int)(1 + Math.random()*100);`

Como se debe ver...



La solución

```
public class NewServlet extends HttpServlet {

    int maxNumber = 0;
    String maxIP = "";

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

      int aleatorio = (int)(1 + Math.random()*100);
      String ip = request.getRemoteAddr();
      if (aleatorio > maxNumber) {
            maxNumber = aleatorio;
            maxIP = ip;
      }

      response.setContentType("text/html"); PrintWriter out = response.getWriter();

      out.println("<h2> Su dirección IP es "+ ip+ "<br>");
      out.println("Su número es "+aleatorio+"<br><br>");
      out.println("El numero mayor generado es "+maxNumber);
      out.println("<br>Fue generado para  "+maxIP);
      out.close();
   }
}
```

Variables estáticas para almacenar los campeones!

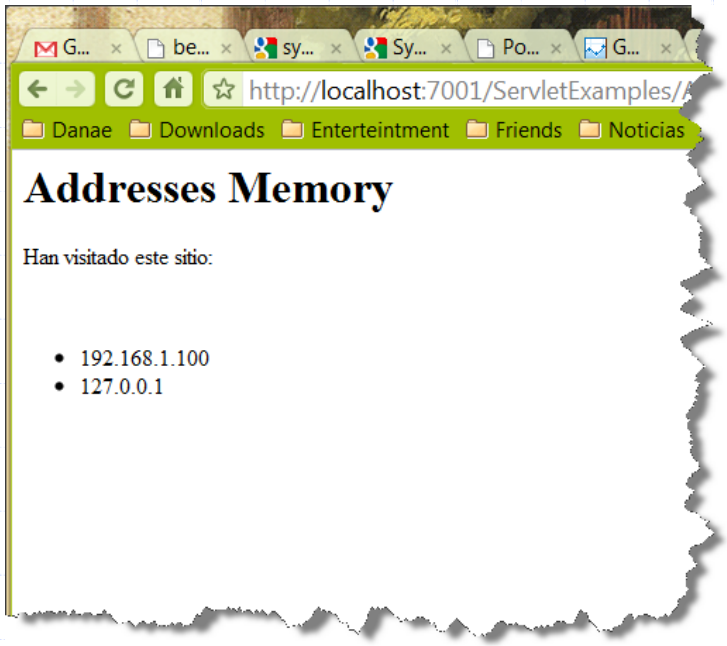
Se genera el número random y se obtiene la IP del cliente.

Actualización del campeon.

Se enuía la información al cliente.

Memoria de las IP visitadas

- Requerimientos Funcionales:
  - ✓ Escriba un Servlet que muestre la IP de todos los computadores que lo han contactado hasta ese momento.



- Tips:
  - ✓ Use un vector o un arreglo para ir guardándo las direcciones IP de los clientes.

Implementación concreta

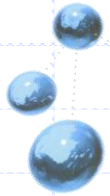
```
public class AddressesServlet extends HttpServlet {

    ArrayList<String> ips = new ArrayList<String>();

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

        /* Se obtiene la IP del cliente */
        String ip = request.getRemoteHost();
        ips.add(ip);

        response.setContentType("text/html"); PrintWriter out = response.getWriter();

        out.println("<html>"); out.println("<body>");

        out.println("<h1>Addresses Memory</h1>");
        out.println("<p>Han visitado este sitio: </p><br>");

        out.println("<ul>");
        for (String laIP : ips) out.println("<li> " + laIP + "</li>");
        out.println("</ul>");

        out.println("</body>"); out.println("</html>");
        out.close();
    }
```

Se obtiene la IP del cliente y se almacena en el arreglo.

Imprime cada IP separado por una línea.

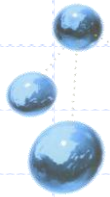
Enviío de parámetros

1. Enviándole datos al Servlet!
2. Ejercicios
3. Elementos de Input HTML
4. Nombres de Parámetros



Enviándole datos al Serulet

Envió de parámetros



Vía URL (vía método get)

- Una de las funcionalidades que hacen que la web sea interactiva es el paso de parámetros (información) del cliente al servidor.
- El cliente puede traspasar parámetros al servidor con el requerimiento (URL) de la siguiente manera:
  - ✓ `http://host:port/servlet?param1=value1&param2=value2`
  - ✓ Esto implica que el servidor recibirá 2 parámetros:
    - Uno con el nombre **`param1`** y valor **`value1`**, y
    - el otro con el nombre **`param2`** y valor **`value2`**.

- Un Servlet puede consultar por estos parámetros de la siguiente manera:
  - ✓ `String valueOfParam1 = request.getParameter("param1");`
  - ✓ `String valueOfParam2 = request.getParameter("param2");`

- Los nombres y valores de los parámetros son de tipo **`String`**.
- Los nombres difieren en el caso de contener mayúsculas y minúsculas (**`Param1 != param1`**)

Ejemplo

- URL enviada:
  http://host:port/MyServlet?firstname=Nelson&lastname=Baloian

```
public class ServletParameter1 extends HttpServlet {

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

         PrintWriter out = response.getWriter();
         response.setContentType("text/html");

         // Se obtiene el valor del parámetro "firstname"
         String fname = request.getParameter("firstname");

          // Se obtiene el valor del parámetro “lastname"
         String lname = request.getParameter(“lastname");
         out.println(<h1> "Hello "+fname+" “+lname</h1");
         out.close();
    }
}
```

Vía un formulario web

- Un formulario es una página HTML que contiene objetos gráficos que recolectarán la información y la traspasarán automáticamente al servidor con la URL.

Inicio del formulario y declaración del action

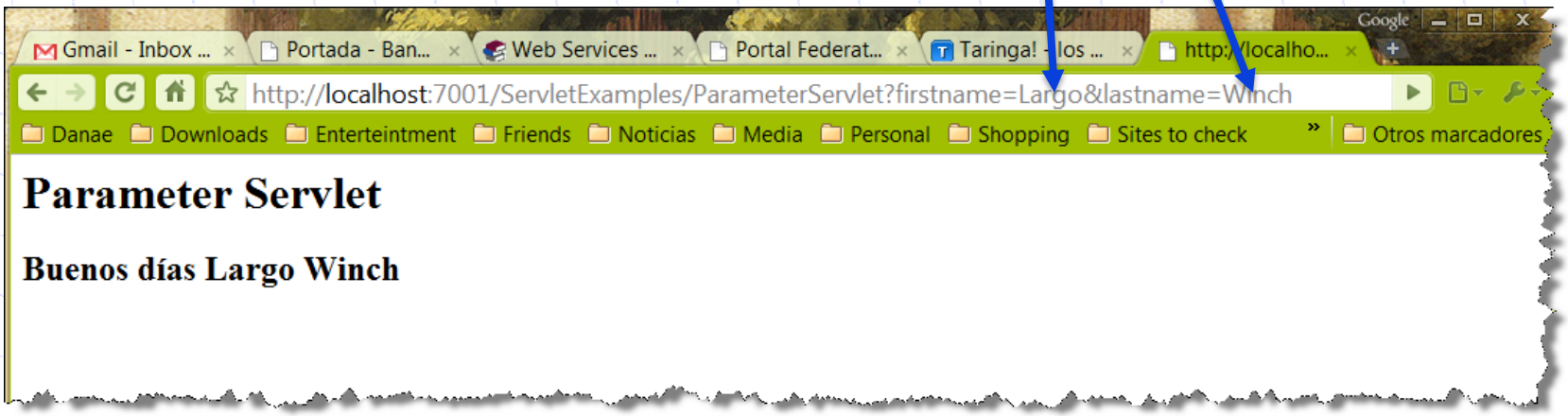
```
<h1>Ingreso de par&aacute;metros</h1>

<form action="/ParameterServlet">
   Nombre: <input type="text" name="fistname"><BR>
   Apellido: <input type="text" name="lastname"><BR>
   <input type="submit" value="Enviar">
</form>
```

Input elemento de entrada de datos, Type define el tipo (texto), y name es el nombre del parámetro.

Type Submit es para enviar la petición con los parámetros a la acción indicada.

Enuío uía método get

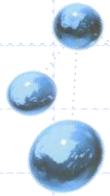
- Al oprimir el botón se obtiene el resultado que muestra la figura.
- Fijarse en la URL que se generó automáticamente con los parámetros.



Ejercicios

Envío de parámetros

1. Jalisco.



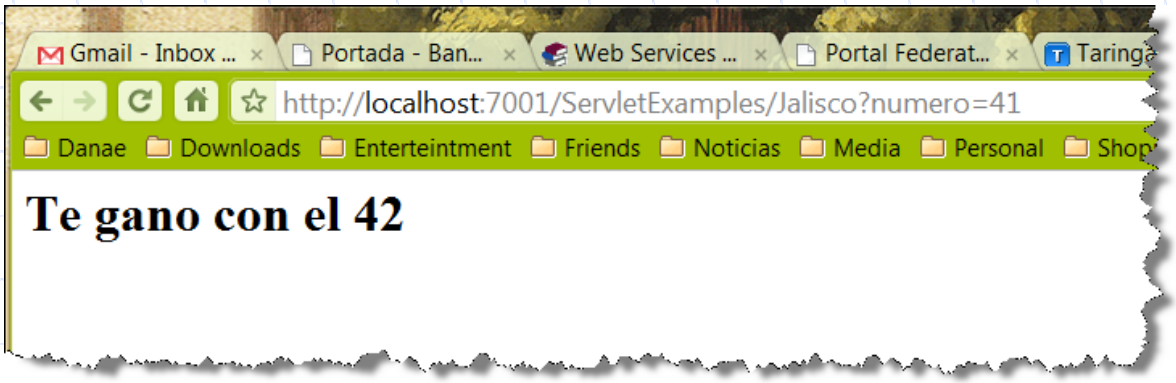
¿Qué página generan estos recursos?

```
<HTML>

<FORM ACTION="/Jalisco">
    Ingresa un número cualquiera y luego oprime el botón:
    <INPUT TYPE="TEXT" NAME="numero"><BR>
    <INPUT TYPE="SUBMIT" VALUE="Jugar">
</FORM>
</HTML>
```

¿Qué página generan este Serulet?

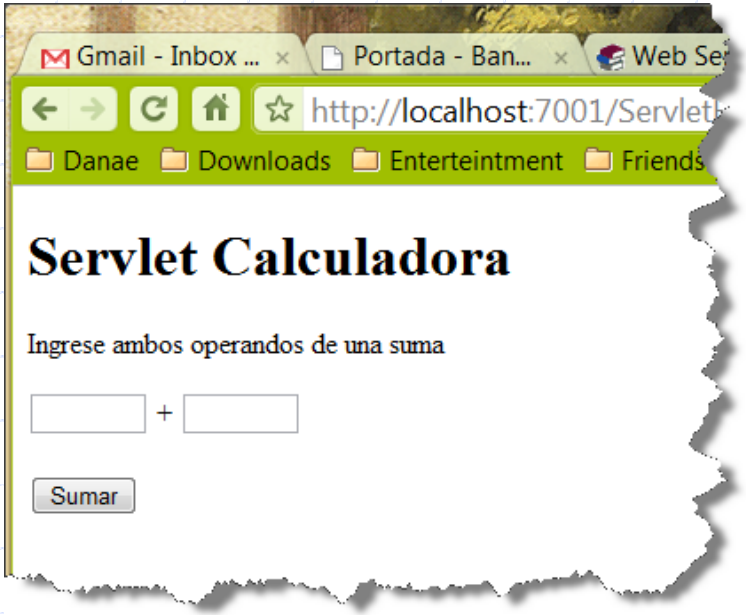
```
public class JaliscoServlet extends HttpServlet {

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

            PrintWriter out = response.getWriter();
            response.setContentType("text/html");

            int elNumero = Integer.parseInt(request.getParameter("numero"));

            out.println("<h1>Te gano con el " + (elNumero + 1) + "</h1>");
            out.close();
    }
}
```

http://host:port/Jalisco?numero=41



¿Qué página genera este HTML?

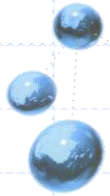
```
<h1>Servlet Calculadora</h1>

<p>Ingrese ambos operandos de una suma.<br>
<form ACTION="/CalculadoraServlet">
    <INPUT TYPE=TEXT SIZE=5 NAME="op1"> + <INPUT TYPE=TEXT SIZE=5 NAME="op2"><BR>
<BR>
<INPUT TYPE=SUBMIT VALUE="Sumar">
</form>
</body>
</html>
```



Propuesto

- Escriba el Servlet **`CalculadoraServlet`** que al ser contactado responda con la suma de ambos números.

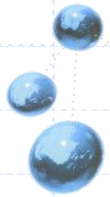
- Modifique el **HTML** de modo que la operación también sea ingresada por el usuario y el servlet haga la operación adecuada (sólo se permite **+**, **-**, *****, y **/**)



Elementos HTML de Input

Envío de parámetros

1. Radio Button.
2. Select.
3. TextArea.
4. CheckBox.



Otros elementos HTML para input

- **Radio Button** (botón radial):
  - ✓ Se puede seleccionar una alternativa entre varias
- **Select** (Selector):
  - ✓ Como el radio pero con un menú desplegable.
- **TextArea** (Área de Texto):
  - ✓ Como el texto pero contiene varias líneas
- **Password** (Contraseña):
  - ✓ Como el texto pero lo que se ingrese se verá como ***** en vez de lo que realmente se ingresará
  - ✓ Ojo que el dato viaja como texto plano de todos modos.

Radio Button: Sintaxis HTML


```
<body>

<h2>Eliga un

<input type="radio" name="radio1"
    value="1">Dave Matthews Band. <br>

<input type="radio" name="radio1"
    value="2">The Beatles. <br>

<input type="radio" name="radio1"
    value="c">Tom Jones. <br>

<input type="radio" name="radio1"
    value="d">Tori Amos. <br>

</body>
</html>
```

El tipo del elemento es "radio"

Cada botón radial tiene un valor implícito distinto de los otros.

Todos los botones asociados al mismo grupo deben tener el mismo nombre.

Radio Button: Recuperación del parámetro

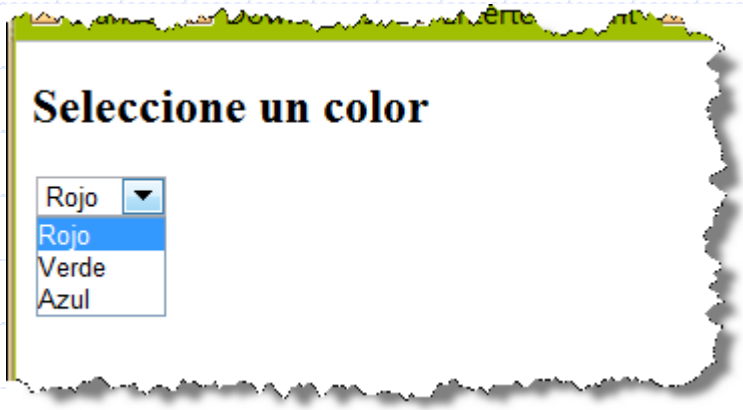
```
<form>
   <input type="radio"
   name="radio1" value="1">Dave
   Matthews Band. <br>

   <input type="radio"
   name="radio1" value="2">The
   Beatles. <br>

   <input type="radio"
   name="radio1" value="c">Tom
   Jones. <br>

   <input type="radio"
   name="radio1" value="d">Tori
   Amos. <br>
</form>
```

```
String alt =
 request.getParameter("radio1");

if (alt.equals("1"))
  out.println("Ud. Eligió Uvas");

else if (alt.equals("2"))
  out.println("Ud. Eligió
   Peras");

else if(alt.equals("c")
  out.println("Ud. Eligió
   Higos");

else
  out.println("Ud. Eligió
   Mangos");
```

Select: Alternativas con pull-down menu

```
<html>
<head>
<meta http-equiv="Content-Type"
    content="text/html; charset=ISO-8859-1">
<title>Select Servlet</title>
</head>
<body>

<form action="SelectServlet">
    <select name="colores" size="1">
            <option value="r">Rojo</option>
            <option value="g">Verde</option>
            <option value="b">Azul</option>
    </select>
</form>

</body>
</html>
```

```
String option =
    request.getParameter("colors");

if (option.equals("r"))
  option = "red";

if (option.equals("g"))
  option = "green";

if (option.equals("b"))
  option = "blue";

out.println("Tu elejiste: " + option);
```



Text Area: Ingreso de texto libre

```
<h2>Ingrese aquí su opinión </h2>
<TEXTAREA NAME="Ta1" ROWS=10
   COLS=40>
   Este texto aparece en el
   campo, pero es editable
</TEXTAREA>
```

```
String texto;
texto =
   request.getParameter("Ta1");
```

Check Box: Parámetros con múltiples valores

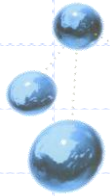
```
¿Qué IDE usa Ud?<BR>

<input type=checkbox name=ide
    value=NB>NetBeans<BR></input>
<input type=checkbox name=ide
    value=EX>Eclipse</input><BR>
<input type=checkbox name=ide
    value=JW>JavaWorkShop</input><BR>
<input type=checkbox name=ide
    value=JP>J++</input><BR>
<input type=checkbox name=ide
    value=CF>Cafe'</input>
```

```
String[] values =
    request.getParameterValues("ide");
for (int i = 0; i < values.length; i++)
     out.println(i+"
    "+values[i]+"<br>");
```


```
1- NB
2- EX
3- JP
```

Nombres de Parámetros

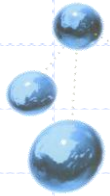
Envío de parámetros



Recuperación

- A veces el programador no puede conocer los nombres de los parámetros (son generados dinámicamente).
  - ✓ Por ejemplo cuando la página ha sido generada como resultado de una consulta a una base de datos

- En este caso podemos usar el método **`getParameterNames()`** que retornará un objeto de tipo **`Enumeration`** conteniendo todos los nombres de los parámetros que vienen en la petición.
  - ✓ `Enumeration en = request.getParameterNames()`



Iterando una enumeración

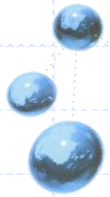
- Para recuperar los valores de los parámetros desconocidos podemos usar los métodos **nextElement()** y **hasMoreElement()**.

- Ejemplo concreto:

```
Enumeration<String> en = request.getParameterNames();
while(en.hasMoreElements()) {
    String parameterName = en.nextElement();
    String parameterValue =
                request.getParameter(parameterName);
    out.println("parametro "+parameterName+
                "tiene valor "+parameterValue);
}
```

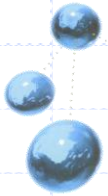
Session Tracking

1. La problemática.
2. Visión de la solución.
3. Ejercicios.
4. Cookies.
5. Soporte para Cookies en Java.



La Problemática

Session Tracking

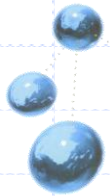


Lea, entienda y opine

```
public class CalculaSession extends HttpServlet {

   int op1, op2;

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

        response.setContentType("text/html);
        PrintWriter out = response.getWriter();

        out.println("<h1> Calcula </h1>");
        op1 = (int)(Math.random()*100)+1;
        op2 = (int)(Math.random()*100)+1;
        out.println("<form method=post>");
        out.println("<h2>"+op1+" + "+op2+" = ");
        out.println("<input type=text size=4 name=resultado>");
        out.println("<input type=submit value=corregir>");
        out.close();
    }
```

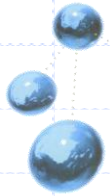
¿Porqué esto no funciona?

```
protected void doPost( request,  response)  throws Exception {

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        String res = request.getParameter("resultado");
        int nres = Integer.parseInt(res);
        out.println("<h2>");
        if (nres == op1+op2)
            out.println("Excelente ");
        else
            out.println(" El resultado era "+(op1+op2));
    }

}
```



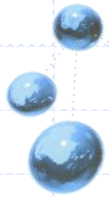
Concurrencia de Serulets

- **op1** y **op2** son variables de instancia del Servlet.
- Cada vez que un usuario visita el Servlet con un **GET** el Servlet genera dos números distintos y reemplaza los antiguos
- Durante el tiempo que un usuario recibe respuesta de **doGet** y va a **doPost** otro usuario puede estar llamado a **doGet** y cambiar los valores de **op1** y **op2**
- El **doPost** va a chequear la respuesta con los últimos valores generados por **doGet** (que pueden no ser los originales)
- La respuesta buena podría ser considerada como mala en este caso porque entre medio se cambiaron los valores
- Para evitar esto el Servlet debe recordar cuales valores generó para que usuario y usar ellos cuando el usuario esté de vuelta
- Esto se hace con la técnica llamada **Session Tracking**.



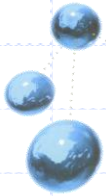
Visión de la solución

Session Tracking



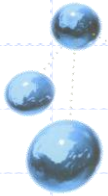
Visión de la solución

- **Session tracking** es un mecanismo que los Servlets pueden usar para mantener información acerca del estado de la conversación con el cliente.
- Una sesión es un dialogo entre una instancia de un browser y el servidor por un cierto período de tiempo.
- Es posible asociar información a una sesión para poder llevar a cabo esto.
- La sesión no es administrada por el Servlet sino por el servidor , particularmente por el Web Container.



Métodos de la clase HttpSession

- HttpSession sesion = request.getSession(true) crea un objeto sesión si no existía antes, si existía lo recupera.
- sesion.isNew() retorna verdadero si el objeto sesión es nuevo
- sesion.putAttribute(String nombre, Object valor) asocia al parámetro nombre el valor valor (value se usa hasta v2.2)
- Object o = sesion.getAttribute("nombre") retorna el objecto asociado al nombre
- sesion.removeAttribute("nombre") borra el par nombre,valor asociado a esa sesión
- Enumeration[]valores = sesion.getAttributeNames()
- String[]valores = sesion.ValueNames() retorna un arreglo/ennumeration de los attributes/values que se han guardado en la sesión
- long l = sesion.getCreationTime() retorna el tiempo (en milisegundos a partir de 1.1.70 0:0:0) de cuando el objeto sesión fue creado
- Long l = sesion.lastAccessedTime() retorna el tiemp en milisegundos de cuando fue accesado por última vez el objeto
- sesion.setMaxInactiveInterval(int seconds) pone (modifica) el timeout de la sesión.



Serulet con sesiones (1/2)

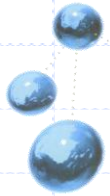
```
public class CalculaSession extends HttpServlet {

    protected void doGet(request,  response)  throws Exception {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession(false);
        out.println("<h1> Calcula </h1>");
        int op1 = (int)(Math.random()*100)+1;
        int op2 = (int)(Math.random()*100)+1;
        session.setAttribute("op1",""+op1);
        session.setAttribute("op2",""+op2);

        out.println("<form method=post>");
        out.println("<h2>"+op1+" + "+op2+" = ");
        out.println("<input type=text size=4 name=resultado>");
        out.println("<input type=submit value=corregir>");
        out.close();
    }
```

Serulet con sesiones (2/2)

```
protected void doPost( request,  response)  throws Exception {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession(true);
        String res = request.getParameter("resultado");
        int nres = Integer.parseInt(res);
        int op1 =
    Integer.parseInt((String)session.getAttribute("op1"));
        int op2 =
    Integer.parseInt((String)session.getAttribute("op2"));
        out.println("<h2>");
        if (nres == op1+op2)
            out.println("Excelente ");
        else
            out.println(" El resultado era "+(op1+op2));
     }
}
```

Ejercicios

Session Tracking



¿ Como acumular productos en varias visitas ?

- Enunciado:
  - ✓ La idea es que en cada visita se vayan seleccionando productos.
  - ✓ Estos se van acumulando para cada usuario.

- Tips:
  - ✓ Se puede usar un objeto de tipo **`Session`** para guardad pares:
    - <codigo de producto><cantidad acumulada>
  - ✓ Después con cada visita se actualiza el estado de estos pares según lo que se seleccionó

Compra de productos: Serulet original

```
public void doGet( .. request, ... response) throws . . .  {
    Hashtable items = Item.getItems();
    . . . .
     . . . .
    out.print("<form action=ProcessPage method='POST'>");
    while(enum.hasMoreElements()) {
        Product e = (Product)enum.nextElement();
        out.print("<TR>");
        out.print("<TD>" + e.number);
        out.print("<TD>" + e.name );
        out.print("<TD>" + e.price+"<TD>");
        out.print("<input type=textarea  SIZE=3 "+
        out.print(" name="+e.number+" value=0 >");
    }
    out.println("</TABLE>");
    out.println("<INPUT TYPE='SUBMIT' VALUE='Process'>");
}
```

Compra de productos: Serulet con session (1/2)

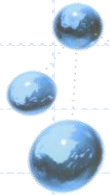
```
public void doGet( .. request, ... response) throws . . .  {
   Hashtable items = Item.getItems();
   HttpSession s = request.getSession(true);
   . . .
   Enumeration en = request.getParameterNames(); int total = 0;
   out.print("<form action=ProcessPayment method='POST'>");
   while(en.hasMoreElements()) {
      String number = (String)en.nextElement();
      String qtty = request.getParameter(number);
      int nqqty = Integer.parceInt(qtty); if (nqqty == 0) continue;
      String qttyAntigua = (String)s.getAttribute(number);
      if (qttyAntigua == null)
         s.setAttribute(codigo,cantidad+"");
      else {
         int qttyNueva = Integer.parseInt(qttyAntigua)+nqtty;
         s.setAttribute(codigo,nuevaCantidad+"");
      }
      Product e = (Product)item.get(number);
      out.print("<TR> <TD>" + e.number+"<TD>" + e.name );
      out.print("<TD>" + e.price+"<TD>"+e.price*nqtty);
   }
   out.println("</TABLE>");
```

Compra de productos: Serulet con session (2/2)

```
         out.println("</TABLE>\n</BODY></HTML>");
        out.println("<br><br><h2> So far you have chosen</h2>"+
                "<TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "<TH>codigo<TH>cantidad<TH> subtotal");
        int total = 0;
        Enumeration e = s.getAttributeNames();
        while(e.hasMoreElements()) {
            String codigo =(String)e.nextElement();
            out.print("<TR><TD>" + codigo + "<TD>");
            String scantidad =s.getAttribute(codigo);
            int ncantidad = Integer.parseInt(scantidad);
            out.println(ncantidad);
            Product e = (Product)item.get(number);
            out.println("<TD>"+(ncantidad*e.price));
            total = total + ncantidad*e.price;
        }
        out.println("</TABLE><br>");
        out.println("<a href='OrderPage'> return to order </a> <br>");
        out.println("<a href='CuentaPage'> make order </a>");
    }
}
```

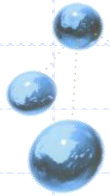
Cookies

Session Tracking



Introducción

- Cookies son otra manera de hacer **session tracking**.
- Por medio de una cookie el Servlet puede enviar información al cliente que la guarda y la devuelve cada vez que el browser contacta al mismo servidor de nuevo.
- Servlets mandan cookies al cliente por medio de la información que va en el Header, así mismo devuelven los clientes esta información al servidor.
- Los clientes devuelven automáticamente las cookies cada vez que contactan al servidor que se las envió (las use o no) en el header.
- Cookies tienen un nombre y un valor (ambos Strings). Adicionalmente pueden guardar un comentario (otro string)
- Un servidor puede mandar más de una cookie.



¿Cómo funciona?

- Para mandar cookies
  - ✓ Instanciar (crear) un objeto Cookie
    - `Cookie c = new Cokie(string, string);`
  - ✓ Mandar la cookie
    - `response.addCookie(c);`

- Recuperando las cookies,
  - ✓ Recuperar toda las cookies
    - `Cookie[] c = request.getCookies();`
  - ✓ Recuperar el nombre y el valor
    - `String name = c[i].getName();`
    - `String value = c[i].getValue();`

Ejemplo

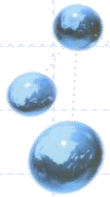
```
void doGet(HttpServletRequest request, HttpServletResponse
   response)
     throws ServletException, IOException {

         response.setContentType("text/html;charset=UTF-8");
         PrintWriter out = response.getWriter();

         out.println("<h1> Agregue una cookie </h1> <h2>");
         out.println("<form method=POST>");
         out.println("Nombre : <input type=text
   name=cnombre>");
         out.println("Valor  : <input type=text name=cvalor>");
         out.println("<br> <input type=submit value=enviar>");
         out.println("</html>");
         out.close();
     }
}
```

Ejemplo

```
protected void doPost( … response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        Cookie[] c = request.getCookies();
        String cn = request.getParameter("cnombre");
        String cv = request.getParameter("cvalor");
        if (cn != null && cv != null) {
            Cookie nuevacookie = new Cookie(cn,cv);
            response.addCookie(nuevacookie);
            out.println("<h1>La cookie con nombre "+cn+
                                    " y valor "+cv+" sera mandada</h1>");
        }
        if (c != null) {
           out.println("<h2> ademas las siguientes cookies fueron recibidas </h2>");
           for (int i = 0; i < c.length; i++)
              out.println("<br>Nombre : "+c[i].getName()+"  valor : "+c[i].getValue());
        }
        out.print("<form><input type=submit value=retornar>");
        out.close();
    }
```

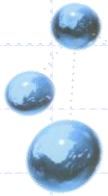
Clase Cookie: Métodos

- `int getMaxAge()`
  - ✓ Retorna la edad máxima de la cookie, especificada en segundos, por defecto. .
  - ✓ El valor "1" indica que la cookie persiste hasta que el browser se cierre.
- `String getName()`
  - ✓ Retorna el nombre de la cookie.
- `String getValue()`
  - ✓ Retorna el valor de la Cookie.
- `void setComment(String comment)`
  - ✓ Especifica un comentario que describe el propósito de la cookie.
- `voidsetMaxAge(int expiry)`
  - ✓ Establece la máxima edad de la cookie en segundos.
- `void setValue(String newValue)`
  - ✓ Le asigna un nuevo valor a la cookie después que la cookie ha sido creada.



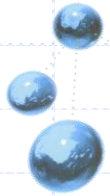
Cookies o sesiones?

- Con sesiones la información es almacenada en el servidor, esto significa que hay un estado que debe ser administrado cuidadosamente.
- Con cookies es el cliente el que tiene la información, lo que significa que la información viaja y vuelve cada vez que el cliente contacta al servidor.
- El cliente puede prohibir el uso de cookies.
- Las sessiones pueden almacenar mucha más (y mejor) información.
- Las sesiones están implementadas con cookies!!!



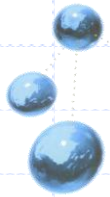
Soporte para Cookies

Session Tracking



Headers (encabezados) de Request y Response

- Provee información de alto nivel desde el client y hacia el cliente
  - ✓ El request permite al servidor obtener características interesantes del cliente.
  - ✓ El response le permite al Servlet definir cómo la información será entregada al browser.
- En general, pueden ayudar a hacer el diálogo con el cliente más efectivo.
- Para el request, hay métodos llamados getXXX o getHeader(XXX) para obtener la información.
- Para el response, hay métodos llamados setHeader(xxx) o setXXX para definir la forma de la información de la respuesta.
- Frecuentemente ambos requieren ser usados en combinación para generar una adecuada respuesta.

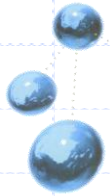


Métodos de la clase HttpRequest

- `getCookies()`:
  - ✓ Permite obtener las cookies que el cliente browser puede haber enviado.
- `getAuthType()`:
  - ✓ Es usado por clientes que tratan de acceder a una página para la cual una contraseña es requerida.
- `getRemoteHost()`:
  - ✓ Utilizado para obtener el nombre de la máquina (hostname) del cliente).
- `getMethod()`:
  - ✓ Retorna el nombre del método utilizado por el browser al enviar la petición (GET, POST, etc.).
- `getProtocol()`:
  - ✓ Versión del protocolo HTTP que el cliente está utilizando.
- `getHeaderNames()`:
  - ✓ Retorna el nombre de todos los encabezados que el cliente ha enviado (es variable: depende de la versión del protocolo HTTP y el Browser).

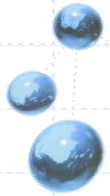
HTTP Request: Métodos getXXX de getHeader("XXX")

- "Accept":
  - ✓ Qué tipos **MIME** entiende el cliente.
- "Accept-Charset":
  - ✓ Cuál set de caracterés está usando el cliente.
- "Accept-Encoding":
  - ✓ Los algoritmos de encoding que está usando el cliente.
- "Accept-Language":
  - ✓ Idiomas (en-us, sp, ge, ..)
- "Authorization":
  - ✓ Para identificar los clientes con una página protegida.
- "Host":
  - ✓ El nombre del computador del cliente.
- "Referer":
  - ✓ La URL de la página que generó el contacto.
- "Cookie":
  - ✓ Para obtener las Cookies.



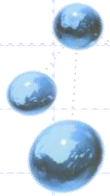
Métodos de la clase Response

- `setContentType(xxx)`:
  - ✓ para informar el tipo MIME de la respuesta.
- `setContentLength(xxx)`:
  - ✓ para informar el largo de la respuesta (usado al transmitir archivos).
- `addCookie(c)`:
  - ✓ para agregar cookies.
- `sendRedirect(xxx)`:
  - ✓ para redirigir el request a otra URL.



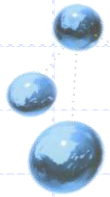
HttpRequest: Algunos getXXX de getHeader("XXX")

- "Content-Type":
  - ✓ Un tipo MIME como "image/gif".
- "Content-Length":
  - ✓ Largo del mensaje (para bytes).
- "Content-Encoding":
  - ✓ Codificación utilizada
- "Content-Language":
  - ✓ Idioma.
- "Cache":
  - ✓ Cómo se debe manejar el cache en el cliente (ej, no-cache, no-store, must-revalidate, max-age=xxxx, etc.)
- "Refresh":
  - ✓ Informa al browser cuán segudo la página debe refrescarse.
- "www-Authenticate":
  - ✓ para administrar páginas protegidas con password.



Serulet Listeners

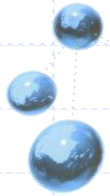
1. Interceptando eventos del ciclo de vida de la aplicación



El contenedor de Serulets: Ciclo de vida

- Cuando una aplicación es desplegada, se instancia un objeto que representa al **Contexto de los Serulets**.

- Esta instancia:
  - ✓ Provee acceso al **Serulet Container**.
  - ✓ Es de tipo `javax.Servlet.ServletContext`.



Notificación y recepción de eventos (1/2)

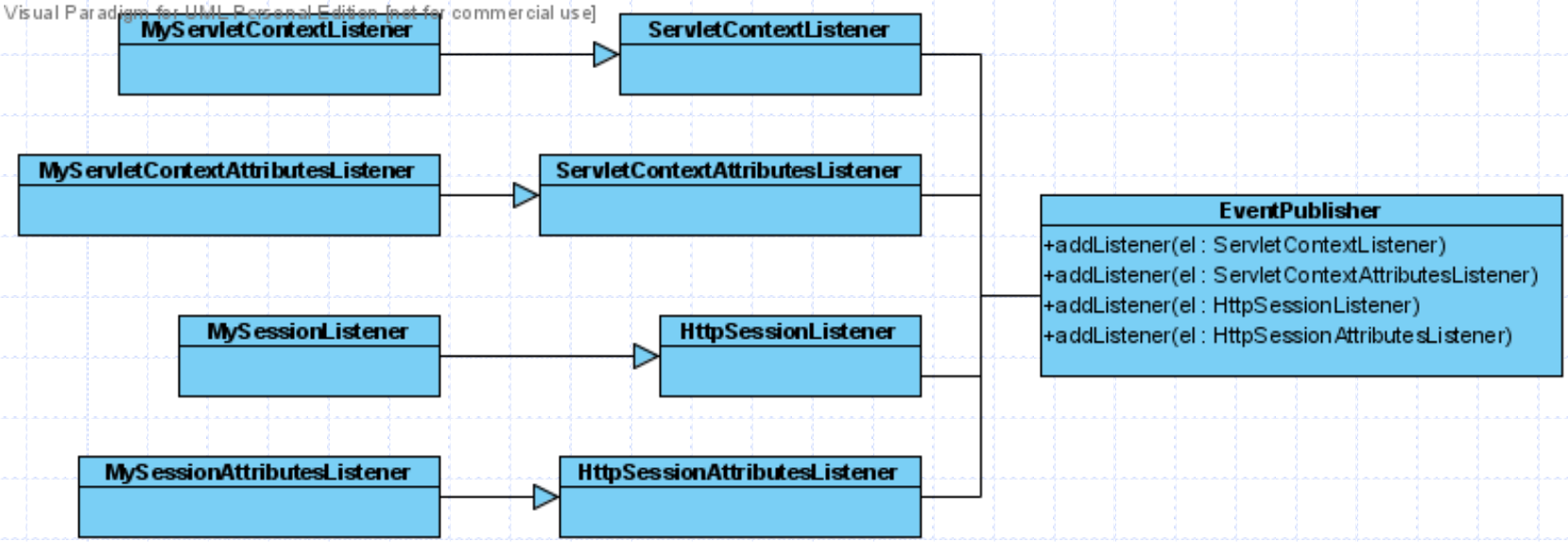
El problema

- En ciertos casos se da la situación de que muchas entidades están interesadas la ocurrencia de un evento particular
- Esto introduce un fuerte acoplamiento entre quien publica la notificación del evento (**Publisher**) y quien se subscribe a la notificación (**Subscriber**).
- Si una nueva entidad necesita recibir el evento, debe realizarse una modificación en el publicador para acomodarse al nuevo subscriptor.

Solución: Patrón Publish-Subscriber

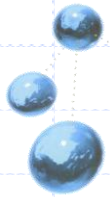
- El publicador provee un método de subscripción a subscriptores que implementen una cierta interfaz.
  - ✓ Se elimina el acoplamiento por el lado del publicador.
  - ✓ Ya no es necesario modificar el código cada vez que hay un nuevo subscriptor.
- La interfaz se compone de métodos que le permite al subscriptor reaccionar ante la ocurrencia de un evento.

Notificación y recepción de eventos (2/2)



Java Beans: Definición básica

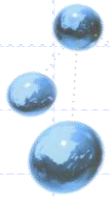
- Es una clase **Java** que posee **atributos**.
- Cada atributo tiene:
  - ✓ Un nombre. Ej. **Edad**
  - ✓ Dos métodos asociados:
    - Un método **getter**: para obtener el valor del atributo
      - `public int getEdad()`
    - Un método setter: para setear el valor del atributo
      - `public void setEdad(int laEdad)`



Eventos del Serulet Context y HttpSession (1/2)

- Un **Event Listener** es una clase Java que:
  - ✓ Sigue el diseño **Java Bean**, y
  - ✓ Provee la interfaz de métodos de notificación.

- Hay dos tipos de eventos y ambos aplican a **`Sevlet Context`** y **`HttpSession`**.
  - ✓ Lifecycle Events.
    - Eventos relativos a la creación o destrucción de los objetos.
  - ✓ Change to Attribute Events.
    - Eventos relativos a la agregación, modificación o eliminación de atributos de un objeto.



Eventos del Serulet Context y HttpSession (2/2)

Serulet Context Events

- Lifecycle.
  - ✓ El contexto del Servlet (de tipo **`ServletContext`**) ha sido creado y está listo para responder peticiones, o está a punto de ser destruido.
  - ✓ Interface del Listener: **`javax.Servlet.ServletContextListener`**.
- Changes to Attributes.
  - ✓ Atributos en el Servlet Context han sido agregados, modificados o eliminados.
  - ✓ Interface del Listener: **`javax.Servlet.ServletContextAttributesListener`**.

Http Session Events

- Lifecycle.
  - ✓ La sesión http (objeto de tipo **`HttpSession`**) ha sido creado o ha sido invalidada o su tiempo de vida a ha expirado (time-out).
  - ✓ Interface del Listener: **`javax.Servlet.http.HttpSessionListener`**.
- Changes to Attributes.
  - ✓ Atributos en el objeto de sesión han sido agregados, modificados o eliminados.
  - ✓ Inteface del Listener: **`javax.Servlet.http.HttpSessionAttributesListener`**.

Definiciones Generales

- Puede haber múltiples **Event Listeners** escuchando los eventos generados por el Contexto de Servlets o por la Sesión.

- Se puede definir el orden en que los distintos tipos de Event Listeners son invocados para un evento.
  - ✓ Se definen 3 tipos de listeners para la sesión: A, B y C.
  - ✓ Se declaran como listeners y se especifica el orden en que los Listeners serán notificados (por tipo): todos los listeners de tipo B primero, luego los A y luego los C.

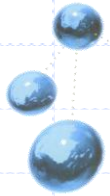
- El Servlet Container:
  - ✓ administra el ciclo de vida de los event Listeners.
  - ✓ Es responsable de crear las instancias antes que la aplicación acepte peticiones.
  - ✓ Debe mantener la referencia de los Event Listeners hasta que la aplicación termine de responder la última petición.

SeruletContext Listeners: Ciclo de uida

- Cuando se despliega una aplicación web, se crea un objeto que representa el contexto de los Servlet asociados a esta aplicación, de tipo **`ServletContext`**, que es asociado a esa aplicación bajo una relación 1 a 1.

- Todos los recursos dentro de una aplicación, tales como Servlets, JSP's, etc. pueden recuperar información almacenada en el **`ServletContext`**.

- Útil particularmente para agregar recursos cuando el contexto es creado y destruirlos cuando el contexto es destruido. Ejemplos:
  - ✓ Crear una conexión a una base de datos y almacenarla en el contexto para que pueda ser accedida por todos los Servlets. Luego, cerrar la conexión a la base al terminar la aplicación.
  - ✓ Abrir archivos de Log para su escritura e iniciar las clases. Al terminar la aplicación se cierra el archivo.

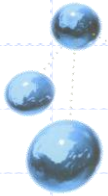


Metodología para crear Serulet Context Listeners

- Crear una clase que herede de la clase **`javax.Servlet.ServletContextListener`**.

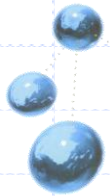
- Sobre-escribir el o los métodos siguientes:
  - ✓ **`void contextDestroyed (ServletContextEvent sce).`**
    - Invocado cuando el Servlet context está a punto de ser destruido.
    - Esto ocurre una sola vez en el ciclo de vida de la aplicación web.
  - ✓ **`void contextInitialized (ServletContextEvent sce).`**
    - Invocado cuando el Servlet Context está listo para atender peticiones.
    - Esto también ocurre una vez en el ciclo de vida de una aplicación web.

- Declarar el nuevo Servlet Listener en el archivo de configuración de la aplicación web (**`web.xml`**) utilizando el elemento **`<listener>`**.



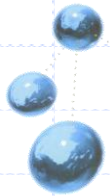
La clase MyContextListener

```
public class MyContextListener implements ServletContextListener {

    private ServletContext context;

    public MyContextListener() {}

  /* Este método se invoca cuando se destruye el ServletContext */
  public void contextDestroyed(ServletContextEvent event)  {
      System.out.println("The Simple Web App. Has Been Removed");
      this.context = null;
    }

  /* Este método se invoca cuando se destruye el ServletContext */
  public void contextInitialized(ServletContextEvent event)  {
      this.context = event.getServletContext();
      System.out.println("The Simple Web App. Is Ready");
    }
  }
```



Declaración del Listener en el web.xml

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

   <web-app>

     ...

     <!-- Define application events listeners -->
     <listener>
       <listener-class>
              com.listeners.MyContextListener
       </listener-class>
     </listener>

     ...

   </web-app>
```

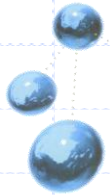


Serulet Context Attribute Listeners: Metodología

- Crear una clase nueva clase que herede (extienda) de la clase **`javax.Servlet.ServletContextAttrubutesListener`**.

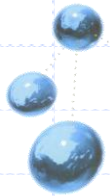
- Sobre escribir alguno de los dos métodos siguientes:
  - ✓ **`void attributeAdded (ServletContextAttributeEvent scab)`**
    - Invocado cuando se agrega un nuevo atributo al Servlet Context.
  - ✓ **`void attributeRemoved (ServletContextAttributeEvent scab)`**
    - Invocado cuando se elimina un atributo del Servlet Context.
  - ✓ **`void attributeReplaced (ServletContextAttributeEvent scab)`**
    - Invocado cuando se ha cambiado el valor de un atributo.

- Declarar el nuevo **Serulet Listener** en el archivo de configuración de la aplicación web (**`web.xml`**) utilizando el elemento **`<listener>`**.



La clase MyServletContextAttributeListener

```
public class ServletContextAttribListener  implements
    ServletContextAttributesListener {

      /* Este método se invoca cuando se destruye el ServletContext */
      public void attributeAdded (ServletContextAttributeEvent scab)  {
        System.out.println("Un atributo fue agregado al ServletContext");
      }

      /* Este método se invoca cuando se destruye el ServletContext */
      public void attributeRemoved (ServletContextAttributeEvent scab)  {
        System.out.println("Un atributo fue eliminado del ServletContext ");
      }

      /* Este método se invoca cuando se destruye el ServletContext */
      public void attributeReplaced (ServletContextAttributeEvent scab)  {
        System.out.println("Un atributo fue modificado en el ServletContext ");
      }
    }
```



Página HTML para modificar atributos

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Jugando con el objeto Servlet Context</title>
</head>
<body>

<h1>Jugando con el objeto ServletContext</h1>

<form name="MyForm" action="./servletcontextattrib" method="POST">
<table width="75%" bgcolor="silver" align="center">
	<tr>
		<td rowspan="3">Elija una acci&oacute;n</td>
		<td><input type="radio" name="action" value="add">Agregar &iacute;tem al Servlet Context</td>
	</tr>

	<tr>
		<td><input type="radio" name="action" value="remove">Eliminar &iacute;tem al Servlet Context</td>
	</tr>

	<tr>
		<td><input type="radio" name="action" value="replace">Reemplazar &iacute;tem al Servlet Context</td>
	</tr>

	<tr>
		<td>Nombre del atributo del Servlet Context</td>
		<td><input type="text" name="name" value=""></td>
	</tr>

	<tr>
		<td>Valor del atributo del Servlet Context</td>
		<td><input type="text" name="value" value=""></td>
	</tr>

	<tr>
		<td colspan="2" align="center"><input type="submit" name="btnSubmit" value="Enviar"></td>
	</tr>
</table>
</form>


</body>
</html>
```

Serulet para modificar atributos en el SeruletContext

```
public class ServletContextAttrib extends HttpServlet {

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

        response.setContentType("text/html");
        ServletOutputStream out = response.getOutputStream();

        out.print("<?xml version='1.0' encoding='UTF-8'?>");
        out.print("<!DOCTYPE html");
        out.print("PUBLIC '-//W3C//DTD XHTML 1.0 Strict//EN'");
        out.print("'DTD/xhtml1-strict.dtd'>");

        out.print("<html>");
        out.print("<head>");
        out.print("<title>ServletContext Attributes</title>");
        out.print("</head>");
        out.print("<body>");

        ServletContext context = getServletContext();

        String action = request.getParameter("action");

        String name = request.getParameter("name");
        String value = request.getParameter("value");

        if (action == null) {}
```

Serulet para modificar atributos en el SeruletContext

```
public class ServletContextAttrib extends HttpServlet {
  ...
     public void service(HttpServletRequest request,HttpServletResponse response)
                throws IOException, ServletException  {
       else {

         if (action.equals("add")){
           String test = (String) context.getAttribute(name);
           if (test == null){
             context.setAttribute(name, value);
             out.print("Added Item To ServletContext object");
           } else {
             context.setAttribute(name, value);
             out.print("Replaced Item in ServletContext");
           }
         }

         else if (action.equals("remove")) {
           String test = (String) context.getAttribute(name);
           if (test == null) {
             out.print("Item does not exist");
           } else {
             context.removeAttribute(name);
             out.print("Removed Item From ServletContext");
           }
         }
```

Serulet para modificar atributos en el SeruletContext

```
public class ServletContextAttrib extends HttpServlet {
      public void service(HttpServletRequest request,HttpServletResponse response)
                throws IOException, ServletException  {

      ...

          else {
            String test = (String) context.getAttribute(name);
            if (test == null) {
              context.setAttribute(name, value);
              out.print("Added Item To ServletContext object");
            } else {
              context.setAttribute(name, value);
              out.print("Replaced Item in ServletContext");
            }
          }
        }

          out.print("<center> <br /> <br />");
          out.print("<a href='./servletcontextattrib.html'>");
          out.print("Back To Home Page");
          out.print("</a>");
          out.print("</center>");

          out.print("</body>");
          out.print("</html>");

      }
    }
```

Declaración del Listener en el web.xml

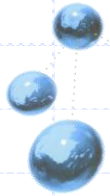
```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
    2.3//EN" "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

    <web-app>
      <listener>
        <listener-class>
          com.listeners.ServletContextAttribListener
        </listener-class>
      </listener>

      <servlet>
        <servlet-name>context attribs</servlet-name>
        <servlet-class>com.servlets.ServletContextAttrib</servlet-class>
      </servlet>

      <!-- Define servlet mappings to urls -->
      <servlet-mapping>
        <servlet-name>context attribs</servlet-name>
        <url-pattern>/servletcontextattrib</url-pattern>
      </servlet-mapping>
      ...
    </web-app>
```

HttpSession Listeners: Metodología

- Los listeners de eventos de aplicaciones notifican también cuando un objeto **`HttpSession`** es creado, destruido, o alguno de sus atributos ha sido modificado.

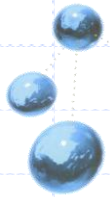
- Para crear un listener que escuche eventos relacionados con el ciclo de vida del objeto **`HttpSession`**, se debe implementar la interfaz **`javax.Servlet.http.HttpSessionListener`**. Esta clase posee los siguientes métodos:
  - ✓ `void sessionCreated (HttpSessionEvent se)`
    - Notificación de que una sesión ha sido creada.
  - ✓ `void sessionDestroyed (HttpSessionEvent se)`
    - Notificación de que una sesión ha sido invalidada.

- Para crear un listener que escuche eventos relacionados con la creación, modificación o eliminación de atributos en el objeto HttpSession, se debe implementar la interfaz javax.Servlet.http.HttpSessionAttributesListener . Esta interfaz posee los siguientes métodos:
  - ✓ `void attributeAdded (HttpSessionBindingEvent se)`
    - Notification that an attribute has been added to a session.
  - ✓ `void attributeRemoved (HttpSessionBindingEvent se)`
    - Notification that an attribute has been removed from a session.
  - ✓ `void attributeReplaced (HttpSessionBindingEvent se)`
    - Notification that an attribute has been replaced in a session.



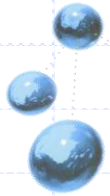
Serulet Filters

1. Interceptando peticiones y respuestas



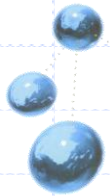
Conceptos generales

- Un filtro te da acceso a los objetos **`HttpServletRequest`** y **`HttpServletResponse`** antes y después, respectivamente, de que lleguen a un recurso web.
- Los filtros se declaran en el **`web.xml`** con el elemento xml **`<filter>`** y su uso se indica mediante **`<filter-mapping>`**.
- Se puede asociar un filtro a un patrón de url, al igual que los otros Servlets.
- Además los filtros se pueden encadenar, es decir, hacer que un filtro se ejecute después de otro.
- Definidos por las interfaces: **`Filter`**, **`FilterConfig`** y **`FilterChain`**.



Ejemplo

```
public class UpperCaseFilter implements Filter {
  private FilterConfig filterConfig = null;

  public void destroy() {

  }

   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {
    System.out.println("Filter");
    Enumeration enum = request.getAttributeNames();
    while (enum.hasMoreElements()) {
      String attributeName = (String) enum.nextElement();
      String attributeValue = (String) request.getAttribute(attributeName);
      request.setAttribute(attributeName, attributeValue.toUpperCase());
    }
    chain.doFilter(request, response);
  }

  public void init(FilterConfig filterConfig) throws ServletException {
    System.out.println("Filter initialized");
    this.filterConfig = filterConfig;
  }
}
```

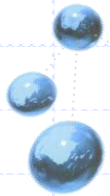


Declaración del Filter en el web.xml

```
<web-app>
  <filter>
    <filter-name>UpperCase Filter</filter-name>
    <filter-class>UpperCaseFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>UpperCaseFilter</filter-name>
    <url-pattern>*</url-pattern>
  </filter-mapping>

  <servlet>
    <servlet-name>DoublyFilteredServlet</servlet-name>
    <servlet-class>DoublyFilteredServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>DoublyFilteredServlet</servlet-name>
    <url-pattern>/servlet/DoublyFilteredServlet</url-pattern>
  </servlet-mapping>
   </web-app>
```

Se declara un filtro igual que un Serulet.

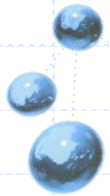
Se define el mapping del Filtro a un patrón de URL's



Declaración del Filter en el web.xml

```
<web-app>
      <filter>
    <filter-name>UpperCase Filter</filter-name>
    <filter-class>UpperCaseFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>UpperCaseFilter</filter-name>
    <servlet-name>DoublyFilteredServlet</servlet-name>
  </filter-mapping>

  <servlet>
    <servlet-name>DoublyFilteredServlet</servlet-name>
    <servlet-class>DoublyFilteredServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>DoublyFilteredServlet</servlet-name>
    <url-pattern>/servlet/DoublyFilteredServlet</url-pattern>
  </servlet-mapping>
    </web-app>
```

Se puede hacer el mapping a un Serulet, de manera que el filtro actúa sobre todas las peticiones filtradas para ese Serulet.



Consultas?