Sistemas de Información

Tecnologías Web. Interactividad y envío de información Cliente → Servidor

Servlets

Agradecimientos: Jesus Villamor Lugo, Simon Pickin de IT/UCIIIM.

mcfp@it.uc3m.es

Concepto

¿Qué es un servlet?

- Son componentes de una aplicación web que se ejecutan en el servidor
  - Permiten extender la funcionalidad del servidor (tanto de servidores HTTP como de otro tipo de servidores como por ejemplo ftp)
  - Son una alternativa a los CGI
  - Cada petición se ejecuta en un hilo. Los Servlets quedan residentes en memoria cuando la petición termina
- Un servlet es un pequeño código Java que el servidor Web carga para manejar peticiones del cliente
  - Estas clases java utilizan **el API Servlet**
  - Se cargan y ejecutan dentro de un servicio de red (ej. un Servidor Web)
  - Implementan determinados interfaces que le permiten:
    - Recibir una petición HTTP
    - Generar una respuesta
- Servlets son a Servidores como Applets son a Navegadores
  - Con la diferencia de que los Servlets no suelen usar GUI

Servlets

Un poco de historia

- JavaSoft lanza el Java Web Server
  - A mediados de 1997, conocido como *Jeeves*
  - Dos objetivos
    - Implementación de un **Servidor de Internet basado en Java**
    - Introducir los **Servlets**
  - Se provee el Java Servlet Development Kit (JSDK)
    - Permite ejecutar servlets en los servidores HTTP más extendidos
      - Apache, Netscape Enterprise Server, Microsoft IIS.
- Ahora forma parte de la especificación J2EE
  - J2EE 1.3 (principios de 2002) incluye servlet 2.3
  - J2EE 1.4 (mediados de 2004) incluye servlet 2.4

Servlets

Documentación e implementaciones

- **Documentación**
  - http://java.sun.com/products/servlet/docs.html
  - http://java.sun.com/products/servlet/2.3/javadoc/index.html
- **J2EE SDK**
  - Implementación de J2EE provisto por SUN
  - Incluye el *Sun Java System Application Server Platform*
    - últimas versiones basadas en Apache Tomcat
- **Apache Jakarta Tomcat**
  - versión 4: implementación software libre de servlet 2.3
  - versión 5: implementación software libre de servlet 2.4

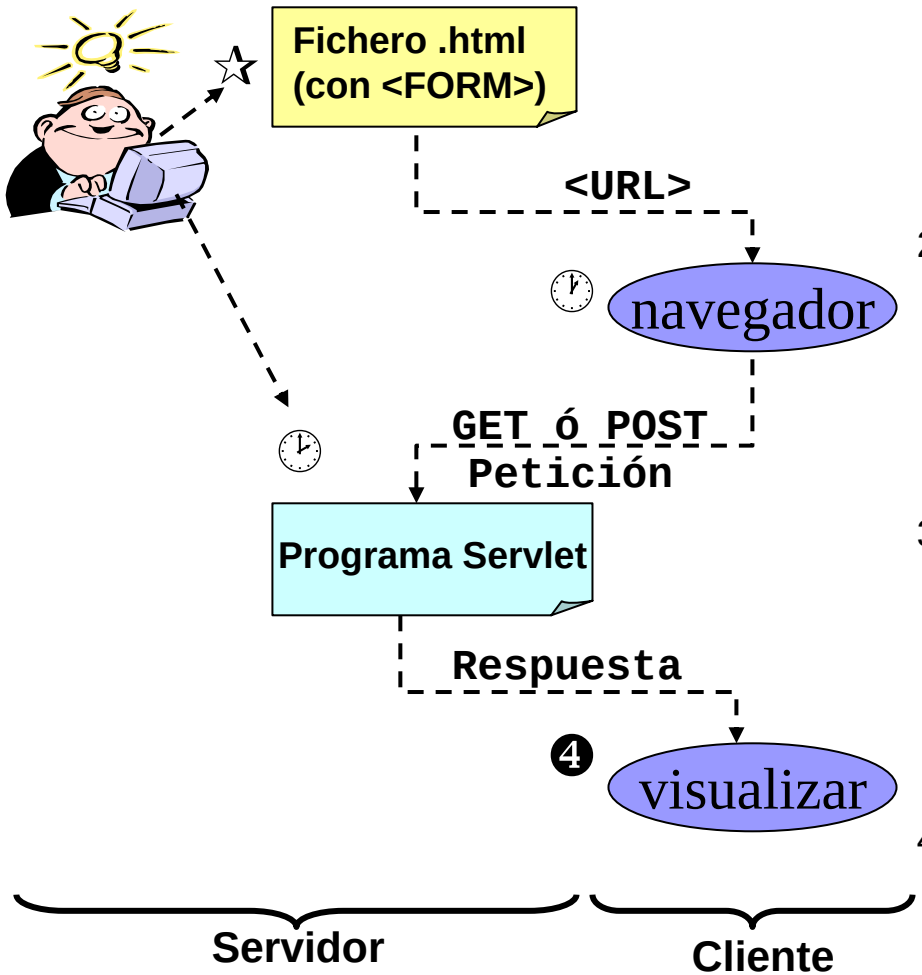
Utilidad

¿Para qué sirve un servlet?

- Extiende funcionalidad del servidor
- Sus usos
  - Reemplazar a los “primitivos” CGIs
    - Ej. Procesamiento de formularios en la parte servidor
  - Colaboración entre personas
    - Ej. Conferencias en-línea
      - Debido a la concurrencia y a la sincronización de peticiones
  - Reenvío de peticiones a otros servidores y servlets

Servlets

¿Cómo funcionan?



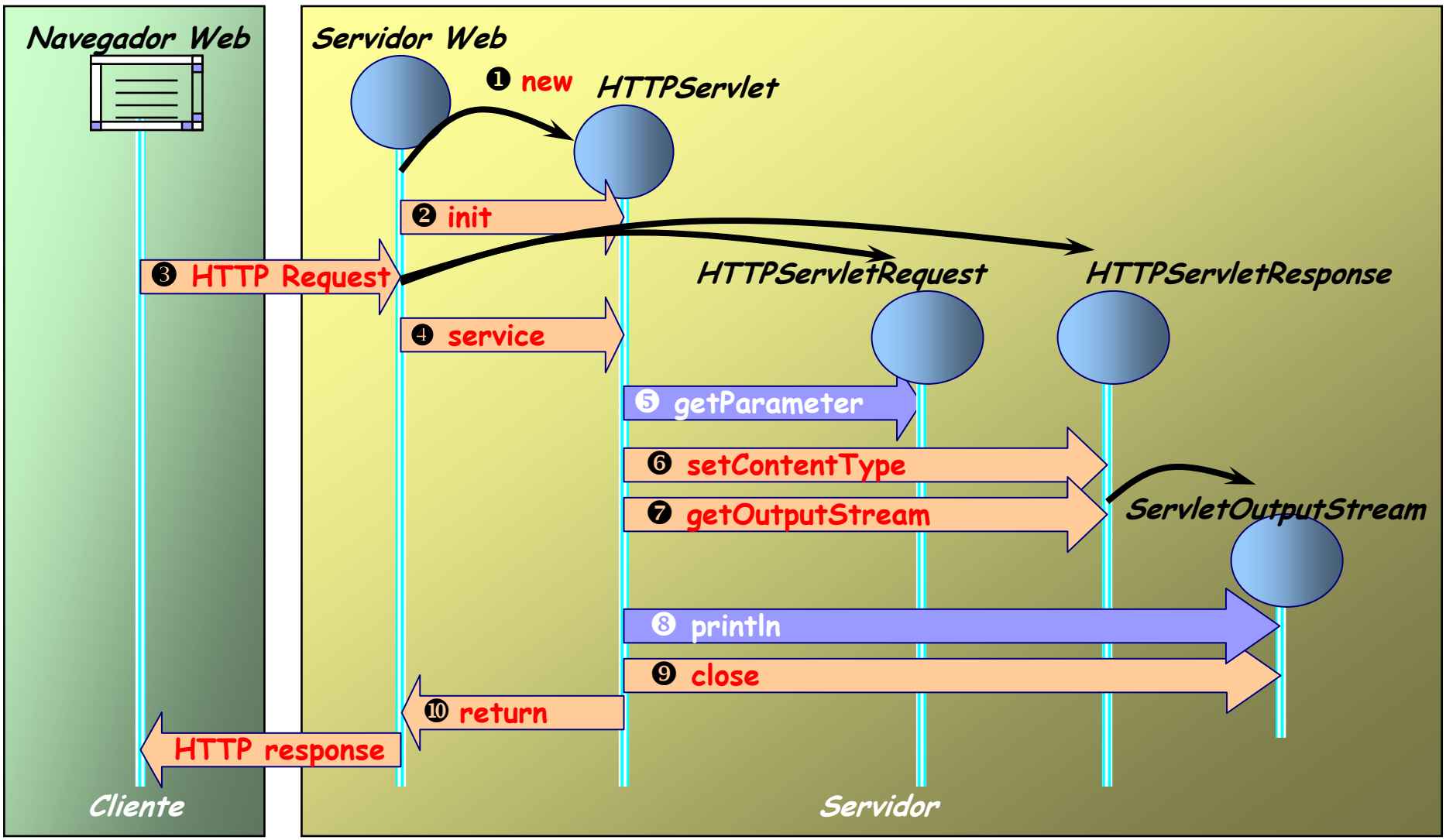
1. Se parte de un Fichero.html **que tenga un formulario** (<FORM>) accesible a través de Internet
   - El <FORM> deberá especificar las diversas formas de paso de parámetros
2. El cliente accederá al fichero html a través de un Navegador, rellenará el formulario y dará a la tecla de aceptar (Submit). Empaquetando la petición.
3. La petición de cliente es atendida por un Programa Servlet (escrito en Java) quien da la debida respuesta tras procesarse la petición.
4. El cliente Visualizará la respuesta

Servlets

Ciclo de vida de un servlet

1. Instanciación e inicialización del servlet. Si aún no existe ninguna instancia del servlet (es la primera llamada), el contenedor web:
   - Primero **carga la clase** del servlet,
   - luego **crea una instancia** y la inicializa invocando su método **init**
2. Entonces, el servidor puede servir continuamente peticiones. Para cada llamada:
   - El contenedor **crea un nuevo thread** (hilo)
   - Se invoca al método **service** de dicho thread.
   - Los servlets residen típicamente en servidores multithread pero el programador es el responsable de sincronizar el acceso a los recursos compartidos.
   - También se puede restringir el acceso a un único thread
3. El método service **determina tipo de petición** que ha llegado y llama al método correspondiente **doGet**, **doPost**, etc.
4. Un servlet se desactiva cuando recibe una llamada al método **destroy** o cuando su servidor se muere

Escenario básico



Aplicación. ¿Qué necesito?

Para desplegar servlet creado por otro

1. Crear página web (o mecanismo alternativo) que realice petición al servlet
2. Instalar motor de servlets que proporcione:
    1. Contenedor donde instalar el servlet
    2. Soporte en tiempo de ejecución (mecanismos de arranque y parada del servidor, logs, etc.)

¿Cómo referenciar un Servlet desde una página web?

- Usando **etiquetas HTML**
  - Usando la etiqueta **A**:

    Texto
  - Usando la etiqueta IMG:

    <**img** src=" direccion_del_servlet ">
  - Otras (javascript, css, etc.)
- Usando un **formulario** (forma + habitual):

  <form action=" direccion_del_servlet " > <!-- Elementos del formulario --> </form>

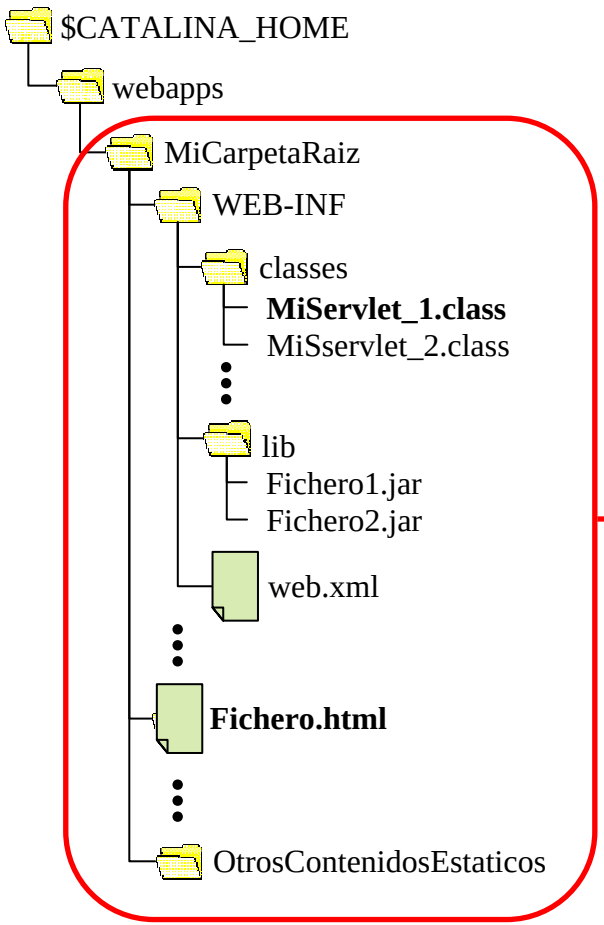
Despliegue de un servlet

Instalación y conf. de un motor de Servlets

- Instalación (ejemplo Tomcat)
- Configuración
  - Dar valor variables de entorno
    - CATALINA_HOME (dir de instalación de Tomcat)
    - JAVA_HOME (dir de instalación del JDK)
  - Añadir al path (variables anteriores/bin)
  - Añadir al classpath
    - ${CATALINA_HOME}/common/lib/servlet.jar
    - Ubicación de las clases de nuestra aplicación

Despliegue de un servlet

Contenedor: ¿Dónde coloco los servlets?



web.xml

```
(...)
<servlet>
      <servlet-name>NombreDelServlet</servlet-name>
      <description>
        Servlet que devuelve un documento XHTML con
        el mensaje "¡Hola Mundo!".
      </description>
      <servlet-class>MiServlet_1</servlet-class>
</servlet>
(...)
```

Fichero.war

http://localhost:8080/MiCarpetaRaíz/servlet/MiServlet_1

Despliegue de un servlet

Soporte en tiempo de ejecución

- Motor de servlets (Apache-Tomcat)
  - Arranque: `startup.sh`
  - Parada: `shutdown.sh`
- Funcionalidad
  - Soporte para servicios de red (evita al programador preocuparse de detalles de comunicación entre el contenedor y el servidor web)
  - Inicializa, invoca y gestiona ciclo de vida del servlet
  - Proporciona una implementación del API de servlets de java

Aplicación ¿Qué necesito?

Para crear mi propio servlet

- JDK (Entorno para compilación, ejecución y depuración)
- Java Servlet API. APIs específicas para el desarrollo de servlets
- Entorno de pruebas (consultar requisitos de despliegue)

Aplicación. Desarrollo

Tareas de un servlet

1. Leer datos enviados por el usuario.
   - Procedentes de: formulario HTML, un applet o una aplicación cliente HTTP, etc.
2. Recuperar información del usuario incluida en petición HTTP
   - Ej: Nombre de la máquina cliente, tipo de navegador,cookies, etc.
3. Generar resultados
   - Para ello puede necesitar acceder a BD, o llamar otros programas locales o remotos
4. Formatear los resultados en un documento
   - Típicamente una página web
5. Poner parámetros de respuesta HTTP
   - Content-type, cookies, etc.
6. Devolver el documento al cliente
   - Documento HTML, zip, gif, etc.

Aplicación. Desarrollo

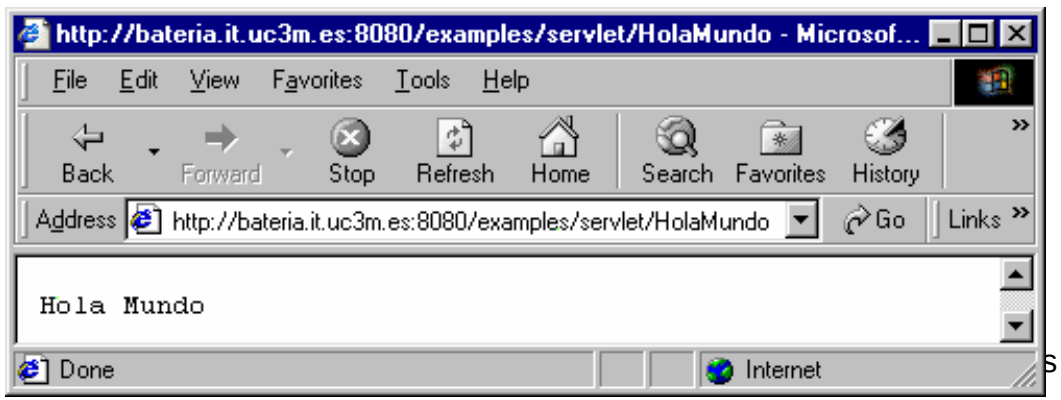
Esqueleto básico

- Todo Servlet tiene que extender la clase **HttpServlet** y sobrescribir el método **doGet** ó **doPost**, dependiendo de si los datos se envían con GET o con POST

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

public class MiServlet extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {

  // Se usa "request" para leer cabeceras HTTP entrantes (e.j. cookies)
  // y datos de formularios HTML (e.j. Datos que el usuario proporciona en el “submit”)

  // Se usa "response" para especificar la línea de respuesta HTTP y las cabeceras
  // (e.j. Especificar el tipo de contenido, poner las cookies).

  PrintWriter out = response.getWriter();
  // Se usa "out" para enviar el contenido al navegador del cliente
  }
}
```

Ejemplo 1 (Sin <FORM>)
El ejemplo más básico

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
  throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```

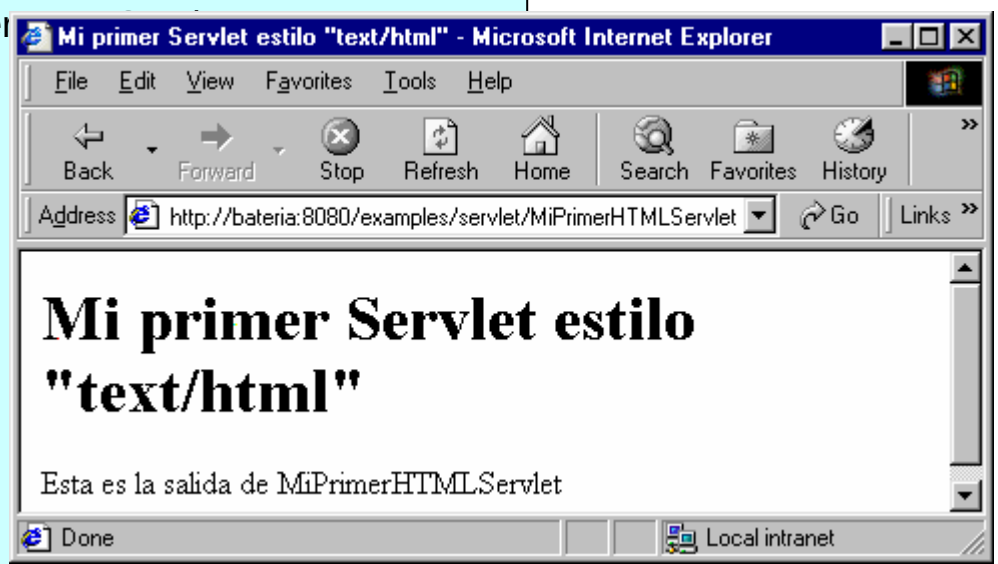


Recordatorios:

1. Deje completamente definido en su servidor http (Apache, MS IIS,...) **dónde se van a dejar sus servlets** (conviene que sea un sitio estándar para todo el mundo).
2. Incluya en el **CLASSPATH** un enlace a servlet.jar y al sitio donde deje sus servlets.
3. Se invocará al Servlet de la forma **http://host/any-path/MiServlet** (siendo MiServlet el nombre de la clase)

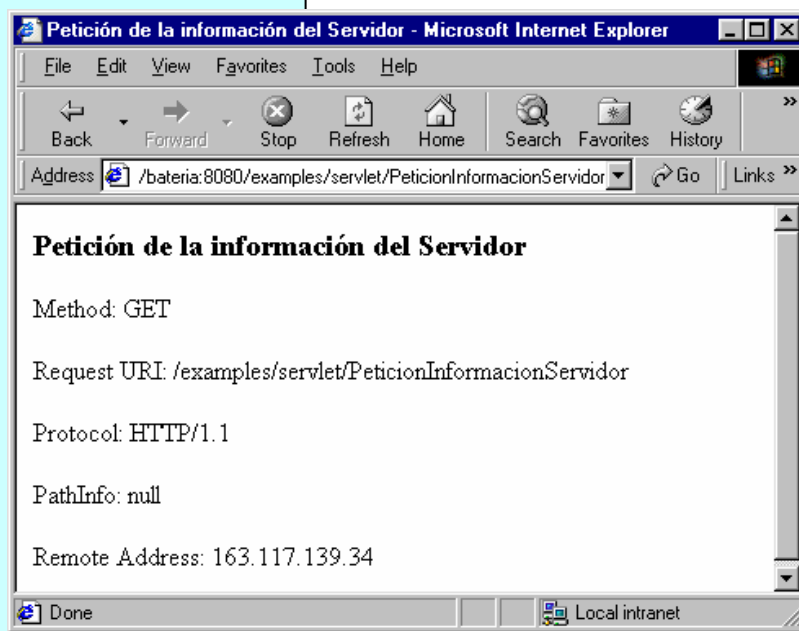
Ejemplo 2 (Sin <FORM>)
Una respuesta “text/html”

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MiPrimerHTMLServlet extends HttpServlet {
public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
  PrintWriter out;
  String titulo = "Mi primer Servlet estilo \"text/html\"";
  String contenido= “Esta es la salida de MiPrimer
  response.setContentType("text/html");
  out = response.getWriter();
  out.println("<HTML><HEAD><TITLE>");
  out.println(titulo);
  out.println("</TITLE></HEAD><BODY>");
  out.println("<H1>" + titulo + "</H1>");
  out.println("<P>“+contenido);
  out.println("</BODY></HTML>");
  out.close();
  }
}
```



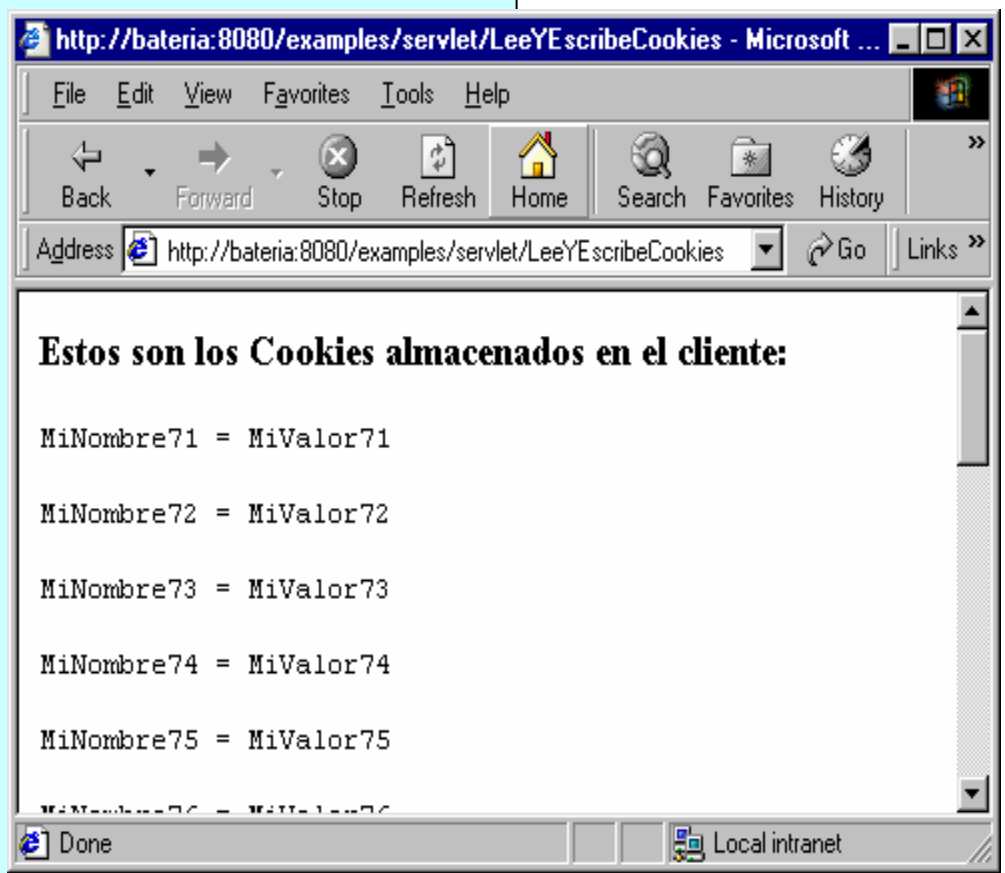
Ejemplo 3 (Sin <FORM>)
Información del Servidor

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MiPrimerHTMLServlet extends HttpServlet {
public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
  PrintWriter out;
  String titulo = "Petici&oacute;n de la informaci&oacute;n del Servidor";
  response.setContentType("text/html");
  out = response.getWriter();
  out.println("<HTML><HEAD><TITLE>");
  out.println(titulo);
  out.println("</TITLE></HEAD><BODY>");
  out.println("<H3>" + titulo + "</H3>");
  out.println("<P>Method: " + request.getMethod());
  out.println("<P>Request URI: " + request.getRequestURI());
  out.println("<P>Protocol: " + request.getProtocol());
  out.println("<P>PathInfo: " + request.getPathInfo());
  out.println("<P>Remote Address: " +
                              request. getRemoteAddr());
  out.println("</BODY></HTML>");
  out.close();
  }
}
```



Ejemplo 4 (Sin <FORM>)

```
import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
public class LeeYEscribeCookies extends HttpServlet {
    static int contador;
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Imprime los cookies
        Cookie[] cookies = request.getCookies();
        out.println("<H3>Estos son los Cookies
                almacenados en el cliente:</H3>");
        out.println("<PRE>");
        for (int i = 0; i < cookies.length; i++) {
            Cookie c = cookies[i];
            String name = c.getName();
            String value = c.getValue();
            out.println(name + " = " + value+"\n");
        }
        // Añade un cookie
        Cookie nuevo_c= new Cookie("MiNombre"+
                contador, "MiValor"+contador);
        contador++;
        response.addCookie(nuevo_c);
    }
}
```



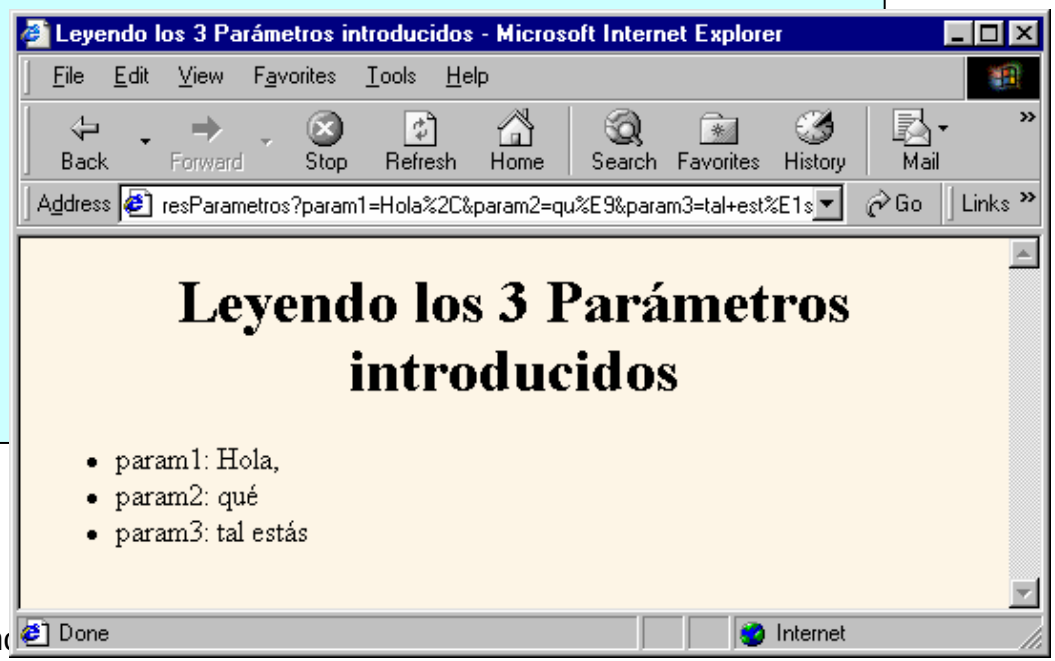
Ejemplo 1 (Con <FORM>)
Leer y Escribir parámetros (1/2)

```
<!-- Fichero TresParametros.html -->
<HTML>
<HEAD>
  <TITLE>Recoge Tres par&aacute;metros</TITLE>
</HEAD>
<BODY BGCOLOR=#FDF5E6>
<H1 ALIGN="CENTER"> Recoge Tres par&aacute;metros </H1>
<FORM ACTION="examples/servlet/TresParametros">
  Primer Par&aacute;metro: <INPUT TYPE="TEXT" NAME="param1"><BR>
  Segundo Par&aacute;metro: <INPUT TYPE="TEXT" NAME="param2"><BR>
  Tercer Par&aacute;metro: <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT"
           VALUE="ENVIA con GET">
  </CENTER>
</FORM>
</BODY>
</HTML>
```

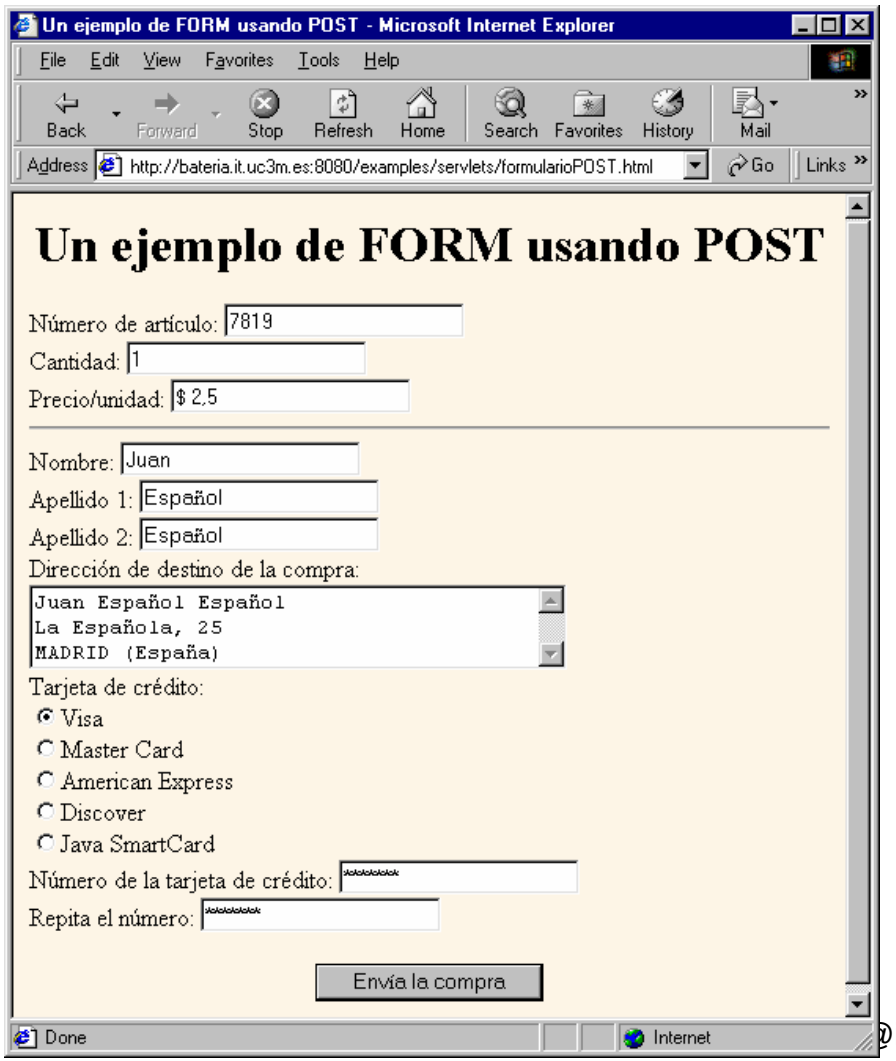
Ejemplo 1 (Con <FORM>)

Leer y Escribir parámetros (2/2)

```
import java.io.*; import javax.servlet.*;   import javax.servlet.http.*;   import java.util.*;
public class TresParametros extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Leyendo los 3 Par&aacute;metros introducidos";
    out.println("<HTML><HEAD><TITLE>"+title+"</TITLE></HEAD>"+
      "<BODY  BGCOLOR=#FDF5E6>\n" +"<H1 ALIGN=CENTER>" + title + "</H1>\n" + "<UL>\n" +
      "  <LI>param1: " +
      request.getParameter("param1") + "\n" +
      "  <LI>param2: " +
      request.getParameter("param2") + "\n" +
      "  <LI>param3: " +
      request.getParameter("param3") + "\n" +
      "</UL>\n" + "</BODY></HTML>");
    }
}
```



Ejemplo 2 (Con <FORM>)
Un formulario usando POST (1/4)



NÓTESE:

1. El formulario está compuesto por:
   - **3 textfields** (para el artículo)
   - **3 textfields** (para el usuario)
   - **1 textarea** (para la dirección)
   - **5 radiobuttons** (para el tipo de tarjeta)
   - **2 textfields** tipo password (para el número de la tarjeta de crédito)
2. El formulario va a enviar los datos por el método **POST**
   - Pueden ser muchos datos para enviarlos por la línea de petición

En la próxima diapositiva se muestra el código HTML resultante

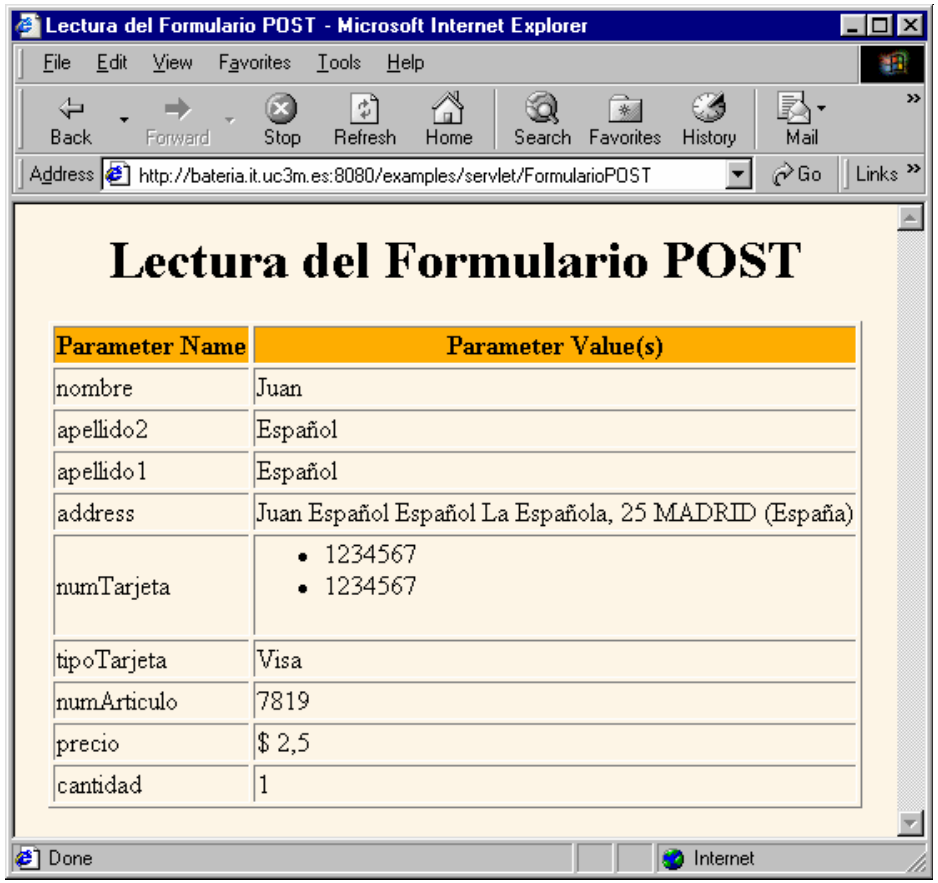
Ejemplo 2 (Con <FORM>)
Un formulario usando POST (2/4)

```
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Un ejemplo de FORM
usando POST</H1>
<FORM ACTION="/examples/servlet/FormularioPOST"
      METHOD="POST">
  N&uacute;mero de art&iacute;culo:
  <INPUT TYPE="TEXT" NAME="numArticulo"><BR>
  Cantidad:
  <INPUT TYPE="TEXT" NAME="cantidad"><BR>
  Precio/unidad:
  <INPUT TYPE="TEXT" NAME="precio" VALUE="$">
  <BR>
  <HR>
  Nombre:
  <INPUT TYPE="TEXT" NAME="nombre"><BR>
  Apellido 1:
  <INPUT TYPE="TEXT" NAME="apellido1"><BR>
  Apellido 2:
  <INPUT TYPE="TEXT" NAME="apellido2"><BR>
Direcci&oacute;n de destino de la compra:
<TEXTAREA NAME="address" ROWS=3 COLS=40>
   </TEXTAREA><BR>
  Tarjeta de crédito:<BR>
```

```
    <INPUT TYPE="RADIO" NAME="tipoTarjeta"
        VALUE="Visa">Visa<BR>
    <INPUT TYPE="RADIO" NAME="tipoTarjeta"
        VALUE="Master Card">Master Card<BR>
    <INPUT TYPE="RADIO" NAME="tipoTarjeta"
         VALUE="Amex">American Express<BR>
    <INPUT TYPE="RADIO" NAME="tipoTarjeta"
         VALUE="Discover">Discover<BR>
    <INPUT TYPE="RADIO" NAME="tipoTarjeta"
          VALUE="Java SmartCard">Java SmartCard
 <BR> N&uacute;mero de la tarjeta de crédito:
 <BR> <INPUT TYPE="PASSWORD"
  NAME="numTarjeta">
  Repita el n&uacute;mero:
  <INPUT TYPE="PASSWORD"
  NAME="numTarjeta">
 <BR><BR> <CENTER>
    <INPUT TYPE="SUBMIT"
    VALUE="Env&iacute;a la compra">
  </CENTER>
</FORM>
</BODY>
</HTML>
```

Ejemplo 2 (Con <FORM>)

Un formulario usando POST (3/4)



NÓTESE:

1. El formulario tiene **carácter genérico**
   - Se imprime en una tabla los nombres y los valores de los parámetros que ha definido la página HTML
2. **Los campos del formulario** se transmiten tal y cómo los ha tecleado el usuario
3. **Los valores de los campos** aparecen de la forma nombre-valor
   - El orden puede variar: se recomienda que el nombre del campo sea significativo

En la próxima diapositiva se muestra el código JAVA resultante

Ejemplo 2 (Con <FORM>)
Un formulario usando POST (4/4)

```
import java.io.*; import javax.servlet.*; import javax.servlet.http.*; import java.util.*;
public class FormularioPOST extends HttpServlet {
  public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String titulo = "Lectura del Formulario POST";
    out.println("<HTML><HEAD><TITLE>"+titulo+"</TITLE></HEAD>"+ "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + titulo + "</H1>\n" + " <TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +  "<TH>Parameter Name<TH>Parameter Value(s)");
    Enumeration paramNames = request.getParameterNames();
    while(paramNames.hasMoreElements()) {
      String paramName = (String) paramNames.nextElement();
      out.println("<TR><TD>" + paramName + "</TD>\n<TD>");
      String[] paramValues = request.getParameterValues(paramName);
      if (paramValues.length == 1)  {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0) out.print("<I>No Value</I>"); else out.print(paramValue);
      } else {
        out.println("<UL>");
        for(int i=0; i<paramValues.length; i++)  { out.println("<LI>" + paramValues[i] + "</LI>"); }
        out.println("</UL>");
      }
      out.println("</TD></TR>");
    } //while
    out.println("</TABLE>\n</BODY></HTML>");
  }
  public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  } }
```

Servlets vs CGIs

- Eficiencia
  - N **threads** vs N **procesos**
    - Los Servlets **guardan el estado** entre peticiones
- Conveniencia
  - Todo puede hacerse en **Java** (**homogeneidad de lenguaje**)
    - Formularios, lectura y escritura de cabeceras HTTP, manejo de cookies, seguimiento de sesiones,...
- Potencia
  - Contiene **bibliotecas** Java que facilitan las acciones en el lado Servidor
    - Comunicación con el Servidor Web, manejo de URLs, comunicación entre Servlets y con bases de datos.
- Portabilidad
  - El código **no necesita cambiarse**: responde al estándar J2EE
    - Corren en MS IIS, Apache Tomcat, Sun JSWDK, BEA WebLogic, IBM WebSphere, ...
- Seguridad
  - Ejecución en la **JVM** vs **Sistema Operativo**
    - No es necesario filtrar caracteres especiales, chequear los límites del string (o array),...
- Coste
  - Poco coste en **dinero** y **esfuerzo**
    - El **JSDK** es gratuito (hoy por hoy) y sin problemas de configuración

Servlets
Algunas referencias

– ***Dan Harkey, Robert Orfali***, ***Client/Server Programming with Java and CORBA, 2nd Edition*** *2nd. Edition (1998) John Wiley & Sons, Inc. ISBN: 0-471-24578-X*
Mirarse el capítulo 12

– ***Marty Hall,*** ***CORE Servlets and JavaServer Pages*** *(May 26, 2000) Prentice Hall PTR; ISBN: 0130893404*

La mejor referencia

(disponible en PDF de:

http://pdf.coreservlets.com/)