

# Tema 5: Servlets y JSP

---

## 1. Servlets.

1. Introducción.
2. Objeto Request.
3. Objeto Response.
4. Cookies.
5. Sesiones.

## 2. JSP.

1. Introducción.
2. Elementos JSP.
3. Java Beans.
4. Etiquetas personalizadas.
5. JDBC
6. Integración Servlets y JSP

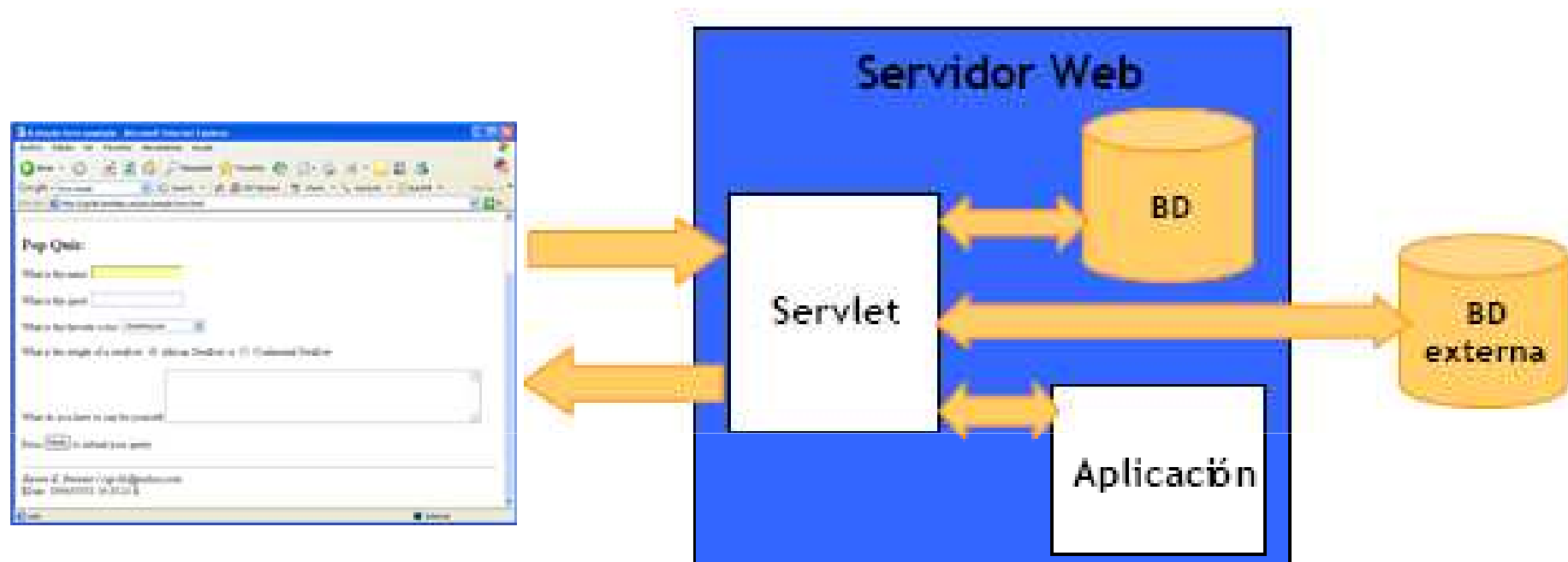
# ¿Qué son? (I)

---

- Los Servlets son la alternativa Java a los CGIs.
- Actúan como capa intermedia entre:
  - Petición proveniente de un Navegador Web u otro cliente HTTP
  - Bases de Datos o Aplicaciones en el servidor HTTP
- Son aplicaciones Java especiales, que extienden la funcionalidad del servidor HTTP, dedicadas a:
  - Leer los datos enviados por el cliente.
  - Extraer cualquier información útil incluida en la cabecera HTTP o en el cuerpo del mensaje de petición enviado por el cliente.
  - Generar dinámicamente resultados.
  - Formatear los resultados en un documento HTML.
  - Establecer los parámetros HTTP adecuados incluidos en la cabecera de la respuesta (por ejemplo: el tipo de documento, cookies, etc.)
  - Enviar el documento final al cliente.

## SERVLETS: 1.1 Introducción

### ¿Qué son? (II)



### ¿Qué son? (III)

---

- Los objetos servlets cumplen los siguientes requisitos:
  - Utilizan el “Servlet Application Programming Interface” (SAPI)
    - El interfaz SAPI define una manera estándar para que las peticiones HTTP sean procesadas por esta clase Java (independiente del servidor).
- Asociados a la URL de la petición, son manejados por el contenedor de servlets con una arquitectura simple.
  - El contenedor provee el entorno de ejecución para todos los servlets basados en los anteriores requisitos.
- Disponibles para la gran mayoría de servidores web.
- Son independientes de la plataforma y del servidor.

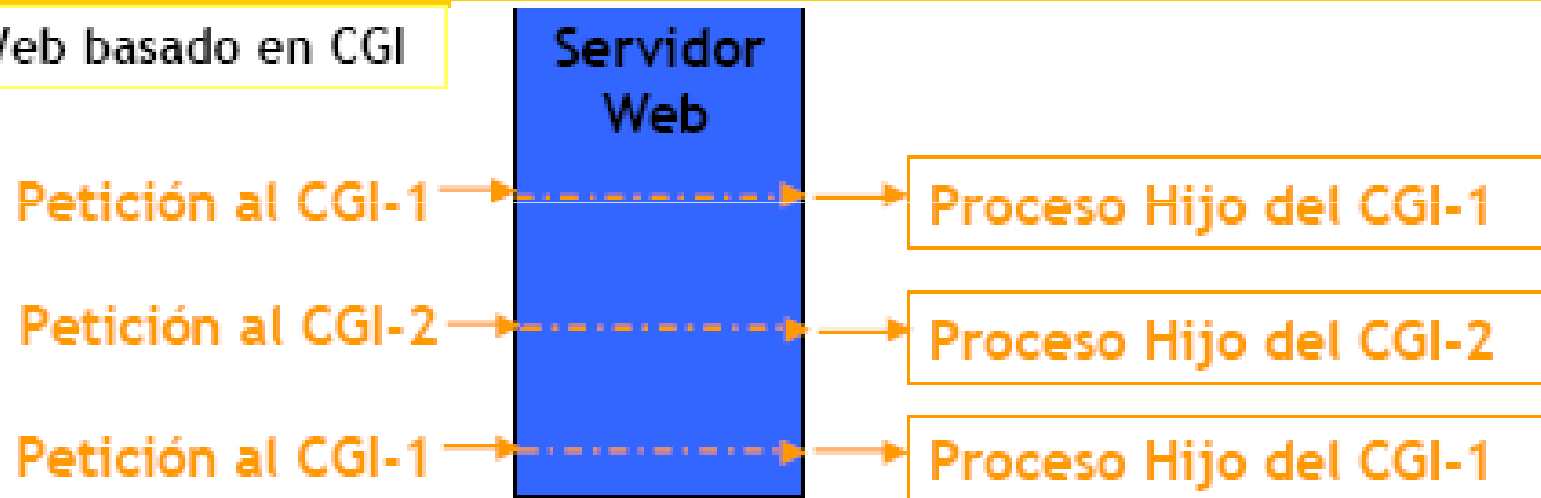
## Ventajas de los Servlets

---

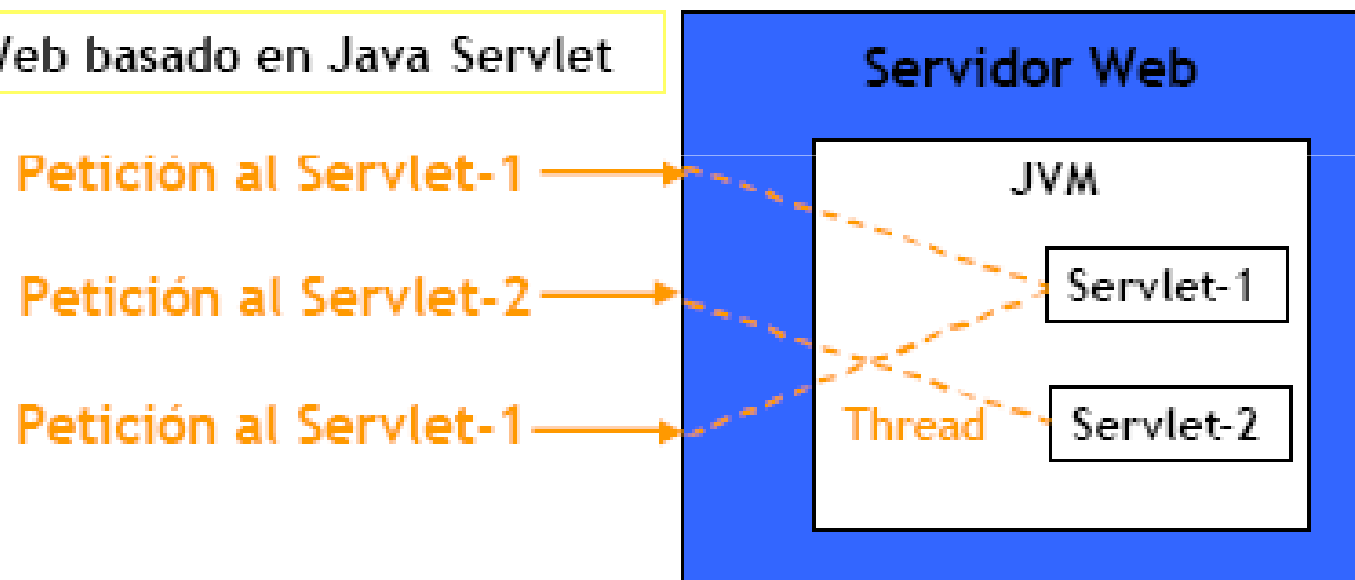
- Eficiencia.
  - Cada petición por parte de un cliente crea un hilo, no un nuevo proceso como ocurría con los CGI's tradicionales.
- Potencia.
  - Son programados en Java, por lo que se puede emplear todas las clases y herramientas disponibles para esta plataforma.
- Seguridad.
  - Controlada por la máquina virtual de Java.
  - La mayoría de problemas de seguridad encontrados en los CGI's no aparecen en los Servlets.
- Portabilidad.
  - Puede ser utilizados sobre cualquier SO. y en la mayoría de servidores Web.
- Precio.
  - Normalmente todo el software necesario es gratis.

## Ventajas de los Servlets

Servidor Web basado en CGI



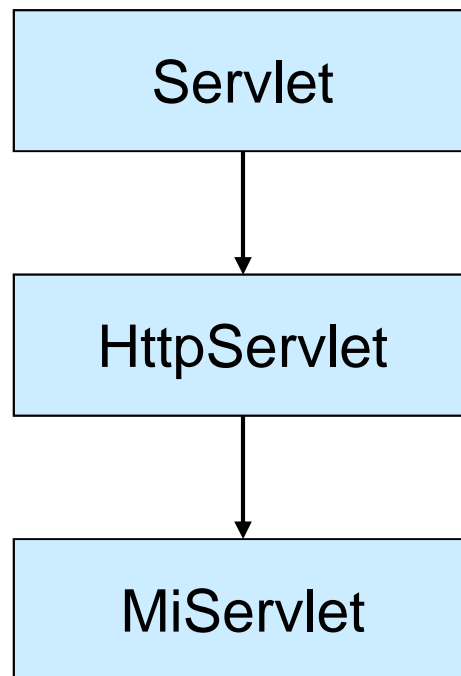
Servidor Web basado en Java Servlet



# Arquitectura de los Servlets

---

- Para implementar los servlets se hace uso de:
  - javax.servlet: entorno básico
  - javax.servlet.http: extensión para servlets http.



# Estructura básica

---

```
import java.io.*;                // Para PrintWriter
import javax.servlet.*;          // Para ServletException
import javax.servlet.http.*;     // Para HttpServlet*

public class PlantillaServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
        // El objeto "request" se utiliza para leer la
        // cabecera HTTP, cookies, datos enviados (GET o POST)
        // El objeto "response" para fijar la respuesta
        PrintWriter out = response.getWriter();
        // out se utiliza para enviar el contenido al cliente
    }
    // Idem para el método doPost
}
```

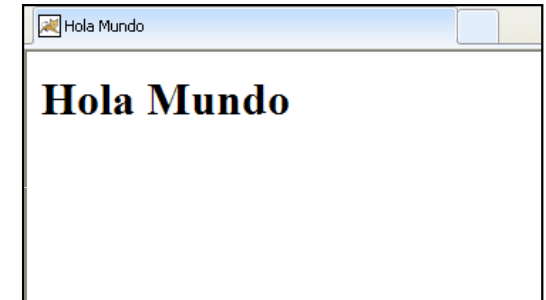


## SERVLETS: 1.1 Introducción

# Ejemplo

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

public class HolaMundo extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println( "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD \" + \"HTML
4.0 Transitional//EN\">" +
            "<html> <head><title>Hola Mundo</title></head>" +
            "<body> <h1>Hola Mundo</h1> </body></html>");
        out.close();
    }
}
```



# Compilando e Invocando el Servlet

---

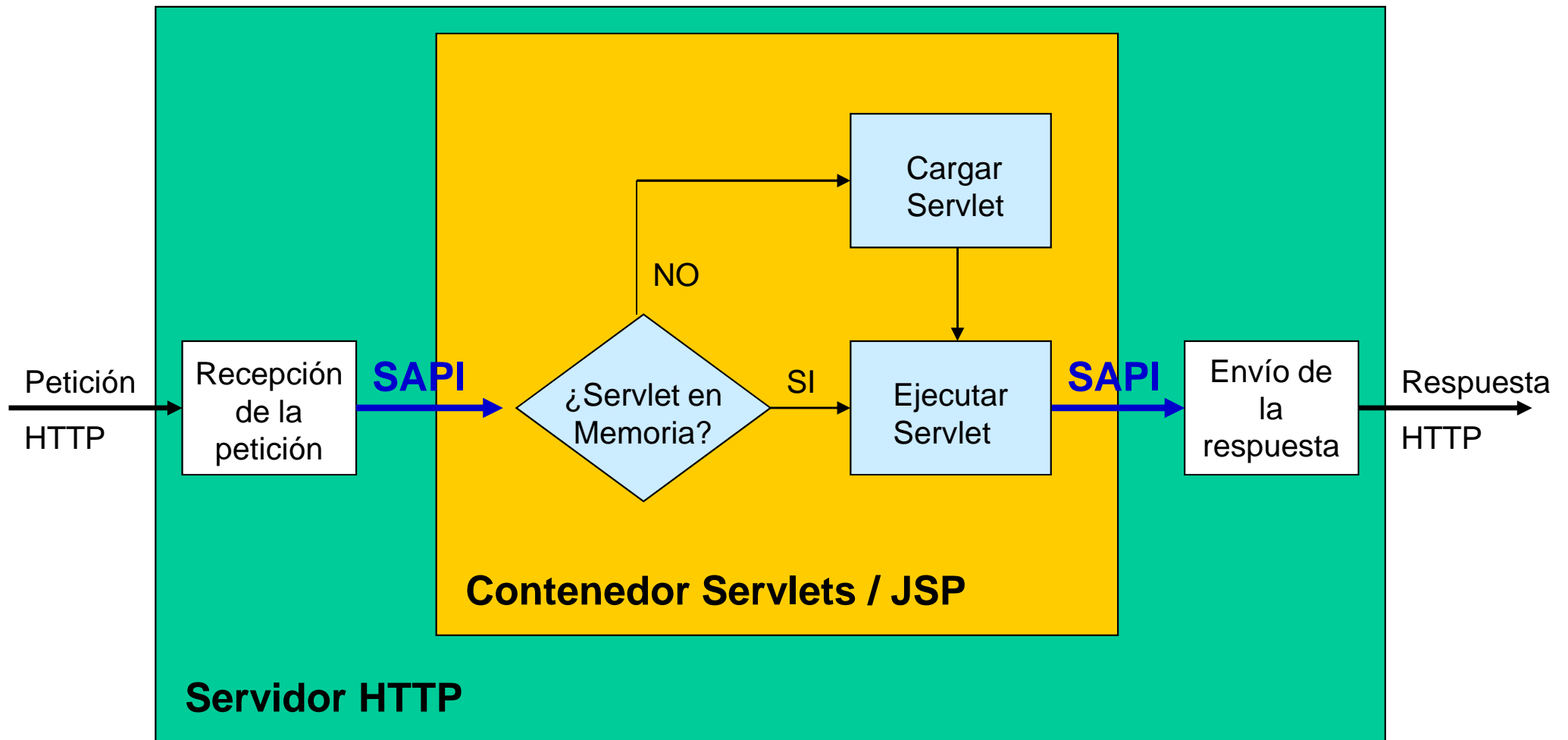
- EL principal servidor (gratuito) de servlets y JSP es “Tomcat” de Apache
  - (<http://jakarta.apache.org/tomcat>)
- Pasos a seguir para el servidor Tomcat:
  - Verificar que el servidor está corriendo
  - Compilar el servlet:
    - `> javac -classpath dir_instalacion/lib/servlet.jar nom_servlet`
  - Situar el servlet compilado en una dirección adecuada:
    - `> cp nom_servlet.class dir_tomcat/webapps/ROOT/WEB-INF/classes`
  - Invocar el servlet desde el browser:
    - `> http://servidor:puerto/servlet/nom_servlet`  
(El puerto se configura en `dir_install/conf/server.xml`)

# Ciclo de vida de un servlet

---

- El servidor recibe una petición que ha de ser manejada por un servlet.
- El servidor comprueba si existe una instancia creada en memoria de la clase servlet correspondiente. Si no, la crea.
- Las peticiones posteriores de otros usuarios utilizarán la misma instancia.
- El objeto servlet permanece en memoria mientras el servidor siga en funcionamiento.

# Diagrama del ciclo de vida



# Métodos implícitos (ciclo de vida)

---

### ■ **init**

- Se ejecuta una vez, la primera vez que es invocado el servlet (el servlet se carga en memoria y se ejecuta sólo la primera vez que es invocado. El resto de peticiones generan un hilo).

### ■ **service** (no debe sobrecribirse)

- Se ejecuta cada vez que se produce una nueva petición.
- Dentro de esta función se invoca a doGet o a doPost.

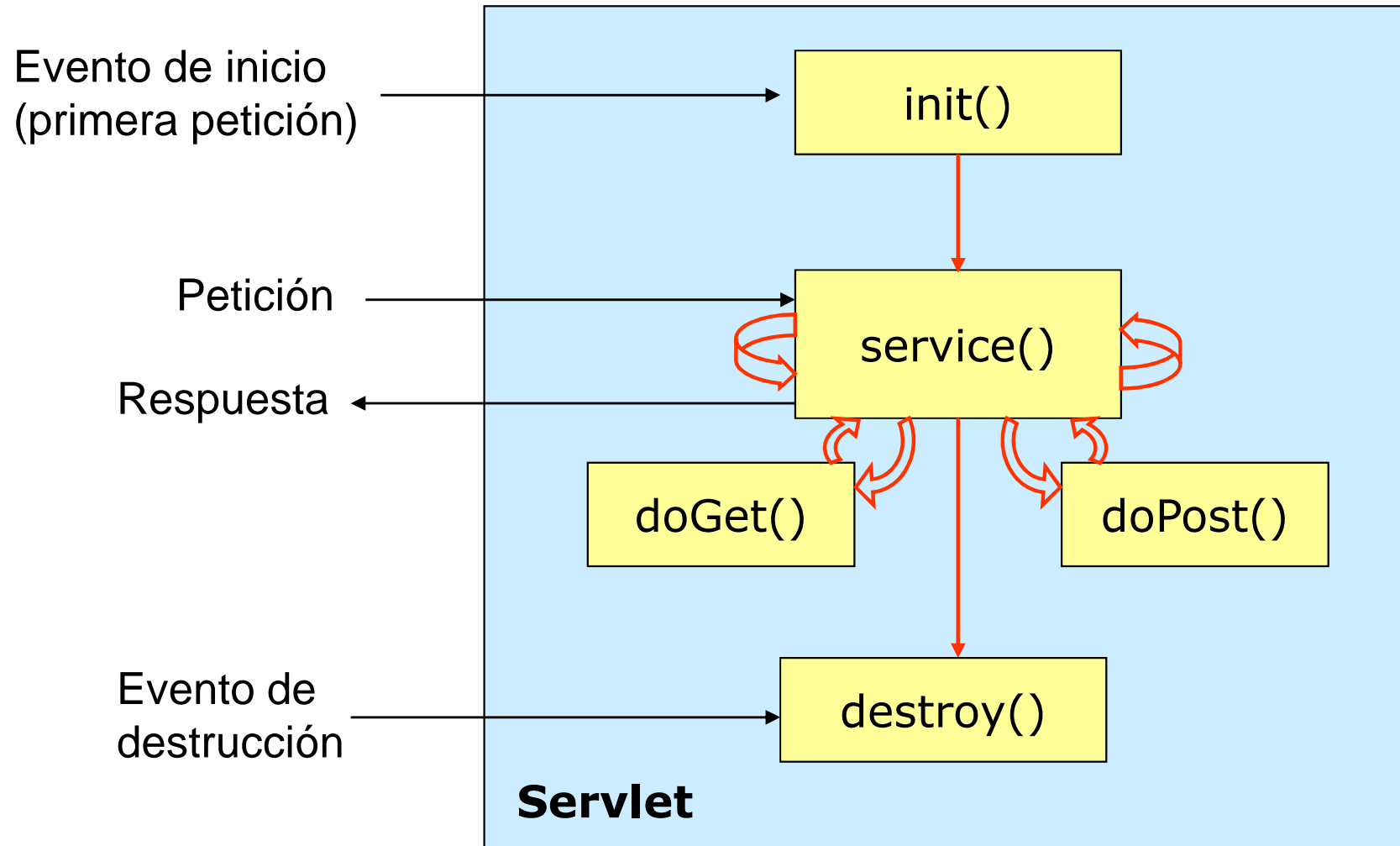
### ■ **doGet y doPost**

- Manejan las peticiones GET y POST.
  - Incluyen el código principal del servlet
- La ejecución del servlet finalizará cuando termine la ejecución de estos métodos.

### ■ **destroy**

- Se invoca cuando el servidor decide eliminar el servlet de la memoria (NO después de cada petición).

# Ciclo de ejecución de los métodos



# Objetos implícitos (I)

---

- Existen una serie de objetos implícitos, disponibles dentro de nuestros servlets (instanciados por el propio contenedor de servlets y JSP).
- Objeto **request**
  - Es una instancia de `HttpServletRequest` (`javax.servlet.http.HttpServletRequest`)
  - Recoge la información enviada desde el cliente.
- Objeto **response**
  - Es una instancia de `HttpServletResponse` (`javax.servlet.http.HttpServletResponse`)
  - Organiza los datos enviados al cliente.
- Objeto **session**
  - Es una instancia de `HttpSession` (`javax.servlet.http.HttpSession`)
  - Almacena información con ámbito de sesión.
  - La información se mantendrá hasta que se cierre el navegador.

## Objetos implícitos (II)

---

- Objeto **application**

- Es una instancia de ServletContext (javax.servlet.ServletContext)
- Almacena información con ámbito de servidor, (podrá ser “vista” por los demás servlets y JSPs del servidor).
- Se mantendrán hasta que se reinicie el servidor.

- Objeto **out**

- Es una instancia de PrintWriter (java.io.PrintWriter)
- Escribe contenido dentro de la página HTML.

- Objeto **config**

- Es una instancia de ServletConfig (javax.servlet.ServletConfig)
- Contiene información relacionada con la configuración del servlet.



## SERVLETS: 1.1 Introducción

### Ejemplo (I)

---

```
public class MuestraMensaje extends HttpServlet {
    private String mensaje;
    private String mensaje_por_defecto = "No hay mensaje";
    private int repeticiones = 1;
    public void init() throws ServletException {
        ServletConfig config = getServletConfig();
        mensaje = config.getInitParameter("mensaje");
        if (mensaje == null) {
            mensaje = mensaje_por_defecto;
        }
        try {
            String repetir_cad =
                config.getInitParameter("repeticiones");
            repeticiones = Integer.parseInt(repetir_cad);
        } catch (NumberFormatException nfe) {}
    }
}
```

# Ejemplo (II)

```
// (continua ..)
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String titulo = "Servlet MuestraMensaje";
    out.println("<html><head><title>" + titulo +
               "</title></head>" +
               "<body bgcolor=\"#FDF5E6\">\n" +
               "<h1 align=\"left\">" + titulo + "</h1>");
    for(int i=0; i< repeticiones; i++)
        out.println(mensaje + "<br/>");
    out.println("</body></html>");
    out.close();
}
}
```



# Ejemplo (III) - web.xml

---

- `<?xml version="1.0" encoding="iso-88591" ?>`
- `<!DOCTYPE web-app PUBLIC=".....(DTD)">`
- `<web-app>`
- `<servlet>`
- `<servlet-name>Muestra Mensaje</servlet-name>`
- `<servlet-class>MuestraMensaje</servlet-class>`
- `<init-param>`
- `<param-name>mensaje<param-name>`
- `<param-value>Bienvenido<param-value>`
- `</init-param>`
- `<init-param>`
- `<param-name>repeticiones</param-name>`
- `<param-value> 5 </param-value>`
- `</init-param>`
- `</servlet>`
- `<servlet>`
- `....`
- `</servlet>`
- `....`
- `</web-app>`

# Datos enviados desde el cliente

---

- El objeto request contiene todos los datos enviados desde el cliente al servidor.
- Todos los servlets implementan la interfaz `ServletRequest`, que define métodos para acceder a:
  - Los parámetros enviados por el cliente dentro de la URL o dentro del cuerpo del mensaje (p.e. a partir de un formulario)
  - Los valores de la cabeceras HTTP del mensaje
    - Cookies
    - Información sobre el protocolo
    - Content-Type
    - Si la petición fue realizada sobre un canal seguro SSL
    - etc.
  - Los datos de otras entradas.

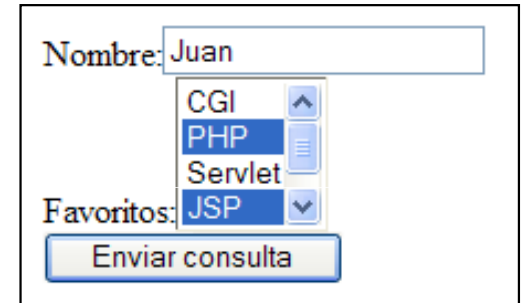
## Datos de un formulario

---

- La forma de leer los datos enviados desde un formulario es independiente del método de envío (GET o POST).
- `String request.getParameter("nom_var")`
  - Devuelve el valor (decodificado URL-encoded) encontrado en la primera ocurrencia de la variable dentro de los datos enviados por el cliente.
  - Devuelve null si la variable no ha sido enviada.
- `String[] request.getParameterValues("nom_var")`
  - Devuelve un array de valores (decodificados URL-encoded) con todos los valores asociados a la variable (SELECT multiple). Si sólo aparece un vez, devuelve un array de un elemento.
  - Devuelve null si la variable no ha sido enviada.
- `Enumeration request.getParameterNames()`
  - Devuelve una enumeración con los nombres de las variables enviadas.

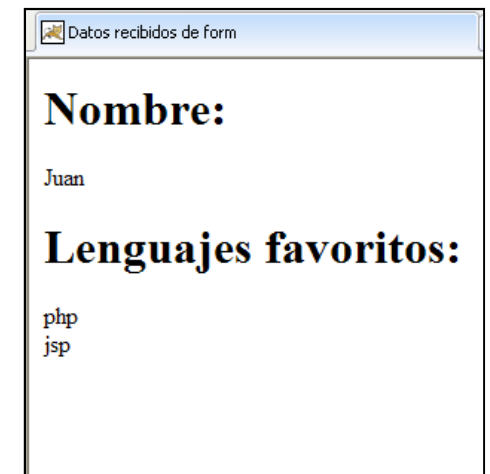
# Datos de un formulario (form.html)

```
<html>
<head><title>Formulario</title>
</head>
<body>
<form action="servlet/Ejemplo" method="POST">
  Nombre:<input type="text" name="nombre"><br/>
  Favoritos:<select name="favoritos" multiple>
    <option value="cgi">CGI</option>
    <option value="php">PHP</option>
    <option value="servlet">Servlet</option>
    <option value="jsp">JSP</option>
    <option value="asp">ASP</option>
  </select><br/>
  <input type="submit">
</form></body>
</html>
```



# Datos de un formulario (Ejemplo.class)

```
public class Ejemplo extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>Datos recibidos de form"
            + "</title></head><body>\n" +
            "<h1>Nombre:</h1>" + request.getParameter("nombre") +
            "<h1>Lenguajes favoritos:</h1>");
        String[] lengj= request.getParameterValues("favoritos");
        for (int i = 0; i < lengj.length; i++ )
            out.println( lengj[i] + "<br/>" );
        out.println("</body></html>");
        out.close();
    } // Fin doGet
}
```



## Datos de un formulario (Ejemplo.class)

---

```
public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
} // Fin doPost
} // Fin clase
```



# Datos de un formulario (Parametros.class)

---

```
public class Parametros extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException  
    {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        String titulo = "Todos los parámetros recibidos";  
        out.println("<html><head><title>" + titulo +  
                    "</title></head><body>\n" +  
                    "<h1 align=\"center\">" + titulo + "</h1>\n" +  
                    "<table border=\"1\" align=\"center\">\n" +  
                    "<tr><th>Nombre<th>Valor(es)");  
    }  
}
```

# Datos de un formulario (Parametros.class)

```
Enumeration nombres_param =
    request.getParameterNames();
while(nombres_param.hasMoreElements()) {
    String nombre_p =
        (String)nombres_param.nextElement();
    out.print("<tr><td>" + nombre_p + "<td>");
    String[] valores_param =
        request.getParameterValues(nombre_p);
    if (valores_param.length == 1)
        out.println(valores_param[0]);
    else {
        out.println("<ul>");
        for(int i=0; i<valores_param.length; i++)
            out.println("<li>" + valores_param[i]);
        out.println("</ul>");
    }
} // Fin while
out.println("</table></body></html>");
out.close();
```

# Datos de un formulario (Parametros.class)

---

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
}
} // Fin clase
```

## Todos los parámetros recibidos

Nombre	Valor(es)
favoritos	<ul style="list-style-type: none"><li>• php</li><li>• jsp</li></ul>
nombre	Juan

### Cabecera HTTP (I)

---

- Existe dos funciones de propósito general para extraer cualquier parámetro de la cabecera HTTP enviada por el cliente:
  - String **getHeader** ( String parametro\_HTTP )
    - Devuelve una cadena con el valor del parámetro.
    - Devuelve null, si el parámetro no está incluido en la cabecera.
  - Enumeration **getHeaders** ( String parametro\_HTTP )
    - Similar a la anterior. Se emplea para recoger los valores de aquellos parámetros que se repiten varias veces dentro de la cabecera.
- Para averiguar todos los parámetros enviados en la cab.:
  - Enumeration **getHeaderNames**()
    - Devuelve una enumeración con los nombres de todos los parámetros incluidos en la cabecera.

## Cabecera HTTP (II)

---

- Existen un conjunto de funciones para extraer los valores de algunos parámetros particulares:
  - Cookie[] **getCookies** ()
    - Extrae las cookies enviadas por el cliente.
  - String **getMethod** ()
    - Método utilizado en la petición (GET o POST).
  - String **getContentLength** ()
    - Longitud de los datos enviados por el cliente (utilizando el método POST) tras la cabecera HTTP.
  - String **getContentType** ()
    - Devuelve el tipo MIME de los datos enviados tras la cabecera.
  - String **getProtocol** ()
    - Devuelve la versión del protocolo HTTP (HTTP/1.0 o HTTP/1.1) utilizado por el cliente en la petición.

# Cabecera HTTP (III)

---

```
public class ContenidoCabeceraHTTP extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String titulo = "Muestra el contenido de la cab.";
        out.println("<html><head><title>" + titulo +
            "</title></head><body>\n" +
            "<h1 align=\"center\">" + titulo + "</h1>\n" +
            "<b>Muestra el contenido de la cab.: </b>" +
            request.getMethod() + "<br/>\n" +
            "<b>Protocolo: </b>" +
            request.getProtocol() + "<br/><br/>\n" +
            "<table border=\"1\" align=\"center\">\n" +
            "<tr bgcolor=\"#FFAD00\">\n" +
            "<th>Nombre del parámetro<th>Valor");
```

## SERVLETS: 1.2 Objeto Request

# Cabecera HTTP (IV)

```
Enumeration nombres_par = request.getHeaderNames();
while(nombres_par.hasMoreElements()) {
    String nom_cab = (String) nombres_par.nextElement();
    out.println("<tr><td>" + nom_cab);
    out.println("<td>" + request.getHeader(nom_cab));
}
out.println("</table>\n</body></html>");
out.close();
}
public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
doGet(request, response);
}
}
```

### Muestra el contenido de la cab.

Método de envío: GET  
Protocolo: HTTP/1.1

Nombre del parámetro	Valor
accept	image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, */*
accept-language	es
user-agent	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
accept-encoding	gzip, deflate
host	slopez.uv.es:8080
connection	Keep-Alive

# Datos enviados al cliente

---

- El objeto response representa los datos enviados desde el servidor al cliente
  - Se emplea, no sólo para escribir el contenido de la página enviada al cliente, sino también para organizar la cabecera HTTP, enviar cookies, etc.
- Todos los servlets implementan el interfaz de `ServletResponse`, que permite:
  - Acceder al canal de salida
  - Indicar el tipo de contenido del mensaje de respuesta
  - Indicar si existe buffer de salida
  - Establecer la localización de la información
- `HttpServletResponse` extiende a `ServletResponse`:
  - Código de estado del mensaje de respuesta
  - Cookies



# Estado de la respuesta (I)

---

- La primera línea de la cabecera HTTP describe el estado de la respuesta.
- Para manipular directamente este parámetro:
  - **setStatus( int *codigo* )**  
donde *codigo* es el número del código del estado. En vez de utilizar el número se puede emplear las constantes predefinidas:
    - **SC\_OK** , que representa el estado: 200 Ok
    - **SC\_MOVED\_PERMANENTLY**: 301 Moved Permanently
    - **SC\_MOVED\_TEMPORALY** : 302 Found
    - **SC\_BAD\_REQUEST** : 400 Bad Request
    - **SC\_UNAUTHORIZED** : 401 Unauthorized
    - **SC\_NOT\_FOUND** : 404 Not Found
    - **SC\_INTERNAL\_SERVER\_ERROR** : 500 ..
    - Etc.

## Estado de la respuesta (II)

---

(Cont..)

- **sendError** ( int *codigo*, String *mensaje*)
  - Manda un código de estado de error (4XX), y escribe el contenido de *mensaje* en el cuerpo del documento HTML.
- **sendRedirect** (String *url*)
  - Redirecciona el navegador del cliente hacia la dirección *url*.
  - Manda un código de estado SC\_MOVED\_TEMPORALY, y asigna al parámetro “Location” de la cabecera HTTP la dirección *url*.

### Estado de la respuesta (III)

---

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>....
    String nueva_direccion;
    ....
    if ( nueva_direccion.length() != 0 ) {
        response.sendRedirect ( nueva_direccion );
    } else {
        response.sendError ( response.SC_NOT_FOUND,
            "<h2>La nueva dirección no es valida</h2>" );
    }
    return;
}
```

## Parámetros de la cabecera HTTP

---

- Para fijar cualquier parámetro de la cabecera:
  - **setHeader** (String nombre\_param, String valor\_param)  
`response.setHeader( "Cache-Control", "no-cache" );`
- Para ciertos parámetros, existen funciones especiales:
  - **setContentType** ( String *codigo\_MIME* )  
Fija el código MIME de la respuesta (Content-Type)  
`response.setContentType( "text/html" );`
  - **addCookie** (Cookie *la\_cookie*)
    - Envía una cookie al cliente.

# Cuerpo del mensaje

---

- El cuerpo del mensaje es generado a partir de los objetos:
  - **PrintWriter**
    - La referencia se extrae con **response.getWriter()**
    - Cuando el código generado es texto HTML (o XML, o plano)

```
PrintWriter out = response.getWriter();  
out.println("..."); out.flush(); out.close();
```

- **ServletOutputStream**
  - La referencia se extrae con **response.getOutputStream()**
  - Cuando el código generado es binario (p.e. una imagen)

```
ServletOutputStream out = response.getOutputStream();
```

# Enviando páginas comprimidas (I)

---

```
public void doGet ( HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException
{
    response.setContentType("text/html");
    String encodings = request.getHeader("Accept-Encoding");
    PrintWriter out;
    String titulo;
    if ( (encodings != null) &&
        (encodings.indexOf("gzip") != -1) )
    {
        titulo = "Página comprimida con GZip";
        ServletOutputStream out1 = response.getOutputStream();
        out = new PrintWriter(new
                               GZIPOutputStream(out1), false);
        response.setHeader("Content-Encoding", "gzip");
    }
}
```

# Enviando páginas comprimidas (II)

---

```
else {
    titulo = "Página no comprimida";
    out = response.getWriter();
}
out.println("<html><head><title>" + titulo +
    "</title></head><body>\n" +
    "<h1 align=\"center\">" + titulo + "</h1>\n");
String linea = " ..... ";
for(int i=0; i<10000; i++)
    out.println(linea);
out.println("</body></html>");
out.close(); //en binario no se envía hasta que se cierra
}
```

# SERVLETS: Mantener Información sobre un cliente

## Formas de seguir la trayectoria de los usuarios

---

- HTTP es un protocolo “*sin estado*”
  - Cada vez que un cliente pide una página Web, abre una conexión separada con el servidor Web y el servidor no mantiene automáticamente *información contextual* acerca del cliente.
- Servlets
  - Permiten obtener y mantener una determinada información acerca de un cliente.
  - Información accesible a diferentes servlets o entre diferentes ejecuciones de un mismo servlet.
- Tres soluciones típicas
  - Cookies
  - Seguimiento de sesiones ( session tracking)
  - Reescritura de URLs



# Enviando/Recibiendo Cookies

- Para enviar cookies al cliente se crea un objeto de la clase `Cookie`, y se invoca el método **`addCookie`** del objeto `response` pasándole como parámetro dicha cookie.

```
Cookie c = new Cookie("nombre", "valor");  
c.setMaxAge(...); // Segundos de vida del cookie  
response.addCookie(c);
```

- Para leer las cookies se emplea el método **`getCookies`** del objeto `request`. Éste devuelve un array con todas las cookies recibidas del cliente.

```
Cookie[] cookies_recb = request.getCookies();  
if (cookies_recb != null)  
    for(int i=0; i<cookies_recb.length; i++) {  
        if (cookies_recb[i].getName().equals("alquiler"))  
            && (cookies_recb[i].getValue().equals("coche"))  
            {cookies_recb[i].setMaxAge(0); //Elimina la cookie  
            } //fin del if  
    } //fin del for  
} // fin del if
```

## Métodos del objeto Cookie (I)

---

- public String **getName()** /  
public void **setName** ( String *nombre\_cookie* )
  - Extrae / fija el nombre del cookie. La función setName raramente se utiliza, ya que el nombre de la nueva cookie normalmente se fija en el constructor del objeto.
- public String **getValue()** /  
public void **setValue** ( String *valor\_cookie* )
  - Extrae / fija el valor de la cookie. La función setValue normalmente no se utiliza (el valor se fija en el constructor).
- public int **getMaxAge()** /  
public void **setMaxAge** ( int *segundos\_vida* )
  - Extrae / fija el número de segundos que la cookie permanece guardado en el disco del cliente.

## Métodos del objeto Cookie (II)

---

- public String **getDomain()** /  
public void **setDomain** ( String *dominio* )
  - Extrae / fija el dominio de los servidores con acceso a la cookie.
- public String **getPath()** /  
public void **setPath** ( String *camino* )
  - Extrae / fija el directorio raíz (virtual) de las páginas con acceso a la cookie.
- public boolean **getSecure()** /  
public void **setSecure** ( boolean *flag\_seguridad* )
  - Extrae / fija el parámetro de seguridad. Si *flag\_seguridad* vale true, la cookie sólo será enviada si la conexión es segura (SSL).

# Ejemplo UtilidadesCookie

---

```
public class UtilidadesCookie
{
    public static String ExtraeValor ( Cookie[] cookies,
                                      String nom_cookie )
    {
        String valor = "";
        if ( cookies != null )
            for ( int i=0; i<cookies.length; i++) {
                Cookie cookie=cookies[i];
                if ( nom_cookie.equals(cookie.getName()) )
                    valor = cookie.getValue();
            }
        return valor;
    }
}
```

# Ejemplo UtilidadesCookie

---

```
public static boolean EsSegura ( Cookie[] cookies,
                                String nom_cookie )
{
    boolean segura = false;
    if ( cookie != null )
        for ( int i=0; i<cookies.length; i++) {
            Cookie cookie= cookies[i];
            if ( nom_cookie.equals(cookie.getName()) )
                segura = cookie.getSecure();
        }
    return segura;
}
// Fin UtilidadesCookie
```

## Objeto HttpSession

---

- Las sesiones se asocian al cliente, bien vía cookies, o bien rescribiendo la URL.
  - El sistema localiza el identificador de la sesión incluido dentro de la cookie, o incluido en la información extra de la URL de la petición. Cuando el identificador no corresponde a un objeto de tipo sesión previamente almacenado, crea una nueva sesión.
- Las sesiones se implementan a través de objetos de la clase HttpSession, creados por el contenedor cuando se inicia una sesión para un nuevo usuario.
  - Para extraer la referencia a este objeto desde un servlet:

```
HttpSession mi_sesion = request.getSession(true);
```
- Las sesiones se utilizan para almacenar variables que transmiten su valor a lo largo del conjunto de páginas visitadas por el cliente durante la sesión.

## API del objeto sesión (I)

---

- public void **setAttribute** (String *nombre*, Object *valor*)
  - Registra una nueva variable dentro de la sesión (*nombre* y *valor* son el nombre y el valor de la variable).
- public Object **getAttribute** ( String *nombre* )
  - Extrae el valor de una variable previamente registrada.
- public void **removeAttribute** (String *nombre*)
  - Borra una variable de la sesión previamente registrada.
- public Enumeration **getAttributeNames** ( )
  - Extrae el nombre de todas las variables registradas en la sesión
- public String **getId** ( )
  - Devuelve el identificador de la sesión.

## API del objeto sesión (II)

---

- public boolean **isNew** ( )
  - Devuelve true si la sesión comienza en esta página.
- public long **getCreationTime** ( )
  - Momento de la creación de la sesión (expresado en milisegundos transcurridos desde el 1 de enero de 1970).
- public long **getLastAccessedTime** ( )
  - Momento del último acceso a una página de la sesión (milisegundos transcurridos desde el 1 de enero de 1970).
- public int **getMaxInactiveInterval** ( ) /  
public void **setMaxInactiveInterval** ( int *segundos* )
  - Extrae / fija los segundos que deben transcurrir desde el último acceso para que la sesión sea cerrada.
- public void **invalidate** ( )
  - Invalida la sesión actual.



## Ejemplo 1

---

...

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

```
CarritoCompras compra =
```

```
    (CarritoCompras)miSesion.getAttribute(miSesion.getId());
```

```
if(compra==null)
```

```
{
```

```
    compra = new CarritoCompras();
```

```
    miSesion.setAttribute(miSesion.getId(), compra);
```

```
}
```

..

### Ejemplo 2

---

```
.....  
HttpSession session = request.getSession(true);  
Integer acc = (Integer)session.getAttribute("accesos");  
String presentacion;  
if (acc == null) {  
    acc = new Integer(0);  
    presentacion = "Bienvenido, nuevo usuario";  
} else {  
    presentacion = "Bienvenido de nuevo";  
    acc = new Integer(acc.intValue() + 1);  
}  
session.setAttribute("accesos", acc);  
.....
```

## Reescritura de URLs

---

- Puede suceder que ciertos clientes no soporten cookies o bien las rechacen
- Solución: Sesiones + Reescritura de URLs
  - El cliente añade ciertos datos extra que identifican la sesión al final de cada URL

`http://host/path/servlet/name?jsessionid=1234`
  - El servidor asocia ese identificador con datos que ha guardado acerca de la sesión
- Métodos: `encodeURL()` y `encodeRedirect()`
  - Leen un String ( URL o URL de redirección) y si es necesario lo reescriben añadiendo el identificador de la sesión.
- Algunas Desventajas
  - Se deben codificar todas las URLs referentes al sitio propio
  - Todas las páginas deben generarse dinámicamente

# Ejemplo

---

```
...
HttpSession miSesion=req.getSession(true);
CarritoCompras compra =
(CarritoCompras)miSesion.getAttribute(miSesion.getId());
if(compra==null) {
compra = new CarritoCompras();
miSesion.setAttribute(miSesion.getId(), compra);
}
...
PrintWriter out = resp.getWriter();
resp.setContentType("text/html");
...
out.println("Esto es un enlace reescrito");
out.println("<a href\""+
resp.encodeUrl("/servlet/buscador?nombre=Javier")+ "\"</a>");
...
```

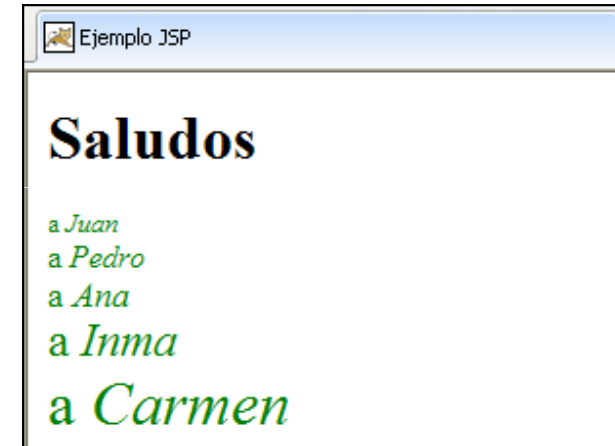
# Marco JSP

---

- Los servlets son aplicaciones java que crean contenido HTML a base de sentencias “out.print”
  - Se hace tedioso crear páginas HTML.
  - Es más complicado mantener el contenido HTML.
- JSP es la otra alternativa java a la generación de contenidos de forma dinámica en el lado del servidor.
- El código Java queda embebido dentro del código HTML de forma similar a PHP o ASP.
  - Separa el código java del HTML.
  - Más conveniente que los servlets para generar contenido HTML.
- JSP es en el fondo una tecnología equivalente a los servlets.
  - Las páginas JSP se traducen en servlets que ejecuta el servidor en cada petición.

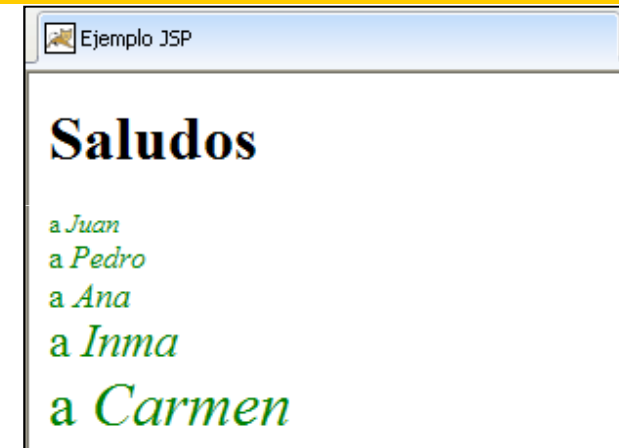
# Ejemplo.jsp

```
<html>
<head><title>Ejemplo JSP</title>
</head>
<body>
<h1>Saludos</h1>
<!-- Esto es un comentario -->
<%
    String[] nombres={"Juan","Pedro","Ana","Inma","Carmen"};
    for ( int i = 0; i < nombres.length; i ++ )
    {
%>
<font color="green" size="<%=i+2%>">
a <i><%= nombres[i]%></i></font><br/>
<% } %>
</body>
</html>
```



# Ejemplo.jsp - Código generado

```
<html>
<head><title>Ejemplo JSP</title>
</head>
<body>
<h1>Saludos</h1>
<font color="green" size="2"> a <i>Juan</i></font><br/>
<font color="green" size="3"> a <i>Pedro</i></font><br/>
<font color="green" size="4"> a <i>Ana</i></font><br/>
<font color="green" size="5"> a <i>Inma</i></font><br/>
<font color="green" size="6"> a <i>Carmen</i></font><br/>
</body>
</html>
```



# Ventajas (I)

---

- Frente a CGI.
  - Seguridad
    - Entorno de ejecución controlado por la JVM
  - Eficiencia
    - Cada nueva petición es atendida por un hilo, no por un nuevo proceso
- Frente a PHP
  - Lenguaje más potente para la generación dinámica
    - Lenguaje de script orientado a objetos (Java)
  - Mejores herramientas de desarrollo y soporte
- Frente a ASP
  - Mejores rendimientos.
    - Código compilado, no interpretado (SDK 1.4 o superior)
  - Lenguaje más potente para la generación dinámica (Java)
  - Independiente de la plataforma
    - Portable a múltiples servidores y SO.

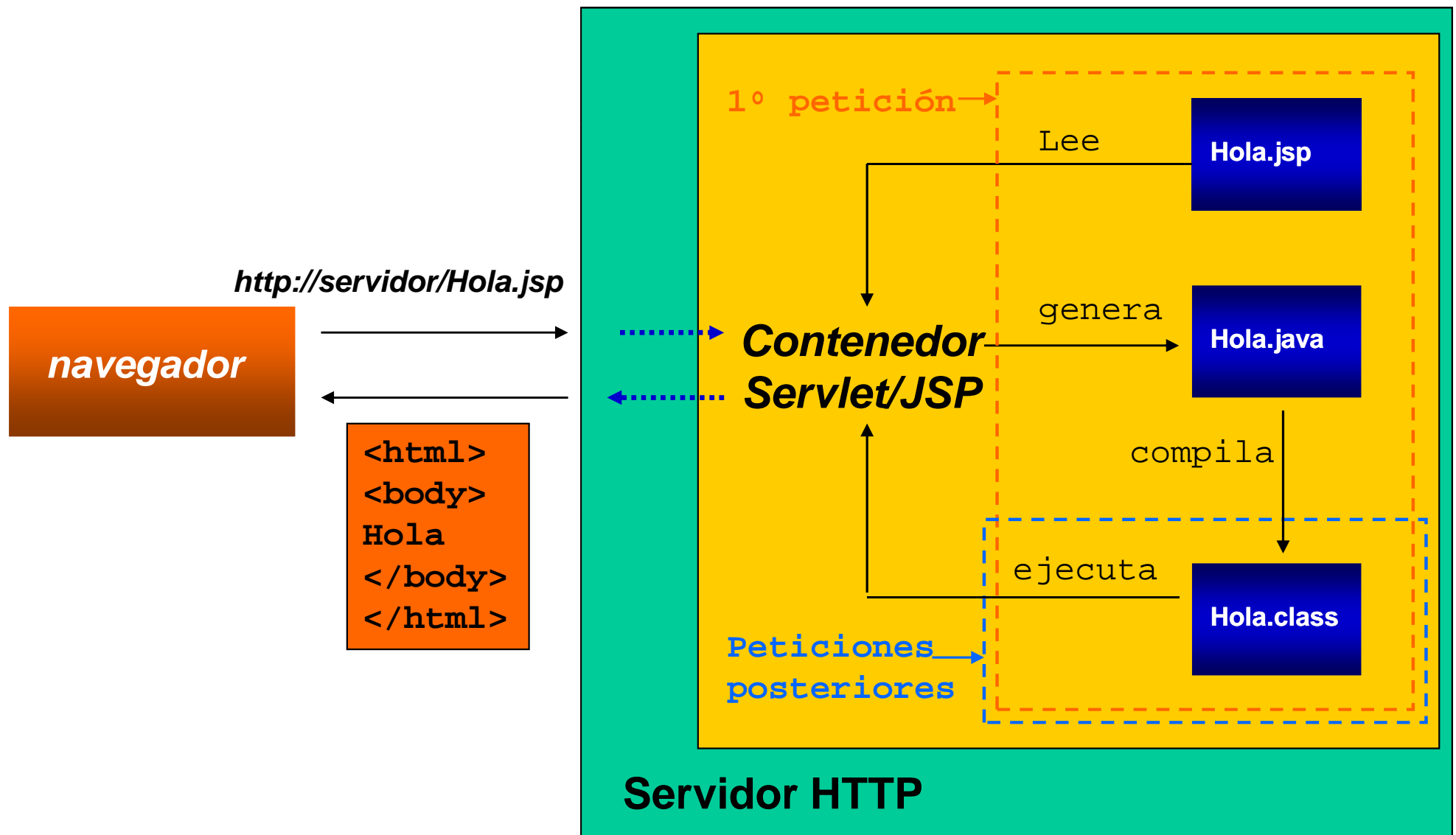


# Ventajas (II)

---

- Frente a servlets puros
  - La lógica de negocio y la presentación están más separados.
  - Simplifican el desarrollo de aplicaciones Web
    - Más conveniente para crear HTML (no es necesario println).
  - Más fácil para desarrolladores Web.
  - Soporte para reutilizar software a través de JavaBeans y etiquetas adaptadas.
  - Puede utilizarse herramientas estándar (p.e. Homesite)
  - Recompila automáticamente las modificaciones en las páginas jsp
  - No es necesario ubicar las páginas en un directorio especial
    - /srv/www/tomcat/base/webapps/ROOT/pagina.jsp
    - La URL tampoco es especial.
      - <http://www.uv.es/pagina.jsp>

# Ciclo de vida de una página JSP (I)



## Ciclo de vida de una página JSP (II)

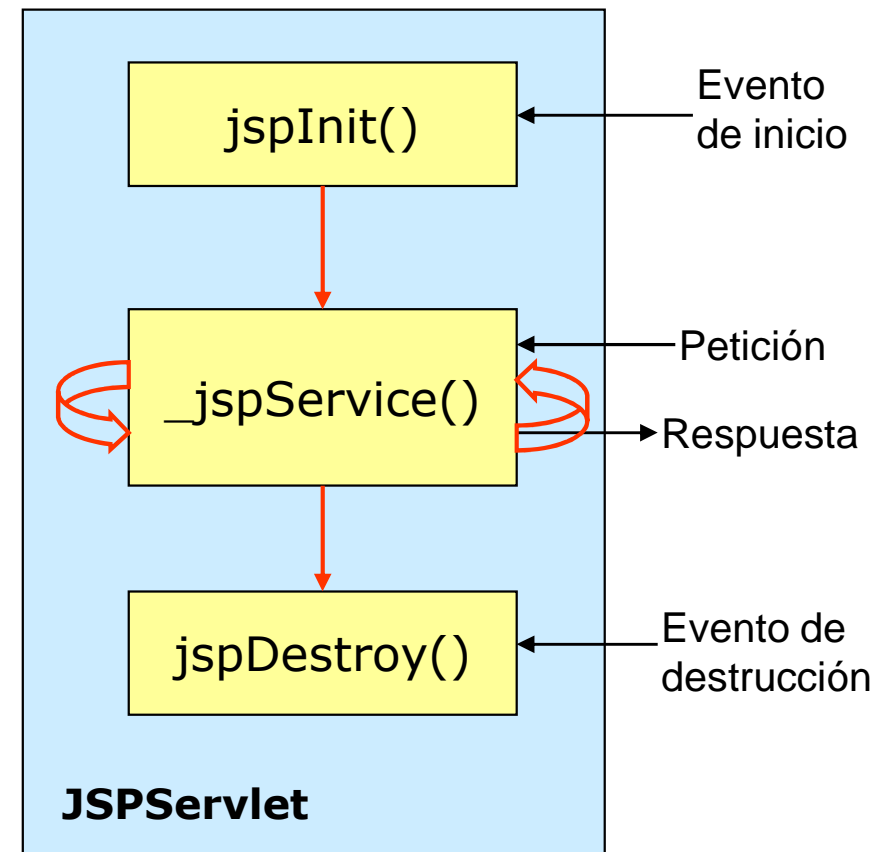
---

- Se divide en varias fases:
- Fase de traducción
  - Tras la primera petición de la página JSP, se traduce en un servlet (código java).
- Fase de compilación
  - Dicho servlet es compilado para poder servir la petición del cliente.
  - Normalmente las fases de traducción y compilación ocurren juntas, y son realizadas por el contenedor automáticamente en la primera petición.
- Fase de ejecución
  - Se crea una instancia de dicho servlet, que residirá en memoria de forma permanente mientras el servidor siga en funcionamiento.
  - Para las peticiones posteriores se emplea la misma instancia del servlet (no se vuelve a compilar la página).

# Ciclo de ejecución de los métodos

- Durante la fase de ejecución, el contenedor invoca del servlet generado los métodos:

- **jspInit():**
  - Permite ejecutar cierto código cuando se produce la primera petición.
- **\_jspService():** Se ejecuta en TODAS las peticiones.
  - El programador JSP no debe administrarlo directamente.
  - La mayoría del código java y HTML se incluye dentro.
- **jspDestroy():**
  - Permite ejecutar código antes de que finalice el servlet.



# Correspondencia JSP/servlet

---

- Ejemplo.jsp:

```
<html><head><title> Aleatorio</title></head>
<body><h3>num. aleatorio: <%= Math.random()%> </h3>
</body></html>
```

- Servlet generado (simplificado):

```
public class Ejemplo_jsp extends HttpJspBase {
    public void _jspService (HttpServletRequest request,
                             HttpServletResponse response)
        throws ServletException, IOException {
        request.setContentType("text/html");
        JspWriter out = response.getWriter();
        out.println("<html><head><title> Aleatorio</title></head>");
        out.print("<body> <h3>num. aleatorio:");
        out.println(Math.random());
        out.println("</h3></body></html>");
    }
}
```

# Elementos básicos

---

- **Los comentarios en las páginas JSP se escriben:**

- `<%-- comentario --%>`

- **Elementos de script**

- Los elementos JSP permiten escribir instrucciones java combinadas con las propias del HTML, este código Java se insertará en el servlet que se genera a partir de la página JSP.
  - Tres tipos:
    - **Scriptlets:**
      - El código se inserta en el método `_jspService` del servlet
    - **Expresiones:**
      - Evaluadas e insertadas en la salida del servlet
    - **Declaraciones:**
      - El código es insertado en la clase del servlet, fuera de los métodos.

- **Directivas**

- Instrucciones que controlan la generación del servlet que resulta de la página JSP.

## Scriptlets

---

- Formato usual:
  - `<% código %>`
- Formato compatible XML:
  - `<jsp:scriptlet>` código `</jsp:scriptlet>`
- Usado para introducir código java arbitrario dentro del método `_jspService` (invocado desde `service`)

```
<% String datos_enviados = request.getQueryString();  
    out.println("Datos enviados con el método GET:" +  
    datos_enviados + "<br/>"); %>
```

# Expresiones

---

- Formato usual:  
`<%= expresión %>`
- Formato compatible XML:  
`<jsp:expression>expresión </jsp:expression>`
- Escribe directamente dentro de la página HTML el resultado.
  - Se evalúa el resultado, se convierte en cadenas y se inserta dentro de la página HTML

Datos Enviados: `<%= request.getQueryString()%><br/>`

Hora actual: `<%= new java.util.Date()%><br/>`



# Declaraciones

- Se usan para indicar declaraciones de variables globales, es decir persistentes.
- Formato:
  - `<%! Código %>`
  - `<jsp:declaration>` código `</jsp:declaration>`
- Se utiliza para declarar nuevos atributos y métodos dentro de la clase servlet derivada (fuera de cualquier método existente), que puede ser utilizados dentro de scriptlets y expresiones.

```
<%! public java.util.Date FechaActual() {  
    return (new java.util.Date());  
} %>  
<html>  
<head><title>Ej. declaración</title></head><body>  
La fecha actual es: <%= FechaActual()%>  
</body></html>
```

## Directivas

---

- Formato:
  - `<%@ directiva %>`
  - `<jsp:directive directiva />`
- Dan información de alto nivel sobre el servlet que será generado a partir de la página JSP.
- Controlan:
  - Las clases importadas.
  - La clase padre del servlet.
  - El tipo MIME generado.
  - La inclusión del servlet dentro de sesiones.
  - El tamaño y comportamiento del buffer de salida.
  - Las páginas que manejan los errores inesperados.

## Directiva page (I)

---

- Formato:
  - **<%@ page atributo="valor" %>**
- Configura atributos de la página JSP.
- Controlan:
  - Las clases importadas.
    - Genera una instrucción import en el servlet.
    - Formato: **<%@ page import="clase importada" %>**  
`<%@ page import="java.util.*" %>`
  - El tipo MIME generado.
    - Formato: **<%@ page contentType="tipoMIME" %>**  
`<%@ page contentType="text/plain" %>`

## Directiva page (II)

---

- Controlan (continuación ..)
  - Otros atributos:
    - **session** : si la página está incluida en sesiones.
    - **buffer** : Cambia el tamaño del buffer utilizado por JspWriter.
    - **extends** : cambia la clase padre del servlet.
    - **errorPage** : Designa una página para manipular los errores no planificados.

```
<%@ page errorPage= "pagina_error.jsp" %>
```
    - **isErrorPage** : Estipula que la página puede ser usada como página de error.

# Directiva include

- Incluye el contenido de un fichero texto (con código html y/o jsp) en una página JSP.
- Formato:
  - **<%@ include file="url\_relativa" %>** → **directiva**
    - Durante la fase de traducción (contenido fijo)
  - **<jsp:include page="url\_relativa" flush="true"/>** → **acción**
    - Durante la fase de ejecución (contenido cambiante)

```
<%@ page import="java.util.Date"%>
<%! private int num_accesos = 0;
      private Date fecha_acceso = new Date(); %>
<html><body> <% num_accesos++; %>
Accesos hasta el momento <%= num_accesos%><br/>
Fecha del ultimo acceso <%= fecha_acceso %><br/>
<% fecha_acceso = new Date(); %>
Fecha actual <%= fecha_acceso %><br/><hr/>
<%@ include file="InfoContacto.jsp" %></body></html>
```

# Directiva taglib

---

- Permite extender, dentro de la página, las etiquetas JSP con etiquetas personalizadas definidas por el propio programador.
- Formato:
  - `<%@ taglib uri="URLLibreria" prefix="PrefijoEtq" %>`
  - El valor de **uri** hace referencia a la biblioteca donde están definidas las etiquetas personalizadas, y el valor de **prefix** asocia el prefijo usado para distinguir dichas etiquetas.
- Ejemplo:

```
<%@ taglib uri="http://www.uv.es/ars_tags" prefix="ars" />
...
<ars:iniConsulta>
...
</ars:iniConsulta>
```

# Ejemplo2.jsp

```
<%@ page language="java"
    contentType="text/html"
    info="Mi primera página en JSP"
    import="java.util.*"
    errorPage="errorFatal.jsp" %>

<%! int count=0; %>

<html>
<head><title>Hola y n̄meros. Introduccīn
a JSP</title></head>
<body bgcolor="white">
Hola, mundo. Te lo repito <%= count++ %>
<% if (count == 1) { %>
vez
<% } else { %>
vecas
<% } %>
</body></html>
```

### Ejemplo3.jsp

```
<%@ page language="java"
contentType="text/html"
info="Mi primera página en JSP"
import="java.util.*"
errorPage="errorFatal.jsp" %>

<%! int count=0; %>

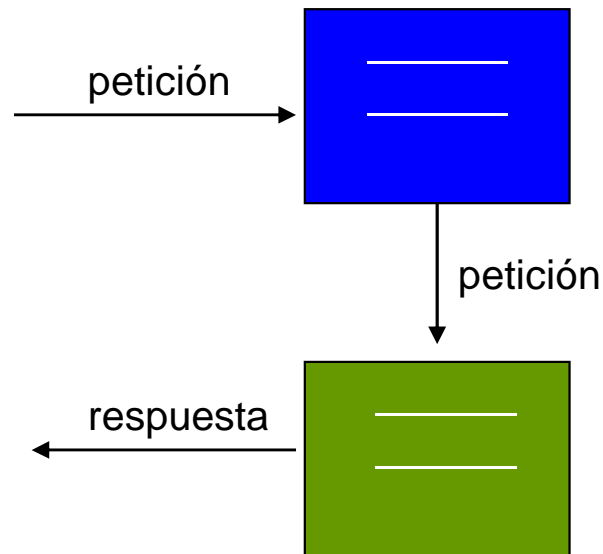
<html>
<head><title>Hola y números. Intro to JSP</title></head>
<body bgcolor="white">
Hola, mundo. Te lo repito <%= count++ %>
<% if (count == 1){
out.println('vez')
} else {
out.println('veces')
} %>
</body></html>
```



## Otros elementos: redirección (I)

---

- JSP (tb. servlets) permite pasar la petición a otras páginas JSP o servlets.



## Otros elementos: redirección (II)

---

- Formato:
  - **<jsp:forward page="url\_relativa"/>**
    - Durante la fase de ejecución (contenido cambiante)
  - La segunda página recibe en request los mismos parámetros que la página inicial.
    - Ejemplo:

```
<jsp:forward page="otra.jsp"/>
```
  - Aparte, puede añadir nuevos parámetros
    - Ejemplo:

```
<jsp:forward page="otra.jsp">  
  <jsp:param name="nuevo_param" value="uno"/>  
</jsp:forward>
```

## Objetos implícitos

---

- **request:** Engloba la información enviada desde el cliente.  
`<body bgcolor=<%=request.getParameter("color_fondo")%>>`
- **response:** Organiza la información enviada al cliente.  
`<% response.addCookie(mi_cookie); %>`
- **session:** Incluye los datos compartidos entre páginas de la sesión
- **Application:** Los datos de la aplicación.
- **out:** Objeto utilizado para insertar contenido dentro de la página respuesta.  
`<% out.println("Buenos días " + nombre + "<br/>"); %>`
- **config:** Información de la configuración de la página **JSP**.

# Ejemplo (I)

---

```
<%@ page errorPage="PaginaError.jsp" %>
<%@ page import = "java.util.*" %>
<%! private int num_pagina = 1; %>
<% String usuario = (String)request.getParameter("login");
    String clave = (String)request.getParameter("passwd");

    if ( usuario == null || clave == null )
    { %>
<html>
<head><title>Página de inicio de sesión</title></head>
<body>
<center>
Es necesario identificarse para iniciar la sesión:
<form name="formulario" method="POST" action="Ej.jsp">
Usuario:<input type="text" name="login"><br/>
```

# Ejemplo (II)

---

```
Contraseña:<input type="text" name="passwd"><br/>
<input type="submit" value="Comenzar la sesión">
</form></body></html>
<% } else {
/* Sobre estas líneas, faltaría comprobar que el usuario
y la clave son correctos (consultado una BD) */

    session.setAttribute("nombre", usuario);
    session.setMaxInactiveInterval(600);
    String sig_pag = "http://" + request.getServerName()
                    + "/pagina" + num_pagina + ".jsp";
    num_pagina ++;
    response.sendRedirect(sig_pag);
} %>
```

# ¿Qué son?

---

- Los componentes JavaBeans son clases java especiales diseñadas para ser fácilmente reutilizables, para lo que siguen ciertos convenios entre los que destaca:
  - El constructor no tiene argumentos.
  - No suelen permitir el acceso directo a sus variables (propiedades privadas).
  - Suelen existir métodos especiales:
    - Para leer los valores de la propiedad: **get**NomPropiedad()
    - Si el atributo es booleano: **is**NomPropiedad()
    - Para describir sus valores: **set**NomPropiedad(valor)
- Suelen contener lógica de negocio o lógica para el acceso a BD.

## Ejemplo de JavaBean

---

```
package cbns;

public class StringBean {
    private String mensaje;
    public StringBean() {
        mensaje = "Mensaje no fijado";
    }
    public String getMensaje() {
        return (mensaje);
    }
    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }
}
```

- (Se ubicaría en los directorios habituales para servlets)

# Uso dentro de JSP

---

- Se utilizan como componentes reutilizables (en diferentes páginas JSP).
- Dentro de una página JSP se puede crear e inicializar JavaBeans, así como leer y fijar los valores de sus propiedades, sin necesidad de emplear sintaxis Java.
- Ventajas del uso de JavaBeans en páginas JSP:
  - Facilita el uso y comprensión a programadores no habituados a la sintaxis del lenguaje Java (pe. diseñadores de páginas Web)
  - Fuerte separación entre el contenido y la presentación.
  - Facilita la reutilización del código.
  - Facilita compartir objetos entre páginas y peticiones.
  - Facilita la tarea de recoger los parámetros enviados en la petición (cadenas) y guardarlos en las propiedades del objeto (normalmente, de tipos distintos).



# Crear JavaBeans dentro de JSP

- Crear un Bean de una clase determinada, es el equivalente a new en Java.
- Formato para la instanciación de JavaBeans sin utilizar sintaxis Java:
  - **<jsp:useBean id="nomBean" class="nomClase" scope="ámbito"/>**
  - **<jsp:useBean id="nomBean" class="nomClase" scope="ámbito">**

...

**</jsp:useBean>**

**Scope → Request | Page | Session | Application**

- Ejemplo:

```
<jsp:useBean id= "carr" class= "CarritoCompra" scope=
"session" />
```

- Ejemplo equivalente usando sintaxis Java:

```
<%
```

```
CarritoCompra carr = (CarritoCompra)Session.getAttribute("carr");
if (carr==null) { carr = new CarritoCompra();
                  Session.setAttribute("carr");
```

```
} %>
```

# Fijar las propiedades del JavaBean

### ■ Formato:

- Vía scriptlet:  
`<% nomBean.setNomPropiedad(value) %>`
- Vía `jsp:setProperty`:  
`<jsp:setProperty name="nomBean"  
property="nomPropiedad" value="cadena"/>`
  - El nombre del bean “nomBean” debe ser el mismo fijado en el atributo **id** de **jsp:useBean**
  - Debe existir un método llamado **setNomPropiedad** definido dentro de la clase del Bean.
  - Se puede asignar como valor un parámetro de request, utilizando `param="nomParametro"` en vez de `value="cadena"`

### ■ Ejemplo:

```
<jsp:setProperty name="carr" property="cantidad"  
value="<%= cantidadTotal% >"/>
```

Equivalente a: `carr.setCantidad(...)`; Atención! Con la “C” mayúsculas.

```
<jsp:setProperty name=".." property=".." param="canttotal" />
```

Equivale a `<jsp:setProperty name=".." property="..  
value="<%=request.getParameter("canttotal") %>" />`

## Leer las propiedades del JavaBean

---

- Al mismo tiempo que se recupera el valor de la propiedad, se escribe su contenido dentro de la página
- Formato:
  - Vía expresión:  
`<%= nomBean.getNomPropiedad() %>`
  - Vía `jsp:getProperty`:  
`<jsp:getProperty name="nomBean"  
                    property= "nomPropiedad" />`
    - El nombre del bean “nomBean” debe ser el mismo fijado en el atributo **id** de **jsp:useBean**
    - Debe existir un método llamado **get**NomPropiedad definido dentro de la clase del Bean.
- Ejemplo:  
`<jsp:getProperty name="carr" property="cantidad" />`

# Ejemplo

---

```
<html>
....
<jsp:useBean id="entrada" class="Ventas" />
  <jsp:setProperty name="entrada" property = "item"
                    param="id_item" />
  <!-- La anterior sentencia es equivalente a: --%>
  <jsp:setProperty name="entrada" property = "item"
                    value="<%= request.getParameter("id_item")%>" />
  <jsp:setProperty name="entrada" property = "cantidad"
                    param="cantidad" />
.....
```

El Total de las ventas es:

```
  <jsp:getProperty name="entrada" property="total" />
....
</html>
```

# Ejemplo (II)

```
public class Ventas {  
    private String item;  
    private int cantidad;  
    private int total;  
    public Ventas() {  
        item = "";  
        cantidad = 0;  
        total = 0;  
    }  
    public int getCantidad() {  
        return (cantidad);  
    }  
    public void setCantidad(int cantidad) {  
        this.cantidad = cantidad;  
    }  
    public int getTotal() {  
        return (total);  
    }  
    public void setTotal (int total) {  
        this.total = total;  
    }  
}
```

```
    public String getItem() {  
        return (item);  
    }  
    public void setItem(String item)  
        this.item = item;  
    }  
    .....  
}
```

# Introducción

---

- Para invocar los métodos de los JavaBeans aún son necesarios scriptlets.  

```
<jsp:useBean id="miBean" ... />  
<% miBean.miMetodo(x) %>
```
- JSP ofrece la posibilidad de extender acciones a través de etiquetas personalizadas (extendidas)
  - Ofrecen un mecanismo para encapsular funcionalidades reutilizables en diversas páginas JSP
    - Se puede invocar dicha funcionalidad sin necesidad de introducir código Java en la página JSP
  - Permiten separar las funciones del diseñador web (que usa HTML y XML) de las del programador web (que usa Java)
  - Ofrecen mayor potencia que los JavaBeans
- Las etiquetas se empaquetan en una librería (fichero XML con extensión .tld)

## Generación de nuevas etiquetas

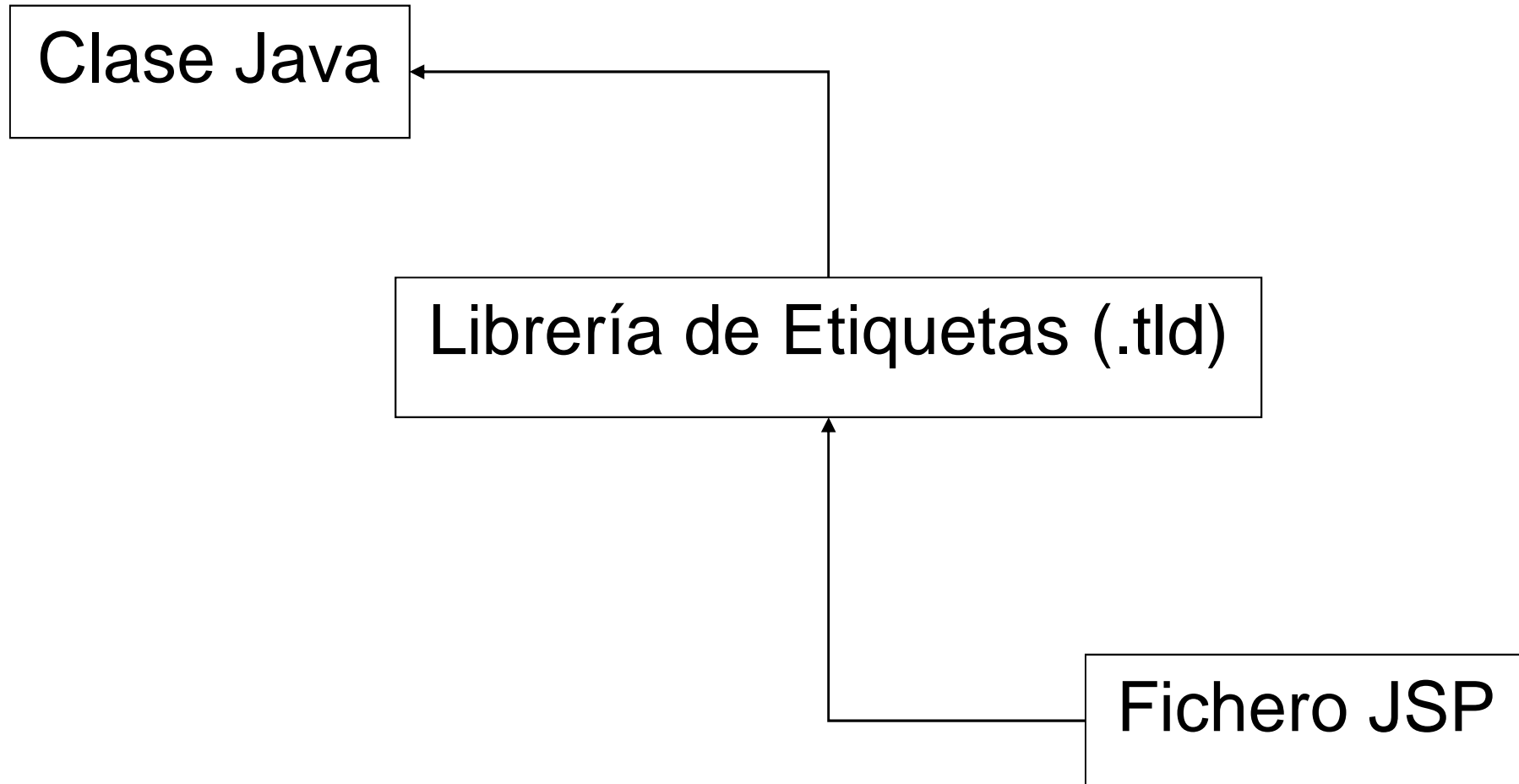
---

- Para generar nuevas etiquetas JSP, se debe seguir los siguientes pasos:
  - Generar una clase Java encargada de realizar la tarea asociada a la etiqueta
    - implementa `javax.servlet.jsp.tagext.Tag`
  - Incluir la etiqueta dentro de la librería de etiquetas (fichero XML de extensión `.tld`) describiendo las propiedades de la etiqueta
  - Utilizar la librería de etiquetas dentro de un fichero JSP
    - Importando la librería de etiquetas

```
<%@ taglib uri=.... Prefix="ist"%>
```
    - Utilizando las etiquetas.

## Generación de nuevas etiquetas (II)

---





## Clase Java asociada a la etiqueta (I)

---

- La clase Java donde se describe la funcionalidad de la etiqueta es una clase derivada (extends) de:
  - TagSupport: Para etiquetas sin contenido o donde el contenido es estático.
  - BodyTagSupport: Para etiquetas con contenido dinámico
- Esta clase constituye un **javaBean** que hereda dos métodos básicos de TagSupport:
  - **doStartTag()** invocado cuando se abre la etiqueta
  - **doEndTag()** invocado cuando se cierra
- Si la etiqueta tiene atributos se definen métodos (set y get) para tratar los valores de los mismos
- Si además tiene cuerpo, hereda de BodyTagSupport dos métodos:
  - **doInitBody()** y **doAfterBody()**

## Clase Java asociada a la etiqueta (II)

---

- Para las etiqueta que no tienen atributos o contenido, sólo es necesario sobrescribir el método `doStartTag`
  - Este devuelve (en general):
    - `SKIP_BODY`: no se procesa el contenido de la etiqueta
    - `EVAL_BODY_INCLUDE`: sí se procesa
- Para las etiquetas con atributos, pe:  
`<prefijo:nombre_etiq atrib1="valor1" atrib2="valor2" ... />`
  - Se definen métodos que se encargan de tratar sus valores.  
`public void setAtrib1(String valor) {...}`

## Clase Java asociada a la etiqueta (III)

---

- Accesible a todos los métodos de la clase, existe predefinida la variable “pageContext”, a partir de la cual se puede obtener objetos:
  - JspWriter (out): `pageContext.getOut()`
  - HttpServletRequest: `pageContext.getRequest()`
  - HttpServletResponse: `pageContext.getResponse()`
  - ServletContext: `pageContext.getServletContext()`
  - HttpSession: `pageContext.getSession()`

### Ejemplo clase

---

```
package p;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
import java.math.*;
public class PrimeTag extends TagSupport {
    protected int length = 50;
    public int doStartTag() {
        try { JspWriter out = pageContext.getOut();
            BigInteger primo = new BigInteger(length,4,new Random());
            out.print(primo);
        } catch (IOException ioe) { }
        return(SKIP_BODY);
    }
    public void setLength(String v_length) {
        try { length = Integer.parseInt(v_length);
        } catch (NumberFormatException nfe) { }
    }
}
```

# Librería de etiquetas

- El fichero TLD asocia la etiqueta a la clase, y define otras propiedades importantes de la etiqueta:

**<tag>**

**<name>** nombre de la etiqueta **</name>**

**<tagclass>** clase asociada a la etiqueta **</tagclass>**

**<bodycontent>** X **</bodycontent>** Donde X es:

- **EMPTY** si la etiqueta no tiene contenido
- **JSP** si la etiqueta contiene código JSP
- **TAGDEPENDENT** si el contenido es procesado por la clase

**<info>** descripción de la etiqueta **</info>**

**<attribute>**

**<name>** nombre del atributo **</name>**

**<required>** false o true **</required>** indica si es opcional

**<rtexprvalue>** false o true **</rtexprvalue>**

indica si el valor puede ser una expresión JSP

**</attribute>**

**</tag>**

# Ejemplo de librería de etiquetas

---

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib ...>
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.2</jspversion>
  <shortname>simple</shortname>
  <uri>http://www.uv.es/ars/simple-tablig</uri>
  <info>...</info>
  <tag>
    <name>primo</name>
    <tagclass>p.PrimeTag</tagclass>
    <bodycontent>EMPTY</bodycontent>
    <info>Primo aleatorio de 50 bits</info>
    <attribute>
      <name>length</name>
      <required>false</required>
    </attribute>
  </tag>
</taglib>
```

# Ejemplo Código JSP

---

```
<html>
<head><title>Números Primos</title></head>
<body>
<h1>Primos de 50 bits</h1>
<%@ taglib uri="http://www.uv.es/ars/libreria_etiq"
    prefix="simple" %>
<ul>
<li><simple:primo length="20"/></li>
<li><simple:primo length="40"/></li>
<li><simple:primo /></li>
</ul>
</body>
</html>
```

## Introducción

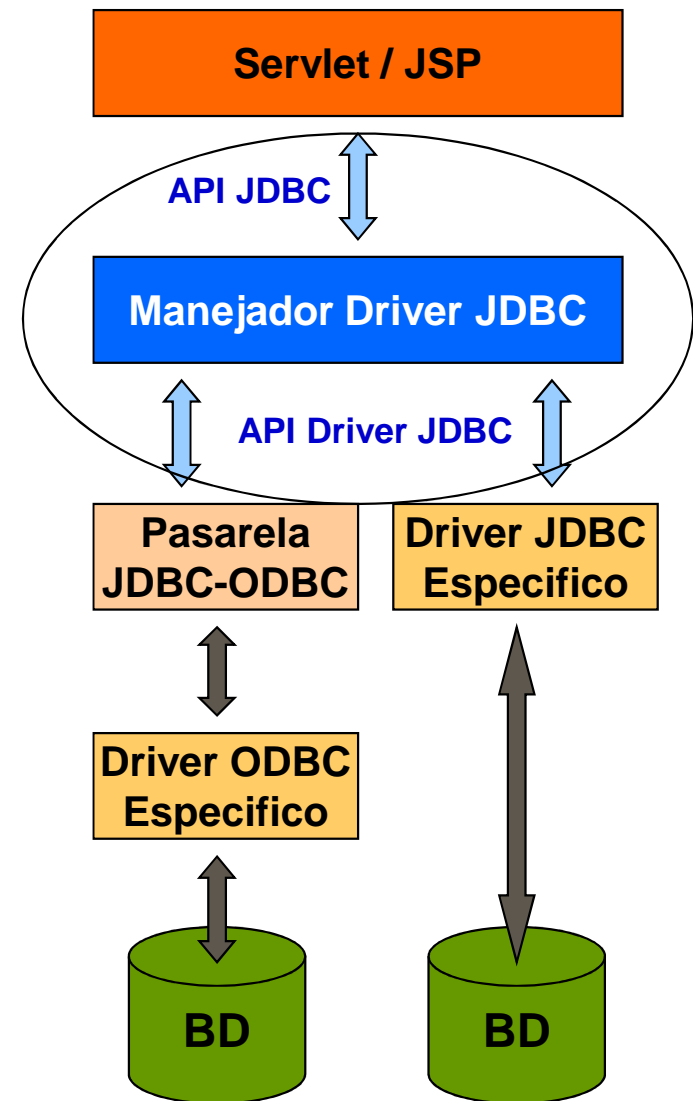
---

- JDBC (Java dataBase Connectivity) proporciona una librería estándar para la conexión de aplicaciones java (web: servlets y JSP) a bases de datos relacionales.
- El API JDBC proporciona un método estándar, independiente de la BD, para:
  - Establecer la conexión con el servidor BD.
  - Realizar consultas y crear tablas con los resultados.
  - Cerrar las conexiones con el servidor.
- Las clases JDBC se encuentran dentro del paquete `java.sql`



## Drivers JDBC

- JDBC permite cambiar el SGBD sin modificar el código del servlet o de la página JSP.
- JDBC consiste en:
  - Un API basado en Java
  - Un manejador del driver JDBC:
    - Se comunica con el driver específico (proporcionado por el vendedor del SGBD) que es el que realiza la conexión real con la BD.



## Pasos básicos (I)

---

### 1. Cargar el driver

- Tenemos que disponer del driver para nuestra B.D. y conocer el nombre de la clase a cargar

```
try {  
    Class.forName("oracle.jdbc.driver.OracleDriver");  
} catch (ClassNotFoundException cnfe) {  
    out.println("<h1>Error al cargar el driver:</h1>" +  
        cnfe);  
}
```

### 2. Definir la conexión URL

- Cada driver utiliza una URL distinta

```
String servidor = "bd.uv.es";  
String NombreBD = "basedatos1";  
int puerto = 1234;  
String url = "jdbc:oracle:thin:@" + servidor +  
    ":" + puerto + ":" + NombreBD;
```

## Pasos básicos (II)

---

### 3. Establecer la conexión:

```
String user = "jsanchez", password = "secreto";  
Connection conex = DriverManager.getConnection  
    (url,user,password);
```

### 4. Realizar la consulta:

```
Statement estado = conex.createStatement();  
String consul = "SELECT col1, col2, col3 FROM tabla1";  
ResultSet resultado = estado.executeQuery(consul);
```

- Para modificar la BD se utiliza `executeUpdate` pasándole una cadena con la operación: UPDATE, DELETE o INSERT.

```
int nfilas = estado.executeUpdate("DELETE FROM  
    tabla1 WHERE ...");
```

## Pasos básicos (III)

---

### 5. Procesar el resultado:

```
out.println("<ul>");
while(resultado.next()) {
    out.println("<li>" + resultado.getString(1) + " " +
        resultado.getString(2) + " " +
        resultado.getString(3) + "</li>");
}
out.println("</ul>");
```

- La clase `ResultSet` proporciona varios métodos `getXxx` que toman como parámetro el número de la columna o el nombre de la columna y devuelven los datos.

### 6. Cerrar la conexión

```
connection.close();
```

# Introducción (I)

---

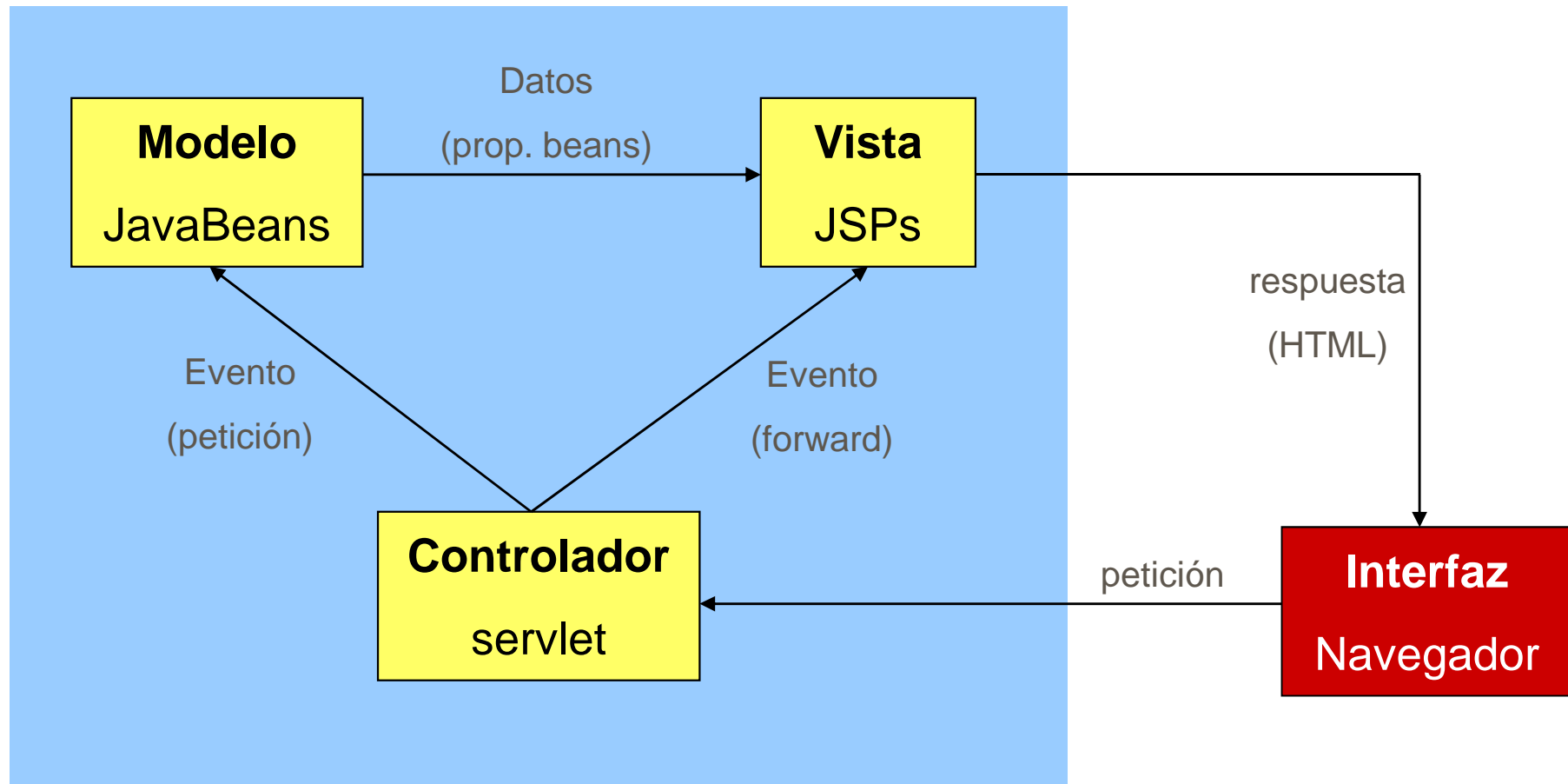
- En las aplicaciones web desarrolladas con Servlets y JSP suele haber una separación clara del código dedicado a la lógica de negocio, al manejo de los datos y a generar la parte del interfaz.
  - En aplicaciones complejas, JSP sólo no suele ser suficiente.
  - Se suele utilizar JSP para desarrollar y mantener la presentación (contenido HTML).
  - El código real se ubica en clases separadas (lógica de negocio):
    - JavaBeans.
    - Etiquetas personalizadas.
    - Servlets.
  - La administración de los datos es gestionada por SGBD (JDBC)

# Introducción (II)

---

- JSP puede ser incluso insuficiente para presentaciones donde los resultados son totalmente diferentes dependiendo de los datos que se reciben.
  - La combinación “JSP + JavaBeans + Etiquetas personalizadas”, aunque muy potente, no puede superar la limitación que impone la secuenciación relativamente fija de los elementos JSP.
  - Solución: usar servlets y JSP.
    - El servlet puede manejar la petición inicial, procesar parcialmente los datos, iniciar los javabeans y pasar a continuación los resultados a un conjunto de páginas JSP.
      - De una sola petición pueden derivar múltiples resultados substancialmente diferentes.
    - Esta aproximación se conoce como **arquitectura MVC** (Model View Controller).
      - MVC modifica el diseño de la aplicación.

# Arquitectura MVC (I)



# Arquitectura MVC (II)

---

- La arquitectura MVC normalmente sigue un conjunto de pautas:
  1. Define javabeans que representan los datos.
  2. Define un servlet que maneja las peticiones.
  3. Invoca el código relacionado con la lógica de negocio y con el manejo de los datos. Los resultados se ubican en los javabeans (del paso 1).
  4. Almacena los javabeans en el contexto adecuado: request, session, application o servlet.
  5. Pasa la petición a una página JSP.
  6. La página JSP accede al javabean para extraer y mostrar los datos.



# Pasando peticiones

- Para que los servlets puedan pasar las peticiones:
  - Sin parámetros: `response.sendRedirect(direccion);`
  - Con parámetros:
    - Se crea un objeto de la clase `RequestDispatcher`.
    - Se utiliza su método `forward` para transferir el control a la URL asociada.

```
public void doGet(...) throws ... {  
    String operacion = request.getParameter("operation");  
    String direccion;  
    if (operacion.equals("order"))  
        direccion = "pedido.jsp";  
    else if (operacion.equals("cancel"))  
        direccion = "cancelacion.jsp";  
    ...  
    RequestDispatcher dispatcher =  
        request.getRequestDispatcher(direccion);  
    dispatcher.forward(request, response);  
}
```

# Proporcionando los datos a la pag. JSP

---

- Lugares donde el servlet almacena los datos que la página JSP utilizará:
  - En el ámbito de la petición:
    - El servlet crea y almacena los datos:

```
UnaClase valor = new UnaClase();  
request.setAttribute("clave", valor);
```
    - La página JSP recupera los datos:

```
<jsp:useBean id="clave" class="UnaClase" scope="request"/>
```
  - En el ámbito de la sesión:
    - El servlet: 

```
session.setAttribute("clave", valor);
```
    - La página JSP:

```
<jsp:useBean id="clave" class="UnaClase" scope="session"/>
```
  - En el ámbito de la aplicación:
    - El servlet: 

```
getServletContext().setAttribute("clave", valor);
```
    - La página JSP:

```
<jsp:useBean id="clave" class="UnaClase" scope="application"/>
```

# URLs relativas en la página destino

---

- El navegador realiza la petición al servlet.
  - No reconoce la existencia de la página JSP a la que el servlet pasa la petición.
- Si la página destino utiliza URLs relativas, pe:  

```
  
<link rel="stylesheet" href="esti.css" type="text/css">  
<a href="bar.jsp">...</a>
```

el navegador los interpreta como relativos al servlet, no a la página JSP.
- Solución: indicar para cada recurso el camino a partir del directorio raíz virtual:  

```
<link rel="stylesheet" href="/camino/esti.css" ...>
```

# Ejemplo: balance banco (servlet)

---

```
public class Balance extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        ClienteBancario cliente =
            ClienteBancario.getCliente(request.getParameter("id"));
        String direccion;
        if (cliente == null)
            direccion = "/WEB-INF/cuenta-banc/ClienteDesconocido.jsp";
        else if (cliente.getBalance() < 0)
            direccion = "/WEB-INF/cuenta-banc/BalanceNeg.jsp";
        request.setAttribute("Moroso", cliente);
        ...
        RequestDispatcher dispatcher =
            request.getRequestDispatcher(direccion);
        dispatcher.forward(request, response);
    }
}
```

# Ejemplo: balance banco (BalanceNeg.jsp)

---

```
...
<body>
<h1> Saldo Negativo!</h1>
<p>

<jsp:useBean id="Moroso" type="Banco.ClienteBancario"
    scope="request" />
Atención,
<jsp:getProperty name="Moroso" property="nombre" />,
Saldo Negativo.
</p><p>
Debe ingresar <jsp:getProperty name="Moroso" property="cantidad"
/>
euros lo antes posible!
</p>
</body>
</html>
```