

# Programación en J2EE

## Componentes Web Servlets.

### Ámbitos de los objetos

#### INDICE:

Programación en J2EE .....	1
Componentes Web Servlets. Ámbitos de los objetos.....	1
1   Objetivos.....	2
2   Arquitectura J2EE: Componente Web Servlet.....	3
3   Estructura de un Servlet.....	5
3.1   Clases componentes de un Servlet.....	5
3.2   Ejemplo desde NetBeans.....	7
4   Pasos de ejecución de un servlet .....	12
4.1   Ciclo de vida de un servlet .....	12
4.2   Ejemplo desde NetBeans.....	14
5   HttpServletRequest y HttpServletResponse.....	16
5.1   HttpServletRequest .....	16
5.2   HttpServletResponse.....	16
5.3   Ejemplo desde NetBeans.....	17
6   Mantener y compartir información.....	19
6.1   Ámbitos de los objetos.....	19
6.2   Ejemplo desde NetBeans.....	22
7   Servlets y formularios.....	25
7.1   Relación entre Servlets y JSPs .....	25
7.2   Ejemplo desde NetBeans.....	26
8   Ejercicios .....	30
8.1   Ejercicio 1(Entregado por el profesor).....	30
8.2   Ejercicio 2 (3 puntos).....	30
8.3   Ejercicio 3 (4 puntos).....	31
8.4   Ejercicio 4 (3 puntos).....	32

## 1 Objetivos

Los objetivos de esta sesión son:

- Estudiar los componentes Web Servlet de la arquitectura J2EE.
- Conocer las posibilidades que nos ofrecen los Servlet y como se complementan con las páginas JSP.
- Estudiar la estructura de un Servlet.
- Capturar y procesar los datos de usuario.
- Estudiar y aplicar los ámbitos de los objetos, sabiendo cuando se aplica cada uno de ellos.
- Estudiar el ciclo de vida de un Servlet comprendiendo como y cuando se ejecuta cada método.
- Aplicar los nuevos conocimientos adquiridos para completar los conocimientos vistos hasta ahora, comenzando ya a crear pequeñas aplicaciones (sin base de datos, que se verá en la siguiente sesión).
- Comprender las ventajas de la separación de generación de contenidos y su presentación que nos ofrecen los servlets.
- Formar al alumno en el uso de NetBeans para la creación de proyectos Web, formularios, JavaBeans, Servlets y páginas JSP.





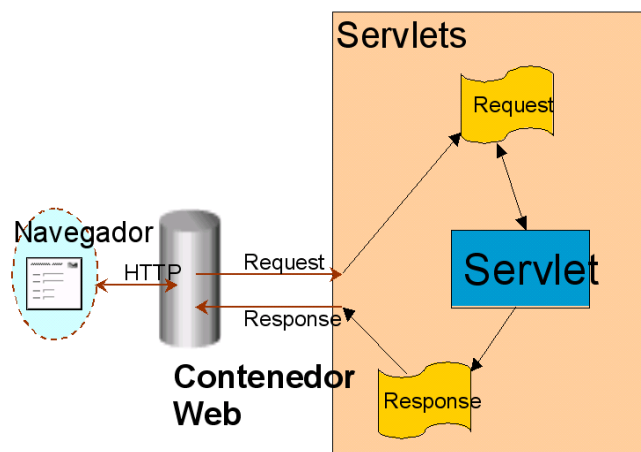
El cliente del servlet será un navegador Web que realiza una petición a través de un formulario Web. El servlet procesará la petición y realizará las operaciones pertinentes (lógica de negocio). Normalmente el servlet devolverá una respuesta que se generará de forma dinámica en formato HTML

En los servlets no existe una clara distinción entre el código HTML (contenido estático) y el código Java (lógica del programa), el contenido HTML que se envía al cliente se suele encontrar como cadenas de texto que se incluyen en sentencias del tipo:

**out.println("<h1>Código HTML</h1>")**

Esto sería un problema dado que mezclaría lógica de negocio con presentación. Veremos como evitar este problema a través de redirigir la salida a páginas JSP. Utilizando también componentes JavaBeans para comunicar los datos a mostrar.

De lo explicado anteriormente se puede deducir que un Servlet proporciona clases e interfaces que nos permiten interactuar con las peticiones y respuestas del protocolo HTTP. Así por ejemplo el interfaz `HttpServletRequest` representa la petición que se ha realizado a un servlet y el interfaz llamado `javax.servlet.http.HttpServletResponse` representa la respuesta que se le va enviar al usuario.



## 3 Estructura de un Servlet

### 3.1 Clases componentes de un Servlet

Todo servlet implementa la interface Servlet del paquete **javax.servlet**. Esto lo puede lograr de dos formas diferentes:

1. heredando de **GenericServlet** si admite peticiones no basadas en ningún protocolo especial
2. o bien, heredando de **HttpServlet** (es subclase de la anterior) si admite peticiones de clientes basadas en el protocolo http. En nuestro curso usaremos este tipo de Servlets

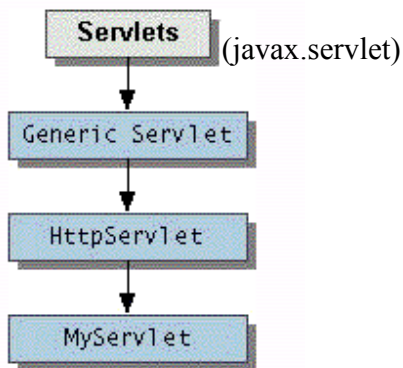
La declaración de la clase asociada a un servlet http genérico sería:

```

1 import javax.servlet.*;
2 import javax.servlet.http.*;
3
4 public class MiServletHttp extends HttpServlet{
5     .
6     .
7     .

```

En la siguiente figura podemos ver que la interfaz definida en javax.servlet.Servlet es implementada por la clase GenericServlet, la cual a su vez es extendida por HttpServlet. Nuestros Servlets extenderán de HttpServlet:



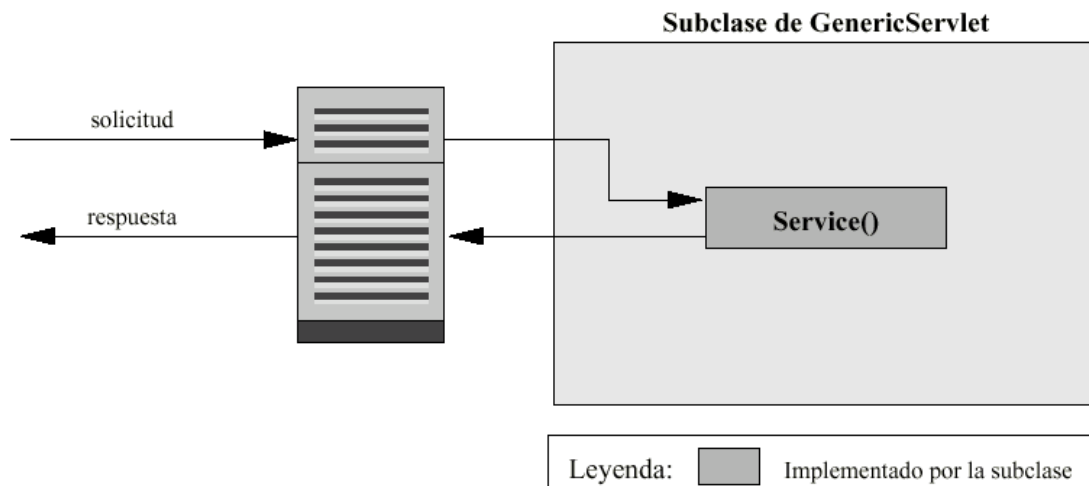
Como nuestros Servlets extienden la clase HttpServlet todo servlet va a poder usar directamente los métodos de esta clase, los de su superclase GenericServlet, además de los de la propia clase.

La interface Servlet declara los métodos del ciclo de vida de un servlet, que son:

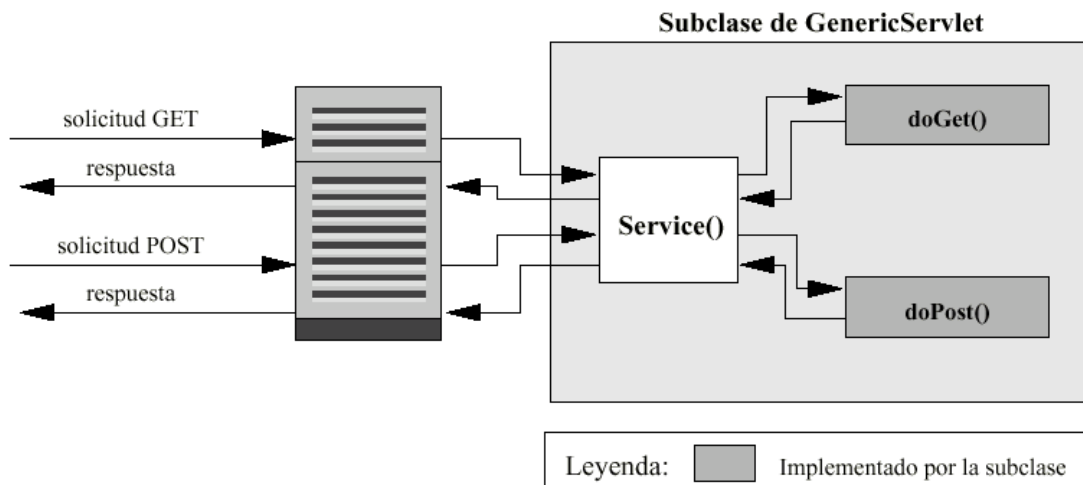
- **void init (ServletConfig config):** es **invocado, una sola vez**, por el contenedor del servidor J2EE compatible donde se hospeda el servlet y se emplea para inicializarlo. Se ejecuta **cuando se realiza la primera petición**.
- **void destroy():** es invocado por el contenedor antes de que el servlet se descargue de memoria y deje de prestar servicio.

- **void service(ServletRequest request, ServletResponse reponse):** es invocado por el contenedor para procesar las peticiones. Sus argumentos son instancias de las interfaces **javax.servlet.ServletRequest** y **javax.servlet.ServletResponse** que modelan, respectivamente, la petición del cliente y la respuesta del servlet. En un servlet de tipo Http (los que usaremos en nuestro curso), este método suele sustituirse por los métodos de **javax.servlet.http.HttpServlet**.
  - **void doPost(HttpServletRequest request, HttpServletResponse response)** para gestionar peticiones post
  - **void doGet(HttpServletRequest request, HttpServletResponse response)** para gestionar peticiones get

Es decir, en un servlet que extienda de GenericServlet ejecutaría el método **service**(deberíamos implementarlo) ante la petición del cliente.

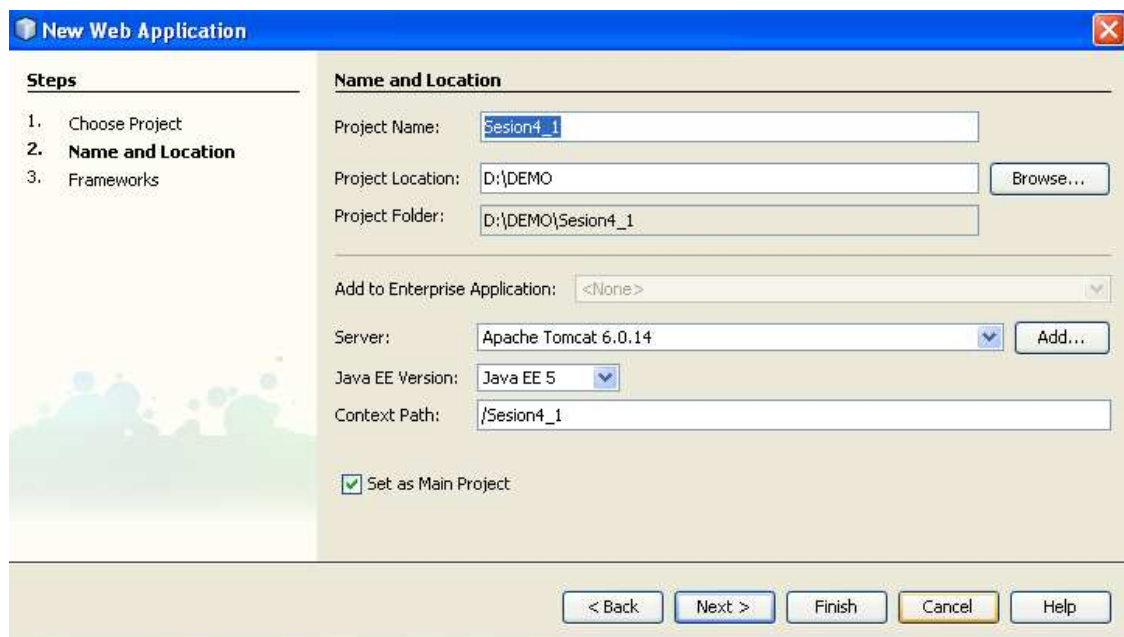


Por otro lado en los servlets que extiendan **HttpServlet** tendríamos que implementar el método **doGet** para procesar peticiones GET y **doPost** para procesar peticiones de tipo POST. Recordemos que el atributo **Method** de los formularios permite indicar el tipo de petición (GET o POST)

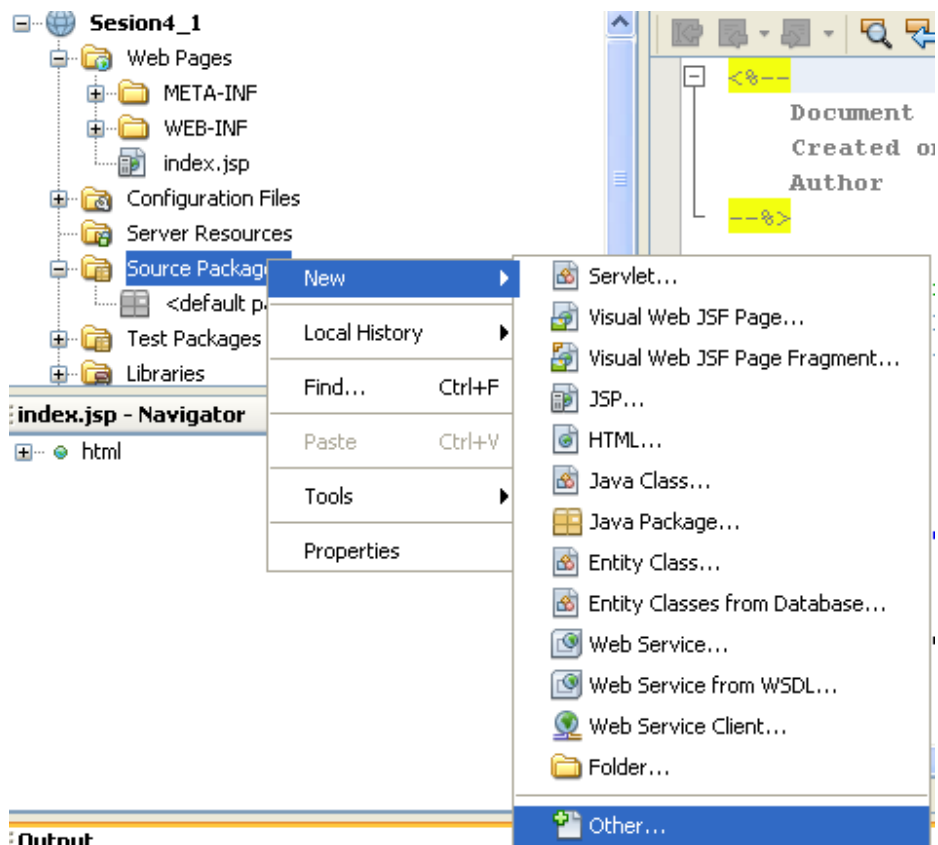


### 3.2 Ejemplo desde NetBeans

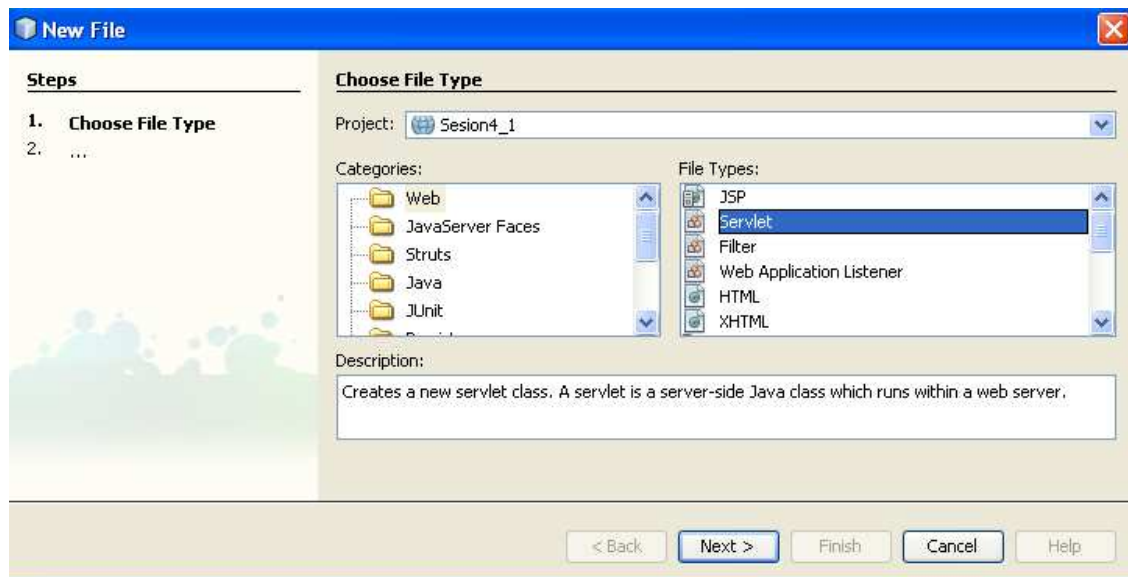
De igual modo que hemos hecho en anteriores sesiones creemos un nuevo proyecto Web llamado Sesión4\_1 desde NetBeans.



En este ejemplo crearemos nuestro primer Servlet. Para crearlo pulsaremos botón derecho sobre Paquetes de origen y seleccionaremos crear un nuevo archivo.

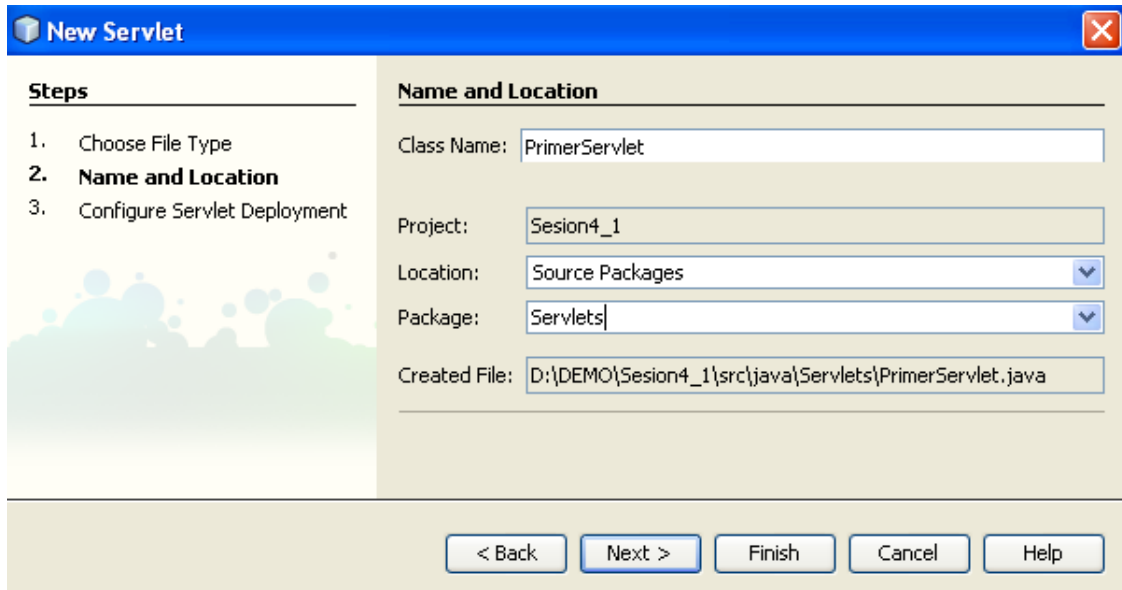


A continuación seleccionaremos el tipo de archivo Servlet.



Indicaremos el nombre del Servlet, en nuestro caso PrimerServlet, y el paquete donde queremos que se cree, en nuestro caso Servlets.





**New Servlet**

**Steps**

1. Choose File Type
2. **Name and Location**
3. Configure Servlet Deployment

**Name and Location**

Class Name:

Project:

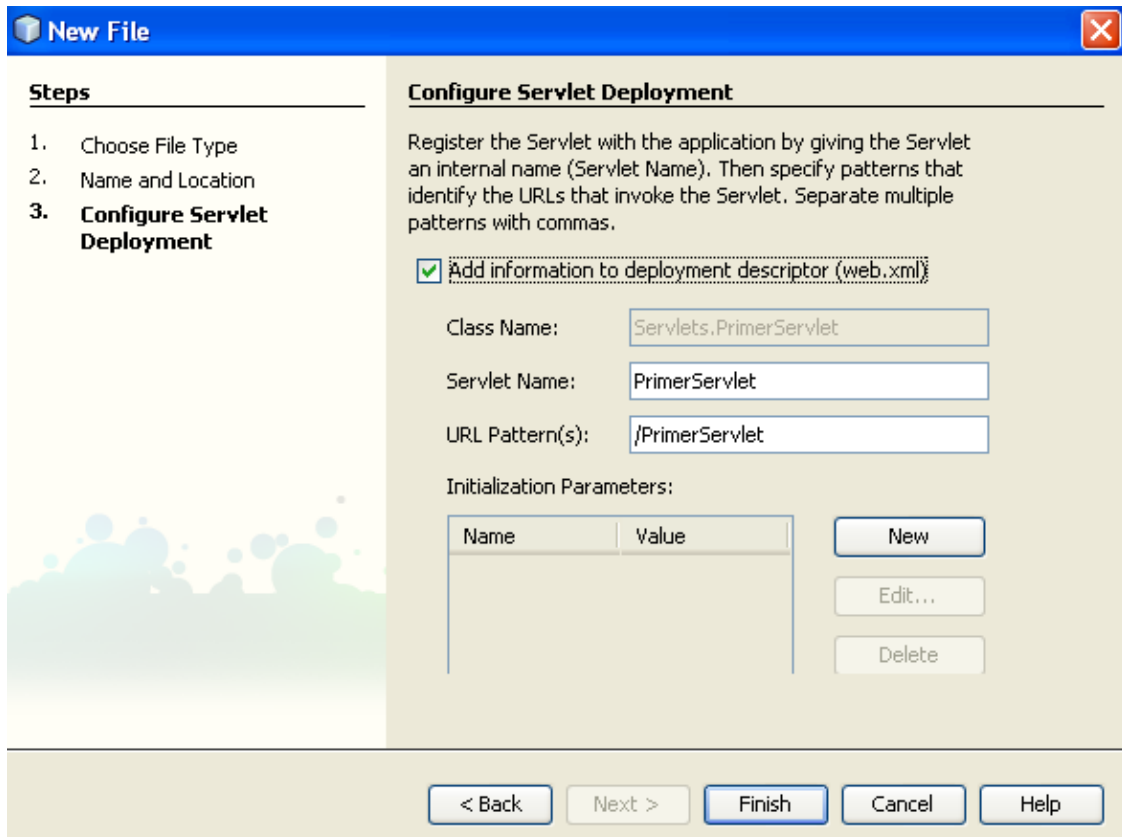
Location:

Package:

Created File:

< Back   Next >   Finish   Cancel   Help

Dejamos la siguiente pantalla de configuración como indica NetBeans.



**New File**

**Steps**

1. Choose File Type
2. Name and Location
3. **Configure Servlet Deployment**

**Configure Servlet Deployment**

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ Add information to deployment descriptor (web.xml)

Class Name:

Servlet Name:

URL Pattern(s):

Initialization Parameters:

Name	Value

New   Edit...   Delete

< Back   Next >   Finish   Cancel   Help

De este modo tendremos un primer Servlet creado, a falta de completar el método **processRequest** que NetBeans nos propone y que no es más que un método llamado

desde los métodos doGet y doPost para tener la misma respuesta ante ambos tipos de formularios.

```
package Servlets;

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class PrimerServlet extends HttpServlet {

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

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


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

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

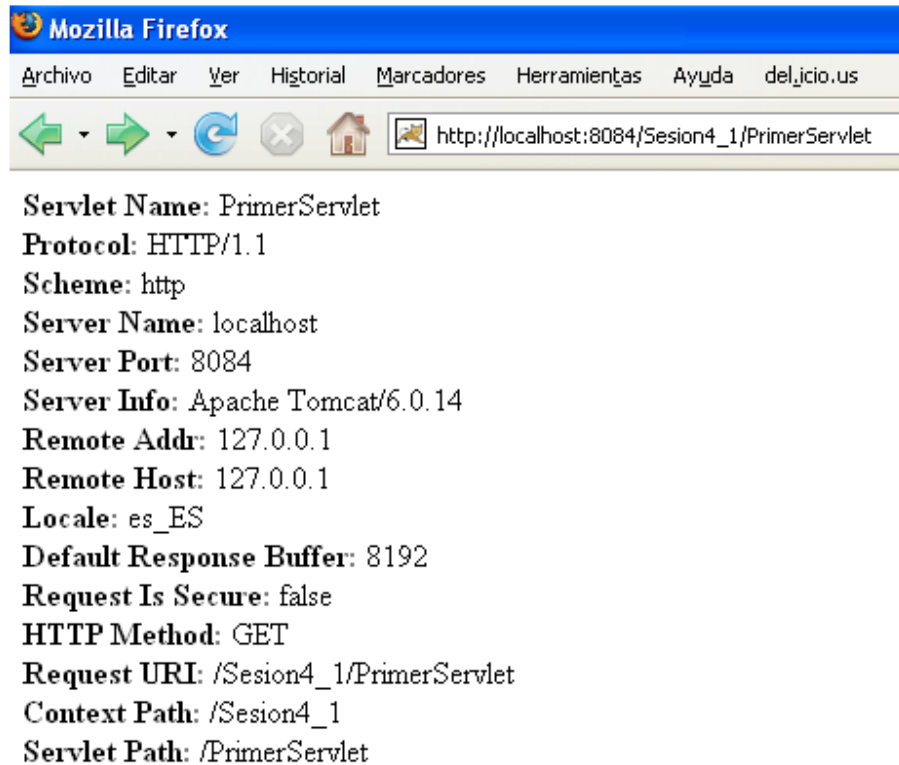
Completaremos el método processRequest con el siguiente código

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<b>Servlet Name:</b> " + getServletName());
        out.println("<br><b>Protocol: </b>" + request.getProtocol().trim());
        out.println("<br><b>Scheme: </b>" + request.getScheme());
        out.println("<br><b>Server Name: </b>" + request.getServerName());
        out.println("<br><b>Server Port: </b>" + request.getServerPort());
        out.println("<br><b>Server Info: </b>" + getServletContext().getServerInfo());
        out.println("<br><b>Remote Addr: </b>" + request.getRemoteAddr());
        out.println("<br><b>Remote Host: </b>" + request.getRemoteHost());
        out.println("<br><b>Locale: </b>" + request.getLocale());
        out.println("<br><b>Default Response Buffer: </b>" + response.getBufferSize());
        out.println("<br><b>Request Is Secure: </b>" + request.isSecure());
        out.println("<br><b>HTTP Method: </b>" + request.getMethod());
        out.println("<br><b>Request URI: </b>" + request.getRequestURI());
        out.println("<br><b>Context Path: </b>" + request.getContextPath());
        out.println("<br><b>Servlet Path: </b>" + request.getServletPath());
    } finally {
        out.close();
    }
}
```

Este código nos muestra información en pantalla tanto del cliente como del servidor.

Para probarlo pulsaremos  y escribiremos en el navegador la siguiente URL  
[http://localhost:8084/Sesion4\\_1/PrimerServlet](http://localhost:8084/Sesion4_1/PrimerServlet)

De este modo, veremos el siguiente resultado:



## 4 Pasos de ejecución de un servlet

### 4.1 Ciclo de vida de un servlet

En este apartado vamos a abordar la forma en la que los servlets son creados y destruidos por el contenedor Web (también llamado en este caso contenedor de servlets). Es el contenedor Web quien controla el ciclo de vida del Servlet.

Cuando se crea un servlet, existirá **una única instancia de este servlet**, que por cada petición que le realicen los usuarios creará un hilo de ejecución (thread) en el que se tratará el método service que invocará el método doGet() o doPost() según el tipo de petición.

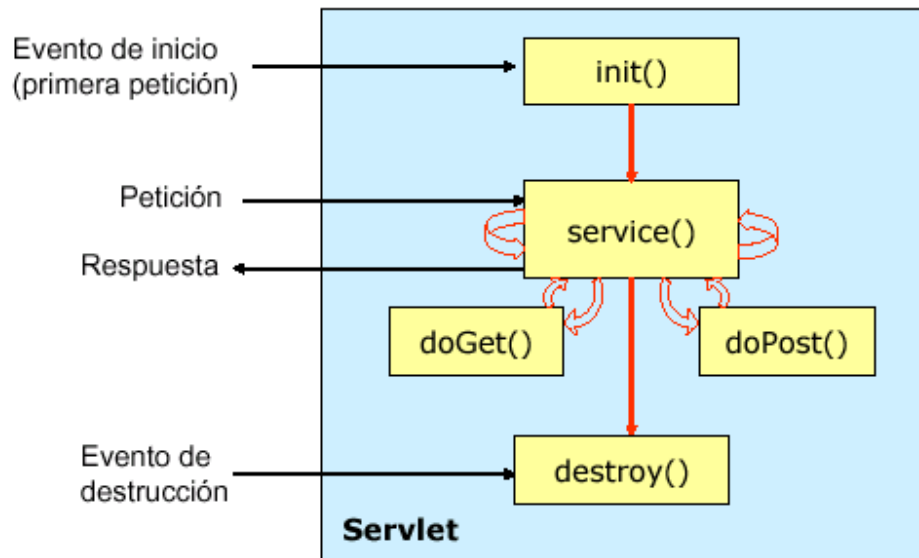
A continuación vamos a tratar todo este proceso en detalle:

Cuando se inicia el Servidor Web este lee los descriptores de despliegue de los Servlets que tiene instalados. De este modo el Servidor Web carga la clase asociada al servlet, instanciándola invocando a su constructor sin argumentos por defecto y cargándola en memoria. El objeto javax.servlet.ServletConfig es creado por el contenedor durante la carga en memoria y “absorbe” multitud de información sobre el servlet, previa lectura del descriptor de despliegue.

Cuando se realiza la primera petición al servlet este es inicializado. Esta **inicialización** es única e implica la **ejecución, por parte del contenedor, del método init(ServletConfig config)** del servlet. Por lo tanto este método contendrá el código de inicialización del servlet. Solo es llamado en esta primera petición.

Finalizada la inicialización, el servlet ya está en disposición de procesar las peticiones y generar una respuesta a las mismas. De este modo, cada petición realizada por un usuario sobre el servlet se traduce en un nuevo hilo de ejecución (thread) que realiza una llamada al método service(). Múltiples peticiones concurrentes normalmente generan múltiples hilos de ejecución que llaman de forma simultánea al método service().

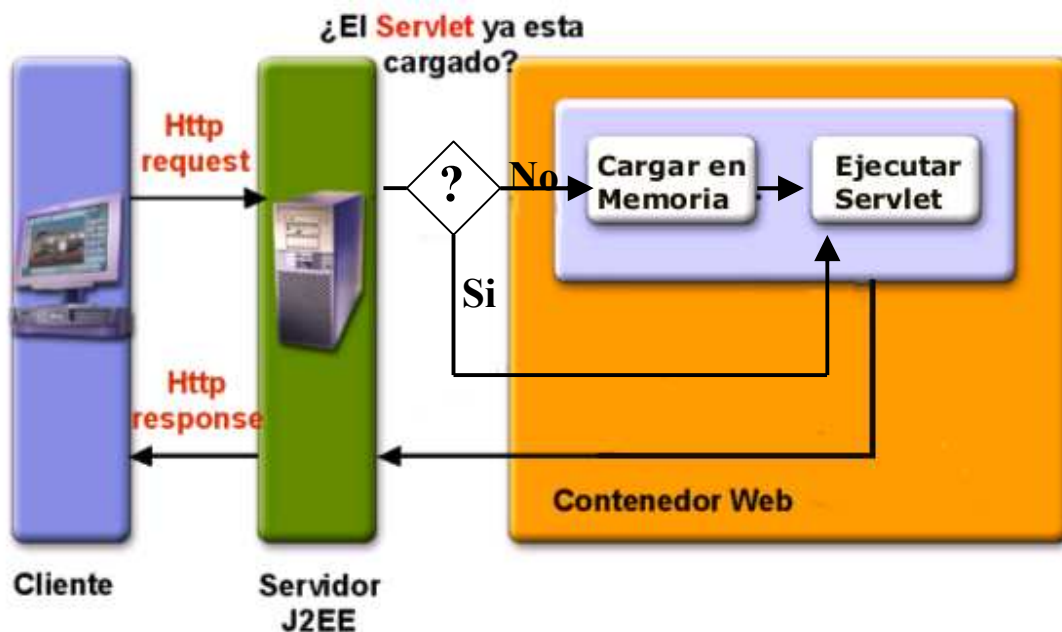
El método service() invocará los métodos doGet() o doPost() dependiendo del tipo de petición HTTP recibida. Los objetos HttpServletRequest y HttpServletResponse asociados a los argumentos de estos métodos son creados también por el contenedor.



Finalmente, si se tumba el servidor, o se sobrepasa un tiempo límite de inactividad del servlet, o si el servidor está trabajando con insuficiente memoria, el contenedor podría eliminar el objeto servlet de la memoria. Para esto el contenedor ejecutaría el método `destroy` del Servlet.

Como se puede ver el ciclo de vida de un servlet sigue el siguiente esquema:

- Ejecución del método `init()` para inicialización del servlet.
- Sucesivas ejecuciones del método `service()` en distintos hilos de ejecución, que resultan en una llamada a un método `doXXX`.
- Ejecución del método `destroy()` para realizar labores de liberación de recursos.



## 4.2 Ejemplo desde NetBeans

Añadiremos un nuevo Servlet llamado ServletCicloVida, al proyecto Sesión4\_1 creado anteriormente, donde insertaremos el siguiente código.

```
package Servlets;
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
/**...*/
public class ServletCicloVida extends HttpServlet {
    int contadorAccesos=0;
    @Override
    public void init(ServletConfig cfg) throws ServletException
    {
        contadorAccesos=5;
        super.init(cfg);
    }
    /**...*/
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            if (contadorAccesos!=0) {
                out.println("Quedan "+contadorAccesos+" accesos antes de " +
                    "dejar de mostrar este mensaje");
                contadorAccesos--;
            }
        } finally {
            out.close();
        }
    }
    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

El propósito de este código es observar como la variable *contadorAccesos* es una variable de la instancia del Servlet en memoria, y por tanto su valor es compartido entre todas las ejecuciones. De este modo, se inicializa en `init` y en `service` (en este caso `processRequest`) se modifica su valor.

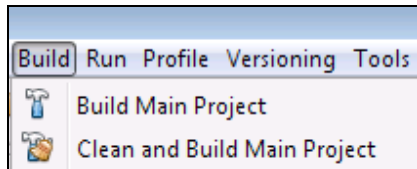
Obsérvese en el código como aunque se sobrescribe el método `init` se sigue llamando al método `init` definido por defecto en la clase `HttpServlet` a través de `super.init(cfg)`. Esto permite que se inicialice correctamente el contexto del servlet.


Podemos probar el Servlet mediante  y escribir en el navegador la siguiente URL [http://localhost:8084/Sesion4\\_1/ServletCicloVida](http://localhost:8084/Sesion4_1/ServletCicloVida)



Podemos comprobar que si accedemos varias veces se van restando los accesos. Lo mismo ocurre si cerramos el navegador y volvemos a acceder. Una vez alcanzados los 5 accesos no mostrará ningún mensaje por pantalla hasta que el contenedor de servlets no decida llamar a destroy. En nuestro caso, para no tener que esperar lo reinicializaremos a través de pulsar:

- Menu Build / Clean and Build Main Project



- y de nuevo  (provocando que el Servlet se descargue de memoria y se vuelva a realizar el despliegue en el contenedor de Servlets)

## 5 **HttpServletRequest** y **HttpServletResponse**

Los métodos de la clase **HttpServlet** que manejan peticiones de cliente (estos métodos son `service`, `doGet`, `doPost`) toman dos argumentos.

1. Un objeto **HttpServletRequest**, que encapsula los datos desde el cliente.
2. Un objeto **HttpServletResponse**, que encapsula la respuesta hacia el cliente.

### 5.1 **HttpServletRequest**

Un objeto **HttpServletRequest** proporciona acceso a los datos de cabecera del protocolo HTTP con el que se ha realizado la petición. Además **HttpServletRequest** también permite obtener los argumentos que el cliente envía como parte de la petición (a través de un formulario).

Para acceder a los datos del cliente tenemos los siguientes métodos:

- **getParameter(\_name)**  
Devuelve el valor del parámetro con nombre `_name`. Si nuestro parámetro pudiera tener más de un valor, deberíamos utilizar **getParameterValues** en su lugar
- **getParameterValues (\_name)**  
Devuelve un array de valores del parámetro con nombre `_name`.
- **getParameterNames**  
Proporciona los nombres de los parámetros que vienen en la petición.

Más adelante veremos el uso que se puede hacer de este objeto para almacenar datos y enviarlos a una página JSP para que los procese.

### 5.2 **HttpServletResponse**

Un objeto **HttpServletResponse** proporciona dos métodos para devolver datos al usuario.



- El método **getWriter** devuelve un objeto de tipo **Writer**
- El método **getOutputStream** devuelve un objeto de tipo **ServletOutputStream**

Se utiliza el método **getWriter** para devolver datos en formato texto al usuario y el método **getOutputStream** para devolver datos binarios (ficheros).

Debemos seleccionar la cabecera de datos HTTP antes de acceder a **Writer** o a **OutputStream**, para ello usaremos el método **setContentType** de la clase **HttpServletResponse** que permite seleccionar el tipo del contenido.

El problema de usar los métodos anteriores de **HttpServletResponse** es que no se diferencia entre la generación de datos y presentación. Esta es la razón por la que usaremos un método llamado **forward** mediante el cual se pasa la respuesta a la página JSP que se desee, evitando que el Servlet deba escribirla.

Este método **forward** pertenece al objeto **RequestDispatcher** que se obtiene del contexto del Servlet. Esto es, estando en un servlet haremos:

```
RequestDispatcher rd =  
this.getServletContext().getRequestDispatcher("url_relativa");  
rd.forward(request, response);
```


donde **url\_relativa** es la dirección de la página JSP a la que redirigimos la salida.

Veremos un ejemplo del uso de **forward** más adelante cuando tratemos la relación entre Servlets y formularios.

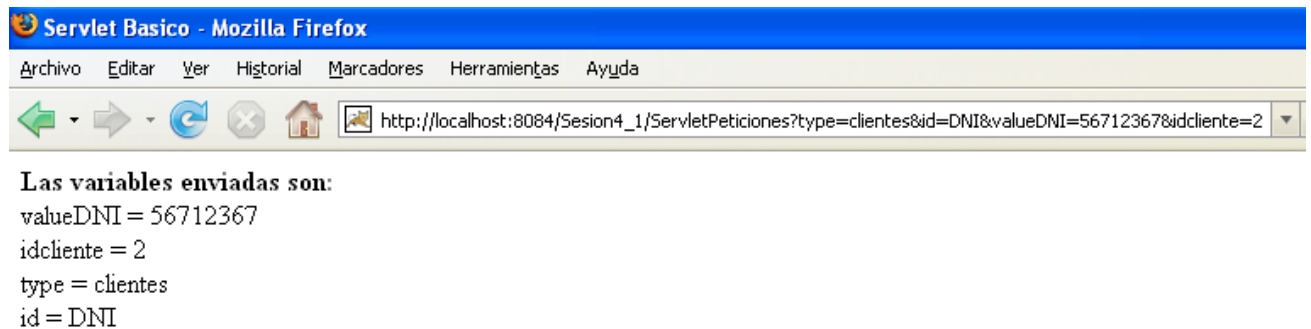
### **5.3 Ejemplo desde NetBeans**

En este ejemplo probaremos la captura de datos enviados por los clientes a través del uso de los métodos del objeto **HttpServletRequest**. Para esto, añadiremos un nuevo Servlet llamado **ServletPeticones**, al proyecto **Sesion4\_1** creado anteriormente, donde insertaremos el siguiente código en el método **processRequest**:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<html>");
        out.println("<head><title>Servlet Basico</title></head>");
        out.println("<body>");
        out.println("<b>Las variables enviadas son: </b> <br>");
        // Imprime en pantalla las variables indicadas en la URL
        java.util.Enumeration _enum=request.getParameterNames();
        String _key="";
        String _value="";
        while (_enum.hasMoreElements()) {
            _key = (String) _enum.nextElement();
            _value = (String) request.getParameter(_key);
            out.println("    " + _key + " = " + _value+"<br>");
        }
        out.println("</body></html>");
    } finally {
        out.close();
    }
}
```

Para probarlo pulsaremos  y escribiremos en el navegador la siguiente URL  
[http://localhost:8084/Sesion4\\_1/ServletPeticiones?type=clientes&id=DNI&valueDNI=56712367&idcliente=2](http://localhost:8084/Sesion4_1/ServletPeticiones?type=clientes&id=DNI&valueDNI=56712367&idcliente=2)

De este modo, veremos el siguiente resultado:



## 6 Mantener y compartir información

### 6.1 Ámbitos de los objetos

Los objetos definidos en un ámbito (también llamado scope) se usan para mantener en memoria esos objetos (en nuestro caso serán Javabeans) disponibles según el ámbito. Evidentemente una vez son descargados de memoria ya no serán accesibles.

Hay cuatro ámbitos de objetos diferentes:

- Application  
cuyo ámbito es la aplicación Web. Establece objetos en memoria que serán accesibles siempre por cualquier elemento de la aplicación J2EE.  
Se accede a través de

```
ServletContext scopeApplication = this.getServletContext();
scopeApplication.setAttribute(_name,_value);
```

donde **\_name** es el nombre con el que se almacenará en la memoria y **\_value** el objeto que almacenamos

- Session  
establece objetos en memoria que se mantendrán durante toda la sesión (navegación del cliente) en que fueron creadas. Se accede a través de

```
HttpSession scopeSession = request.getSession();
scopeSession.setAttribute(_name,_value);
```

donde **\_name** es el nombre con el que se almacenará en la memoria y **\_value** el objeto que almacenamos

- Request  
establece objetos en memoria que se mantendrán desde el inicio de la petición de una página hasta su procesamiento completo. Se accede a través de

```
request.setAttribute(_name,_value);
```

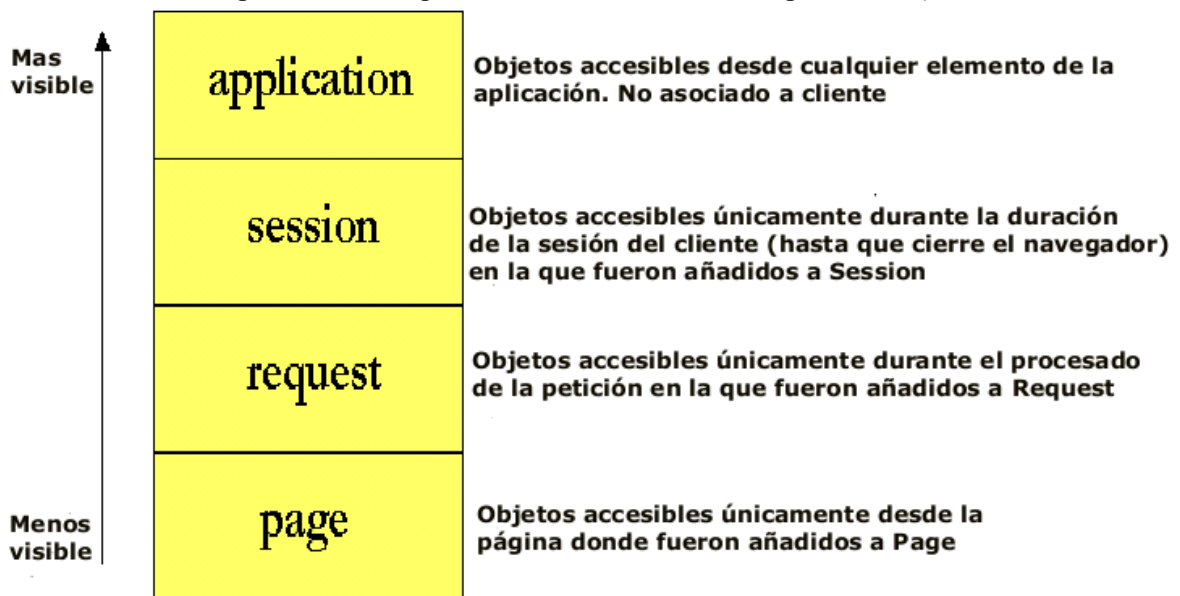
donde **\_name** es el nombre con el que se almacenará en la memoria y **\_value** el objeto que almacenamos

- Page  
establece objetos que se mantendrán únicamente en la página que fueron creados. Este ámbito solo tiene vigencia en las páginas JSP. Se accede a través de

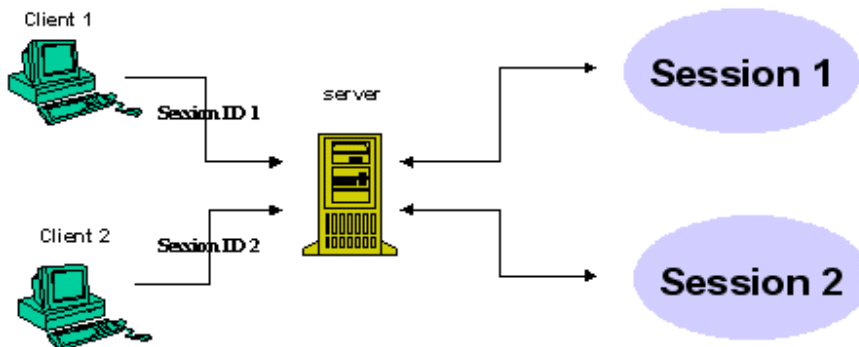
```
<jsp:useBean id="name" class="package.class" scope=="page" />
```

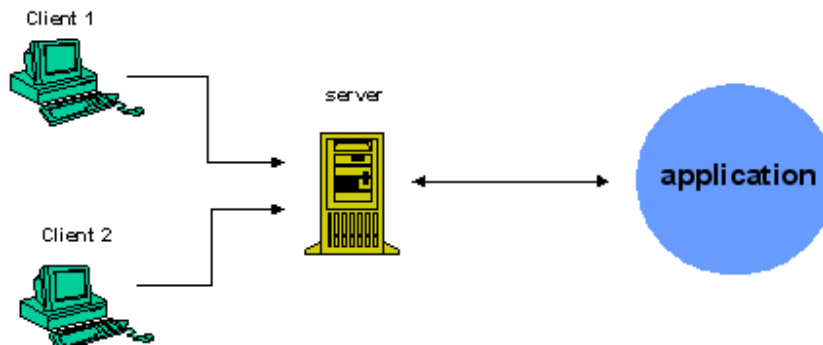
que ya se explicó en la anterior sesión

Los ámbitos Application, Session y Request permiten compartir objetos JavaBean entre Servlets y páginas JSP. De este modo, una vez el Servlet tenga un JavaBean con los datos preparados para presentar lo almacenará en uno de esos tres ámbito mediante el método **setAttribute** correspondiente y pedirá mediante **forward** (lo veremos más adelante) que una página JSP presente esos datos de forma adecuada (recuperará los datos mediante el **getAttribute** correspondiente). Como podemos observar hemos logrado separar la obtención de datos (JavaBean) por los Servlet de su presentación en las páginas JSP (esto es conocido como patrón MVC, que estudiaremos en sesiones posteriores).

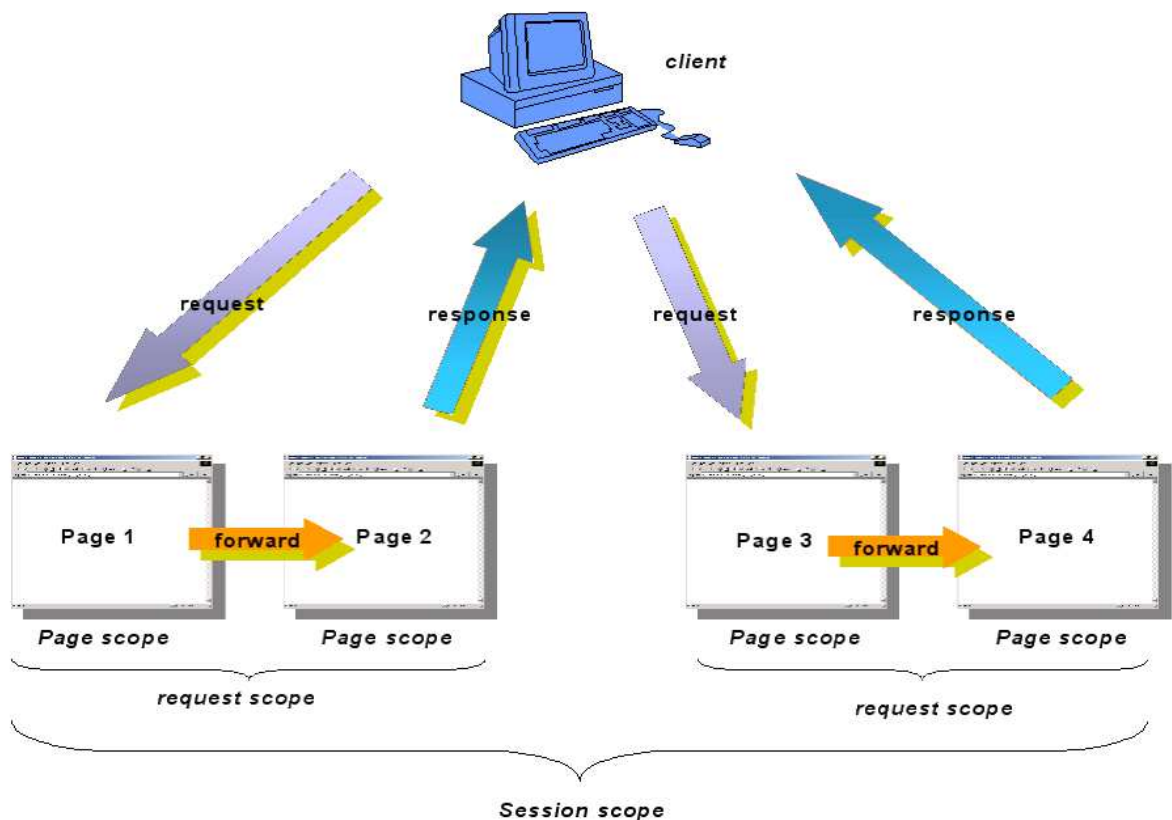


La siguiente figura muestra la diferencia entre ámbitos de sesión y de aplicación. Un ámbito de sesión corresponde a un cliente en particular mientras que un ámbito de aplicación es compartido por todos los clientes que acceden a una misma aplicación.





La siguiente figura muestra la relación entre los ámbitos **session**, **request** y **page**. Como podemos observar existe un ámbito **page** por cada página JSP. Por otro lado el ámbito de **request** comprende todas las páginas, Servlets y demás elementos involucrados en el proceso de ir de una página a otra. Por último el ámbito de **session** incluye todas las peticiones y respuestas que un cliente en particular realiza.



Como se ve en la figura anterior, el ámbito Session permite mantener el estado de un cliente a lo largo de su navegación. Otra posibilidad sería por medio de **cookies**, las cuales son un mecanismo que el servlet utiliza para mantener en el cliente una pequeña cantidad de información asociada con el usuario. Esta tercera forma no la estudiaremos debido a que obligaría a activar el uso de cookies por parte del navegador del cliente.

## 6.2 Ejemplo desde NetBeans

Para comprobar lo explicado sobre los ámbitos de los objetos vamos a realizar una prueba usando tres servlets:

- Desde el primero (llamado ServletScope1) se establecen los parámetros y pasa la petición al segundo Servlet
- Desde el segundo (llamado ServletScope2) se acceden a los parámetros y se crea un enlace al tercer servlet
- Desde el tercero (llamado ServletScope3) únicamente se accede a los parámetros

Se comprobará que el segundo Servlet ve todos los parámetros, mientras que al tercero nunca llega el parámetro de Request. Se puede comprobar que si se cierra el navegador y se accede directamente al segundo o al tercer Servlet seguirá vigente únicamente el atributo del ámbito de aplicación. Pasemos a implementarlo:

Añadiremos un nuevo Servlet llamado ServletScope1, al proyecto Sesión4\_1 creado anteriormente, donde insertaremos el siguiente código en el método **processRequest**:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        request.setAttribute("atributoRequest", "Prueba de atributo request");
        request.getSession().setAttribute("atributoSession", "Prueba de atributo en sesion");
        this.getServletContext().setAttribute("atributoApplication", "Prueba de atributo en Application");
        RequestDispatcher rd = this.getServletContext().getRequestDispatcher("/ServletScope2");
        rd.forward(request, response);
    } finally {
        out.close();
    }
}
```

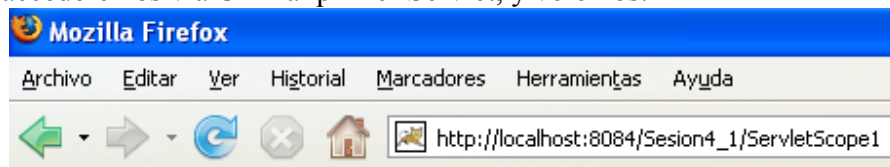
Crearemos un segundo Servlet llamado ServletScope2 donde insertaremos el siguiente código en el método **processRequest**:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<b>Los parametros establecidos son:</b><br>");
        out.println("Request:"+(String) request.getAttribute("atributoRequest")+"<br>");
        out.println("Session:"+(String) request.getSession().getAttribute("atributoSession")+"<br>");
        out.println("Application:"+(String) this.getServletContext().getAttribute("atributoApplication")+"<br>");
        out.println("Pulse <a href=\"./ServletScope3\">aquí</a> para pasar a verlos desde la siguiente página");
    } finally {
        out.close();
    }
}
```

Por último, crearemos un tercer Servlet llamado ServletScope3 donde insertaremos el siguiente código en el método **processRequest**:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<b>Los parametros establecidos son:</b><br>");
        out.println("Request: "+(String) request.getAttribute("atributoRequest")+"<br>");
        out.println("Session: "+(String) request.getSession().getAttribute("atributoSession")+"<br>");
        out.println("Application: "+(String) this.getServletContext().getAttribute("atributoApplication")+"<br>");
    } finally {
        out.close();
    }
}
```

Una vez hemos creado estos 3 Servlets, pasemos a realizar las pruebas. Primero, accederemos vía URL al primer Servlet, y veremos:



#### Los parametros establecidos son:

Request:Prueba de atributo request

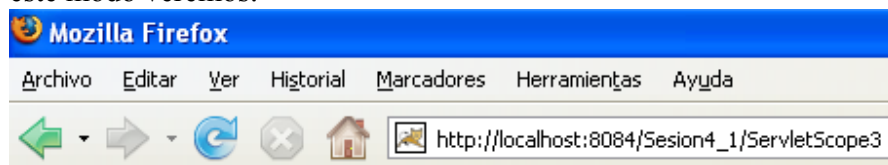
Session:Prueba de atributo en sesion

Application:Prueba de atributo en Application

Pulse [aquí](#) para pasar a verlos desde la siguiente página

Lo que ha ocurrido ha sido que se han establecido los parámetros en ServletScope1 y se ha pasado la petición internamente al segundo Servlet.

A continuación pulsamos en la palabra “**aquí**” para navegar hacia el tercer Servlet. De este modo veremos:



#### Los parametros establecidos son:

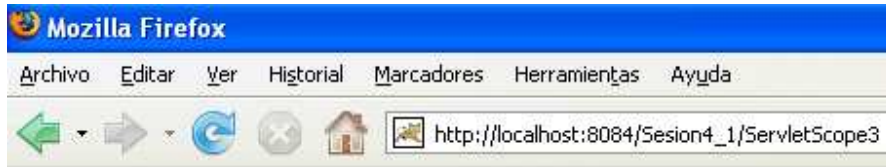
Request:null

Session:Prueba de atributo en sesion

Application:Prueba de atributo en Application

Como se puede comprobar el parámetro Request es válido durante el procesado de la petición. Más allá este valor se pierde, como podemos comprobar en la pantalla. Sin embargo, los atributos de Session y Application tienen una vigencia más amplia.

La última prueba que haremos es cerrar el navegador y acceder directamente a la URL del tercer servlet. Este es [http://localhost:8084/Sesion4\\_1/ServletScope3](http://localhost:8084/Sesion4_1/ServletScope3). De este modo veremos:



**Los parametros establecidos son:**

Request:null

Session:null

Application:Prueba de atributo en Application

En la figura anterior se puede comprobar como el parámetro Session es válido hasta que el usuario cierra el navegador. Sin embargo, el atributo de Application tiene una vigencia más amplia.

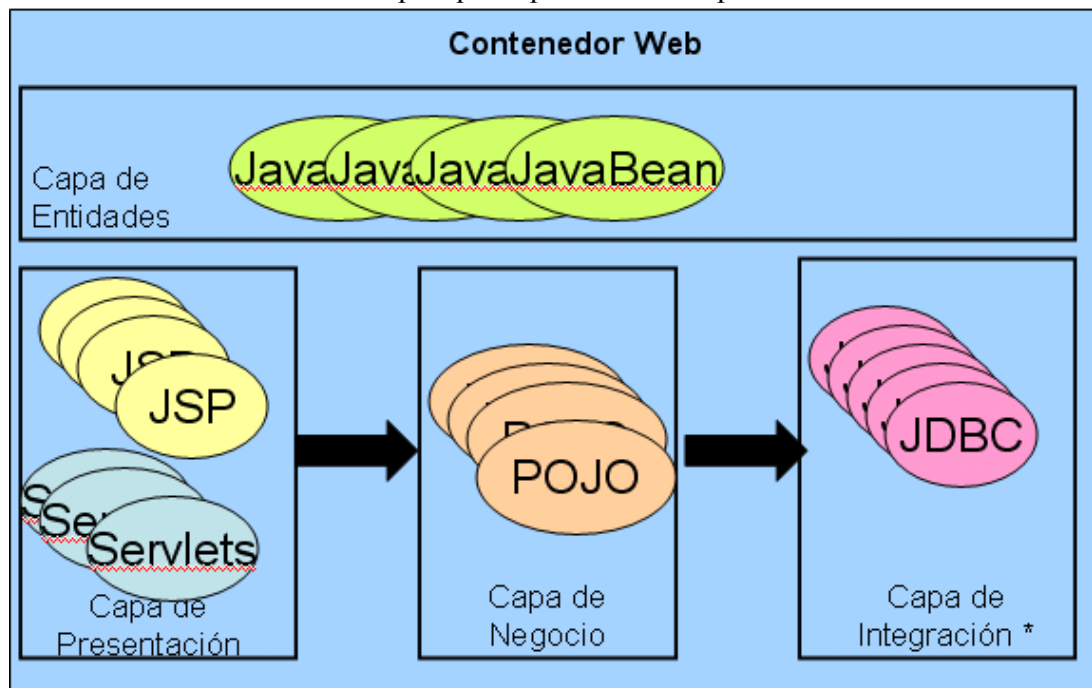


## 7 Servlets y formularios

### 7.1 Relación entre Servlets y JSPs

En este apartado vamos a explicar la utilidad final de todo lo explicado hasta ahora: documentos HTML, páginas JSP, JavaBean y Servlets.

Recordemos la estructura de capas que explicamos en la primera sesión



\*Aunque ponga JDBC en el dibujo realmente son dases que usan la tecnología JDBC para acceder a base de datos

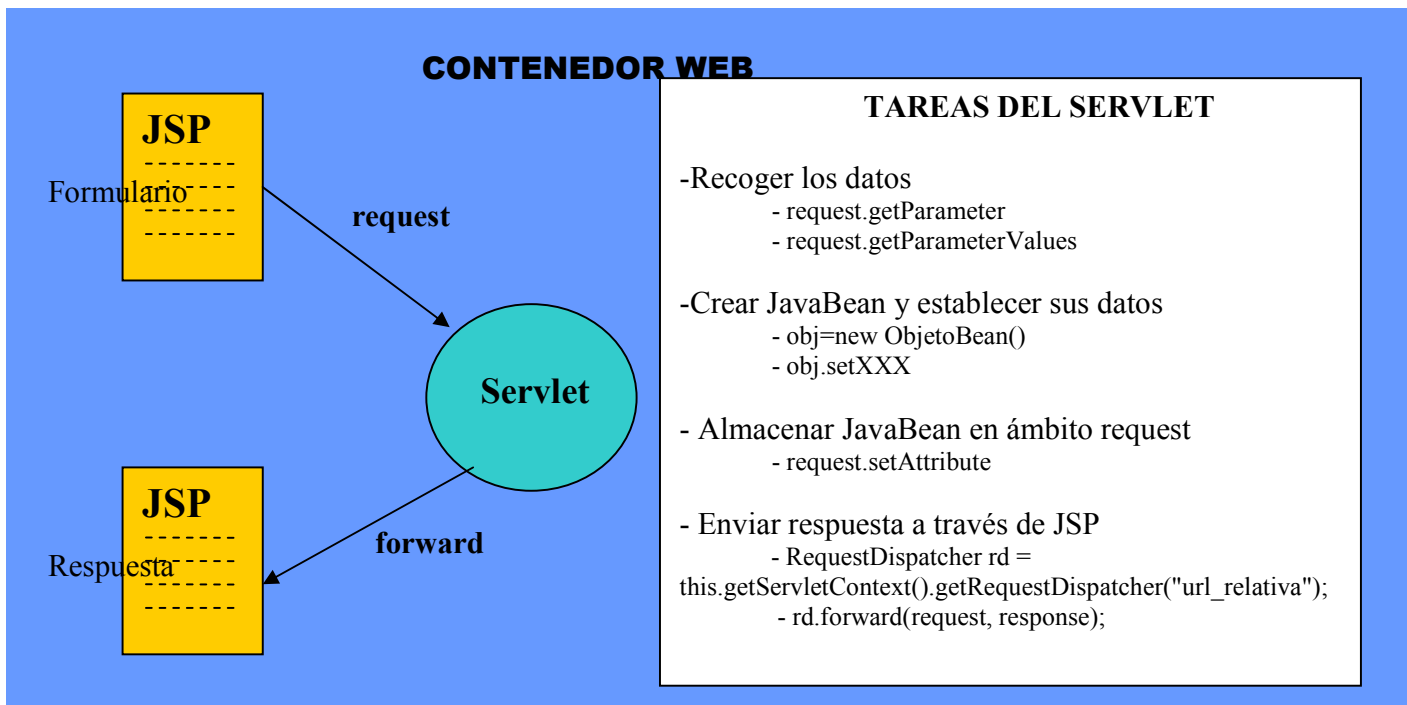
Con todo lo visto hasta ahora hemos trabajado en tecnología necesaria para la capa de Presentación y la de Entidades. En la siguiente sesión abarcaremos la capa de integración.

Nos encontramos en la capa de presentación y pretendemos ser capaces de

1. mostrar un formulario en pantalla a través de una página JSP
2. crear un objeto JavaBean
3. recoger en un Servlet los datos enviados y pasarlos a una instancia del JavaBean
4. almacenarlo en un ámbito de Request
5. mostrar la respuesta al cliente a través de otra página JSP

Para ello necesitaremos dos páginas JSP, una con el formulario y otra con la respuesta, y un Servlet capaz de recoger y procesar los datos.

Veámoslo a través de la siguiente figura



Veamos el ejemplo anterior en un caso concreto con NetBeans

## 7.2 Ejemplo desde NetBeans

Para el desarrollo del ejemplo anterior crearemos un nuevo proyecto llamado Sesión4\_2.

Rellenaremos, tal y como vimos en la anterior sesión, el formulario JSP llamado index.jsp que el proyecto propone por defecto para recoger los datos de un alumno. El código será el siguiente:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Entrada de datos</title>
  </head>
  <body>
    <form action="RecogeDatos" method="POST">
      <h2>Datos de Alumno</h2>
      Nombre: <input type="text" name="nombre" /><br>
      Primer apellido: <input type="text" name="apellido1" /><br>
      Segundo apellido: <input type="text" name="apellido2" /><br>
      <input type="submit" value="Enviar" />
    </form>
  </body>
</html>
```

Mediante el anterior formulario recogeremos los datos del usuario y se enviarán a un servlet llamado RecogeDatos. Previamente a crear este Servlet pasaremos a crear un JavaBean, tal y como estudiamos en la anterior sesión. Este lo llamaremos InfoAlumno, estará en el paquete Entidad y su código será:

```
package Entidad;

/**...*/
public class InfoAlumno {
    private String nombre;
    private String primerApellido;
    private String segundoApellido;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getPrimerApellido() {
        return primerApellido;
    }
    public void setPrimerApellido(String primerApellido) {
        this.primerApellido = primerApellido;
    }
    public String getSegundoApellido() {
        return segundoApellido;
    }
    public void setSegundoApellido(String segundoApellido) {
        this.segundoApellido = segundoApellido;
    }
}
```

Este Javabeen será usado tanto por el servlet RecogeDatos para establecer sus propiedades como por la página JSP que accederá a esas propiedades para mostrarlas por pantalla. Por lo tanto, hemos de crear un servlet llamado RecogeDatos, en el paquete Servlets, que tendrá el siguiente código en su método processRequest

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    InfoAlumno _datosAlumno=new InfoAlumno();
    _datosAlumno.setNombre(request.getParameter("nombre"));
    _datosAlumno.setPrimerApellido(request.getParameter("apellido1"));
    _datosAlumno.setSegundoApellido(request.getParameter("apellido2"));
    request.setAttribute("alumno", _datosAlumno);
    RequestDispatcher rd = this.getServletContext().getRequestDispatcher("/mostrarDatos.jsp");
    rd.forward(request, response);
}
```

Como se observa en la figura anterior se acceden a las variables enviadas por el formulario a través de `request.getParameter(..)`. Además se almacenan estos datos en el JavaBean anteriormente creado. Por último se pasa a mostrar la salida a través de la página JSP `mostrarDatos.jsp`.

Luego, nos queda por definir la página JSP llamada `mostrarDatos.jsp` que tal y como estudiamos en la sesión anterior hace uso de las acciones JSP que permiten acceder al JavaBeans (recordemos que NetBeans tiene ayudas gráficas para realizar esto). El código es:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<jsp:useBean id="alumno" scope="request" class="Entidad.InfoAlumno" />
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
</head>
<body>
    <h2>Alumno introdujo los siguientes datos:</h2><br>
    Nombre: <jsp:getProperty name="alumno" property="nombre" /> <br>
    Primer apellido: <jsp:getProperty name="alumno" property="primerApellido" /> <br>
    Segundo apellido: <jsp:getProperty name="alumno" property="segundoApellido" /> <br>
</body>
</html>
```

Si probamos la miniaplicación creada tendremos, al acceder a `index.jsp` un formulario con los datos del alumno a rellenar



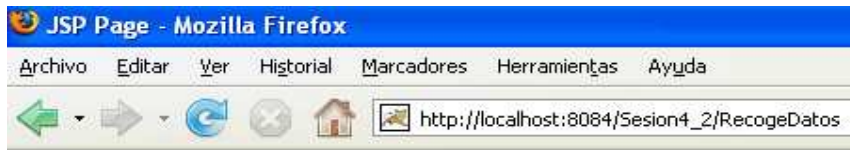
## Datos de Alumno

Nombre:

Primer apellido:

Segundo apellido:

Una vez rellenados los datos y pulsemos enviar pasaremos a ver la siguiente pantalla.



### **Alumno introdujo los siguientes datos:**

Nombre: Juan  
Primer apellido: Perez  
Segundo apellido: Pinzon

De este modo habremos completado un proceso de navegación de cliente (aunque sea básico). En esta miniaplicación hemos aplicado lo visto hasta esta sesión:

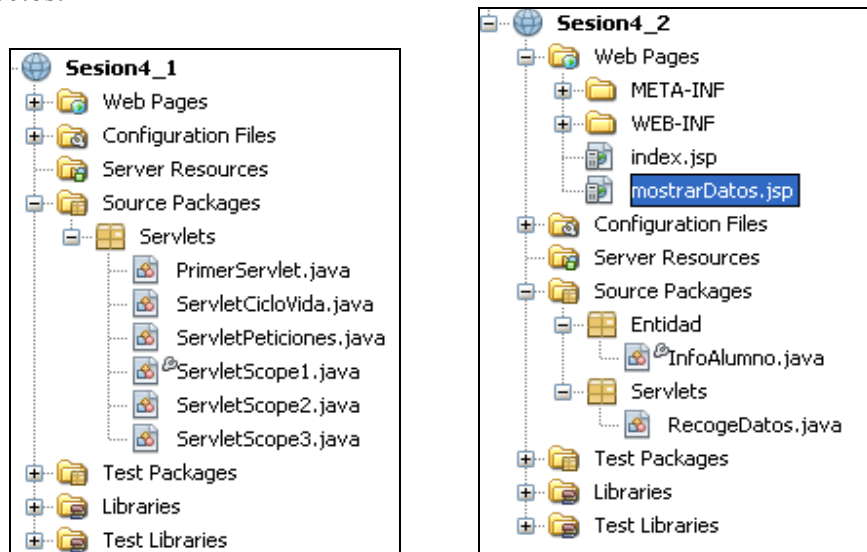
- Uso de formularios
- Uso de páginas JSP
- Uso de Servlets
- Uso de ámbitos de los objetos
- Uso de JavaBeans

## 8 Ejercicios

Realiza los siguientes ejercicios y envíalos en un comprimido con tu nombre y apellidos al profesor. (Ej.: vperezcabello\_sesion4.zip).

### 8.1 Ejercicio 1(Entregado por el profesor)

Realiza las aplicaciones ejemplos Sesión4\_1 y Sesión4\_2 explicadas a lo largo de la sesión. Para que te sirva de guía estos son los elementos que deben tener estos proyectos:



**NOTA:** Este ejercicio es entregado por el profesor, de modo que el alumno tenga disponible el código indicado en la sesión.

Realiza los siguientes ejercicios creando una nueva aplicación Web **Sesión4\_3** desde NetBeans

### 8.2 Ejercicio 2 (3 puntos)

Basándonos en el ejercicio 4 de la sesión 3, copia los formularios y el JavaBean AlquilerBean, pero ahora el formulario1.jsp no debe procesar los datos, sino que solo debe mostrar la pantalla de captura de información y enviar la información a un servlet llamado CapturaDatosVideoClub. Añadiremos este servlet para recoger los datos, completar el javabean y pasar la presentación al formulario2.jsp. De este modo tendremos:

- Formulario1.jsp (visualmente será idéntico al de la sesión3-ejercicio4)
- AlquilerBean (JavaBean)
- CapturaDatosVideoClub (servlet)
- Formulario2.jsp (visualmente será idéntico al de la sesión3-ejercicio4)

Nota: El objetivo de este ejercicio es comprender la mejora que supone separar la presentación del procesamiento de los datos.

### 8.3 Ejercicio 3 (4 puntos)

Queremos realizar un buscador de alumnos por DNI. Dado que todavía no hemos visto la parte de acceso a datos almacenaremos esta información en un listado (ArrayList) de un Servlet.

Tendremos un primer formulario llamado IntroduceDNI.jsp, en el cual introduciremos el DNI. Su aspecto será:

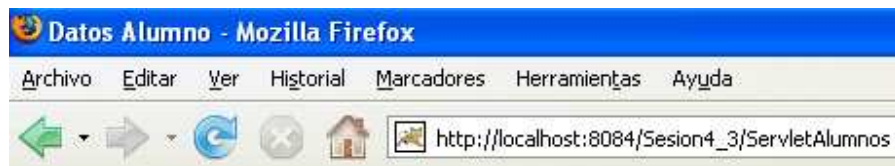


Crearemos un JavaBean (en package Entidades) llamado AlumnoBean que tendrá como propiedades el nombre, el primer apellido, el segundo apellido y el DNI.

El formulario IntroduceDNI.jsp enviará la información a un Servlet llamado ServletAlumnos, el cual hará

- método init
  - o Instanciará en tres objetos diferentes la clase AlumnoBean
  - o Cargará un ArrayList (java.util.ArrayList) con los tres alumnos instanciados
- método processRequest
  - o Buscará dentro del ArrayList inicializado en init el alumno cuyo DNI coincida con el indicado
  - o Si encuentra este DNI mostrará los datos por pantalla a través de un formulario llamado MuestraDatosAlumno.jsp

El formulario MuestraDatosAlumno.jsp mostrará los datos del alumno elegido por pantalla. Su aspecto será el siguiente:



### Los datos del alumno son:

**Nombre:** 123  
**Primer apellido:** Perez  
**Segundo apellido:** Pinzon  
**DNI:** Juan

Nota\_1: la clase ArrayList pertenece al package java.util.ArrayList. Un ejemplo de uso (en este caso para añadir información):

```
AlumnoBean _alumno=new AlumnoBean();
_alumno.setDni("123");
_alumno.setPrimerApellido("Perez");
_alumno.setSegundoApellido("Pinzon");
_alumno.setNombre("Juan");
_listado.add(_alumno);
```

Nota\_2: este ejercicio amplía el ejemplo 7\_2

### 8.4 Ejercicio 4 (3 puntos)

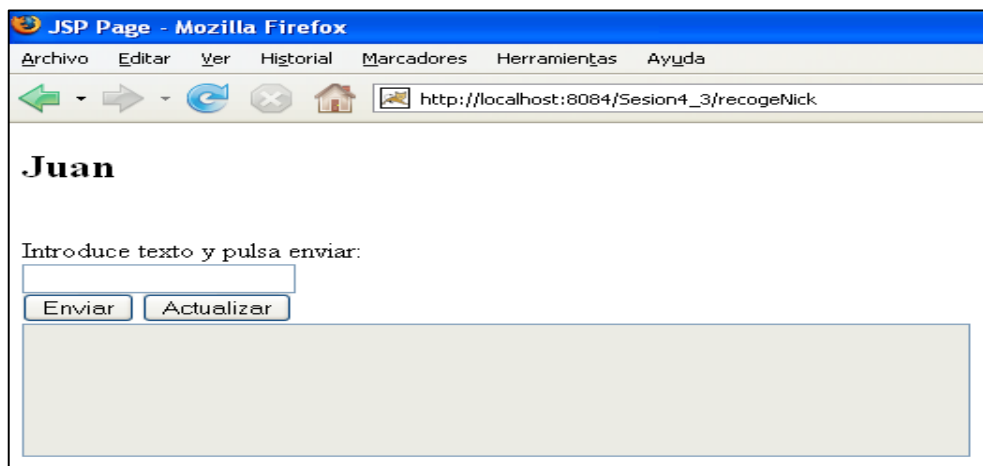
Realizaremos una pequeña aplicación que implemente un chat entre usuarios por pantalla. Para ello tendremos un primer formulario llamado entradaChat.jsp donde solicitaremos el Nick. Su aspecto será:

A screenshot of a Mozilla Firefox browser window. The title bar says 'Introduce Nick - Mozilla Firefox'. The address bar shows 'http://localhost:8084/Sesion4\_3/entradaChat.jsp'. The main content area has the heading 'Introduce el Nick que usaras' in bold. Below the heading is a text input field and a button labeled 'Enviar'.

Este nick se enviará a un servlet llamado recogeNick, el cual almacenará este Nick en el **ámbito de sesión**, y pasará la salida a otro formulario llamado datosChat.jsp. Este formulario datosChat.jsp debe

- Tener un campo de texto donde el usuario escriba
- mostrar en pantalla el Nick del usuario
- mostrar en pantalla lo que los demás usuarios indican, a través de un textarea y usando un JavaBean llamado datosPantallaBean que esta almacenado en el **ámbito de application**.
- Enviar la información a un servlet llamado recogeDatosChat a través del botón Enviar o del botón Actualizar

Su aspecto será:

A screenshot of a Mozilla Firefox browser window. The title bar says 'JSP Page - Mozilla Firefox'. The address bar shows 'http://localhost:8084/Sesion4\_3/recogeNick'. The main content area has the heading 'Juan' in bold. Below the heading is the text 'Introduce texto y pulsa enviar:'. There is a text input field, and below it are two buttons: 'Enviar' and 'Actualizar'. At the bottom of the page is a large, empty text area.

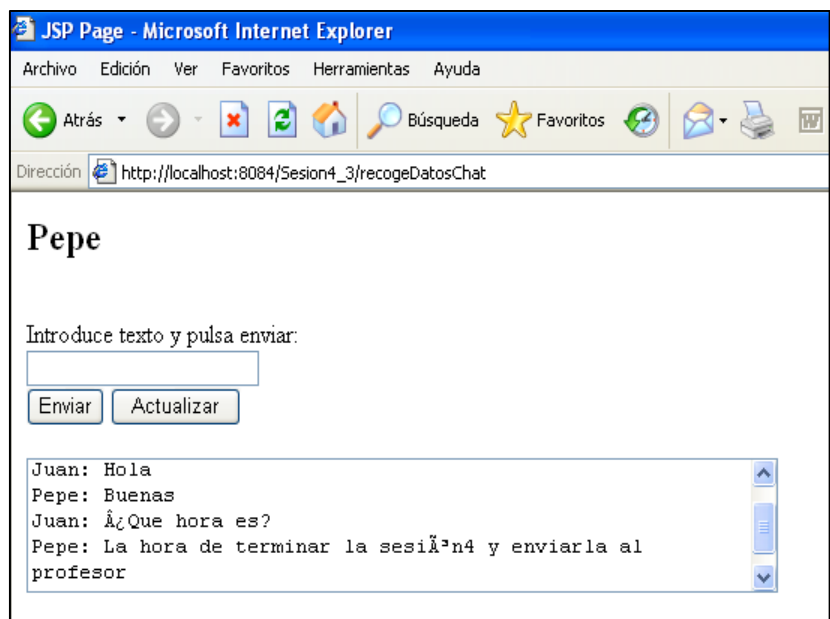
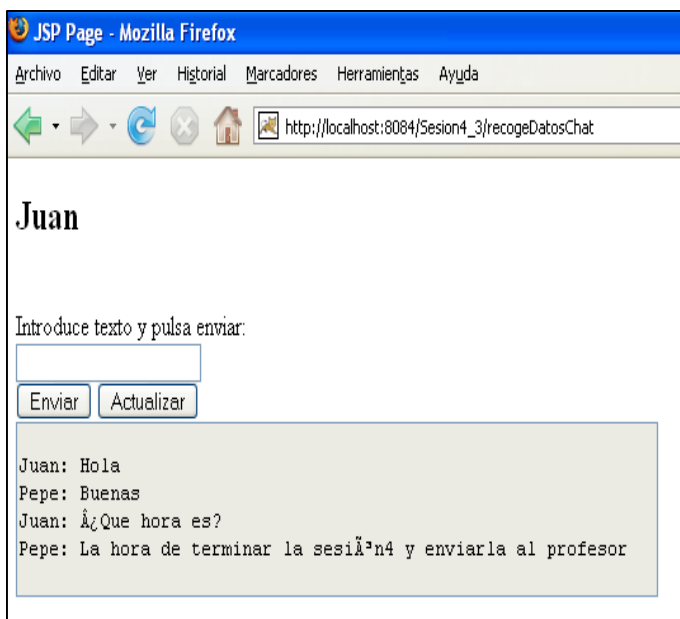


El JavaBean llamado datosPantallaBean tendrá una única propiedad llamada texto introducido.

El servlet recogeDatosChat deberá

- recoger el texto introducido por el usuario
- Si el texto es distinto de blanco lo almacenará junto al texto ya existente en el JavaBean datosPantallaBean, que estará en el **ámbito de application**. Si el texto es blanco (suponemos que habrán pulsado Actualizar) no haremos este paso (más adelante estudiaremos como hacer esta distinción de forma más adecuada).
- pasar la salida a datosChat.jsp de nuevo.

De este modo, un ejemplo de conversación sería



Nota: El aspecto del proyecto sesion4\_3 es

