**MÓDULO 3**

**EJERCICIOS RESUELTOS ATENDIENDO PETICIONES SERVLET**

**TEORIA 1**

Explica el ciclo de vida de un Servlet.

1) La clase Java que representa el Servlet se despliega en el contenedor web.

2) El contenedor web crea una instancia de la clase Servlet y la carga en memoria.

3) El contenedor Web pasa seguidamente a la fase de inicialización del Servlet, llamando al metodo init(ServletConfig).La implementación de dicho método permite, mediante el objeto ServletConfig, recuperar los parámetros de inicialización del Servlet, definidos en descriptor de despliegue (web.xml) de la aplicación.

4) El Servlet pasa a continuación a un estado llamado “en Servicio”, en espera de recepción de peticiones cliente. El contenedor web genera una cola de espera de threads, y ejecuta la instancia del servlet en un contexto multitarea. Cada thread llama al método Service(…request,….response) del servlet, quien se ocupa de ejecutar el método http correspondiente (doPost(),doGet())

5) La instancia del servlet permanece cargada en memoria mientras se ejecute el contenedor web. Si éste se para, se llama al método destroy() sobre la instancia Servlet, indicando que ya no está en servicio.

6) Y, para acabar, como un Servlet es un objeto Java, la instancia del Servlet es borrada de la memoria por el garbage collector de Java.

**TEORIA 2**

Explica cómo funciona un Servlet a nivel interno (métodos service y doXXX)

Cuando un cliente llama a un servlet emitiendo una petición http, se llama al método service() sobre el contenedor Web. El método service(…) recibe dos parámetros, de tipo javax.servlet.ServletRequest y javax.serlet.ServletResponse.

El único proceso realizado por el método Service(..) es la invocación del propio método service(..) sobrecargado, que toma dos parámetros, del tipo javax.servlet.http.HttpServletRequest y javax.servlet.http.HttpServletResponse.

Además el método service(..) invoca por defecto al metodo correspondiente al método HTTP utilizado por el cliente (principalmente doGet(…),doPost(..),doPut(..),doDelete(..)), transmitiéndole los parámetros de tipo javax.servlet.http.HttpServletRequest y javax.servlet.http.HttpServletResponse.

El desarrollador será quien se encargue de implementar entonces los procesos sobre la respuesta del método doXXX(…) o aquellos que utilicen la misma.

**TEORIA 3**

3. Comenta la relación que hay entre las clases Servlet, ServletRequest, ServletResponse,

HTTPServlet, HTTPServletRequest y HTPServletResponse.

Las clases Servlet, ServletRequest y ServletResponse estan contenidas en la API Servlet en el paquete javax.servlet.

Las clases HTTPServlet, HTTPServletRequest y HTTPServletResponse están contenidas en la API Servlet en el paquete javax.servlet.http

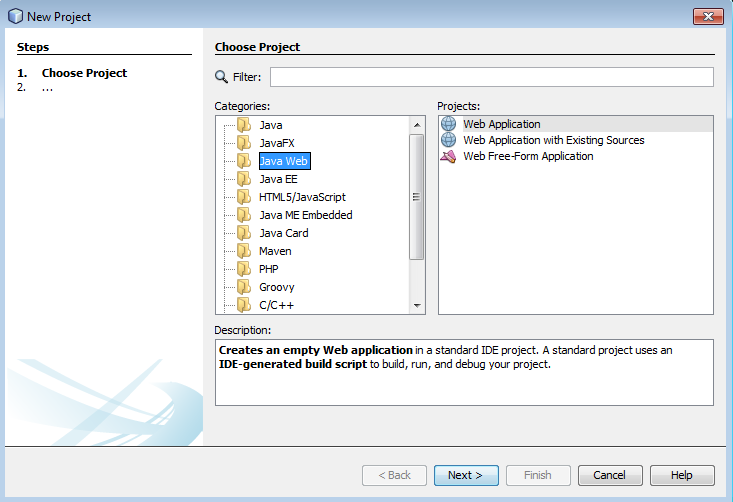
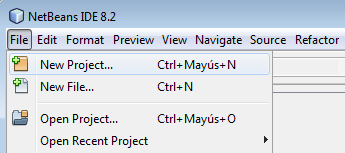
ServletRequest y ServletResponse se utilizan como parámetros para la implementación del método service (public void service(ServletRequest req, ServletResponse res)), mientras que HTTPServletRequest y HTTPServletResponse se utilizan para la implementación del método sobrecargado:

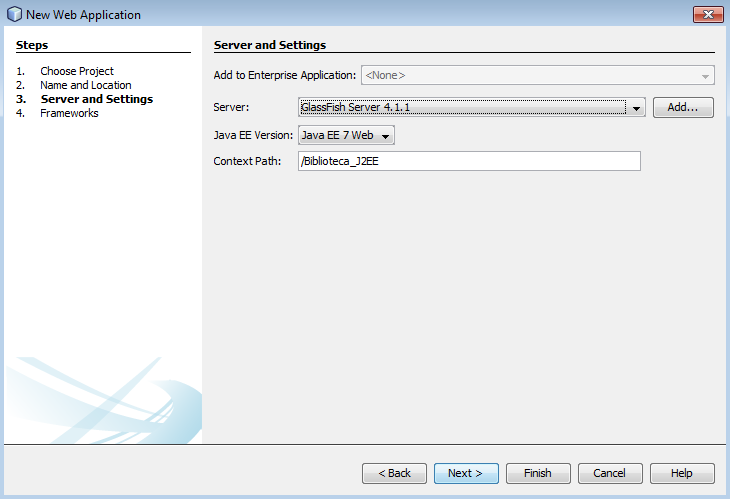
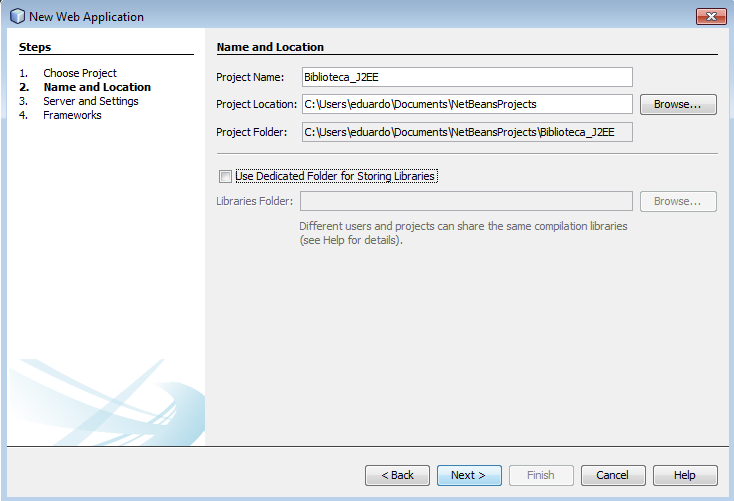
service( public void service (que HTTPServletRequest req, HTTPServletResponse res))

**EJERCICIO 1**

Crea un Proyecto J2EE al que llamaremos Biblioteca\_J2EE con NetBeans o Eclipse. Estos IDE’s crean automáticamente la estructura del proyecto J2EE con sus carpetas (WEB-INF, clases… etc.) y su fichero de configuración web.xml. Si el IDE no tiene un servidor predefinido, agrégalo para que el proyecto lo tenga asignado. Esto lo podemos hacer en el momento de la creación del proyecto o posteriormente cuando ya está creado:

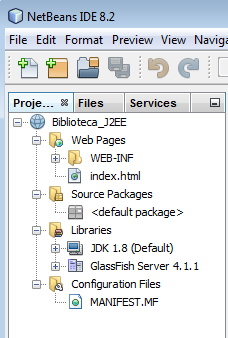
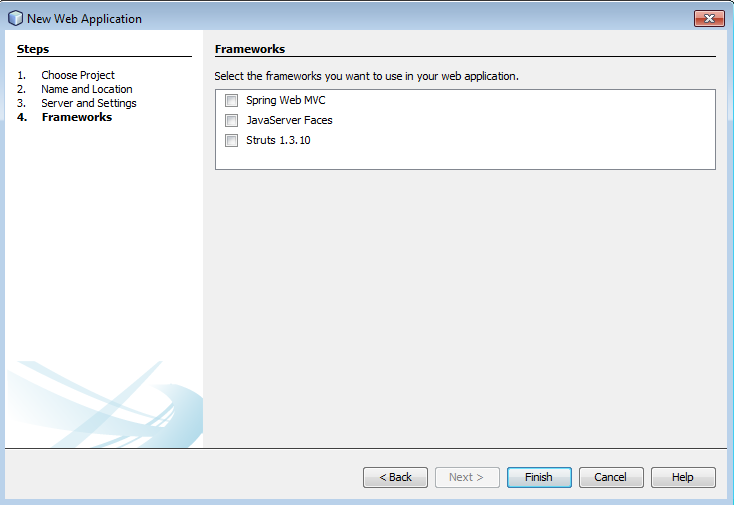
**NETBEANS**



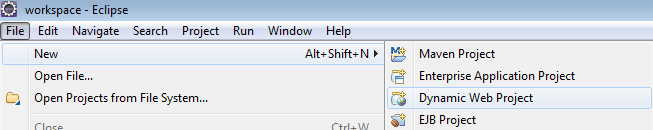


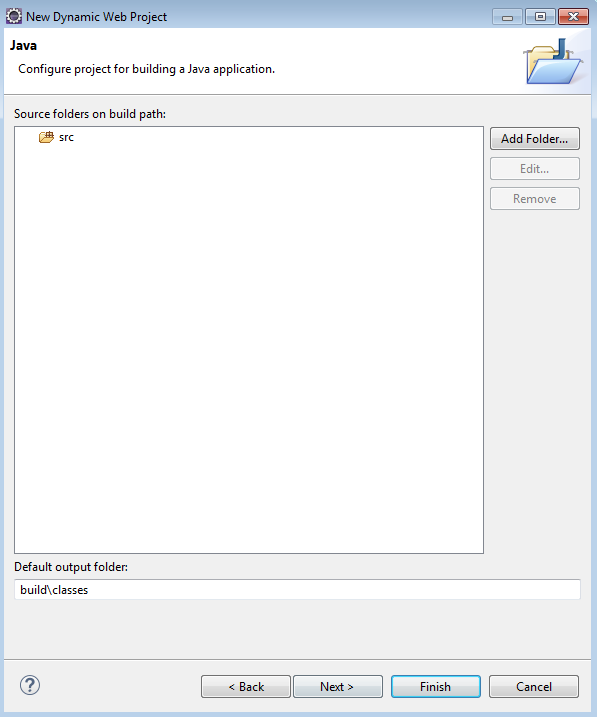
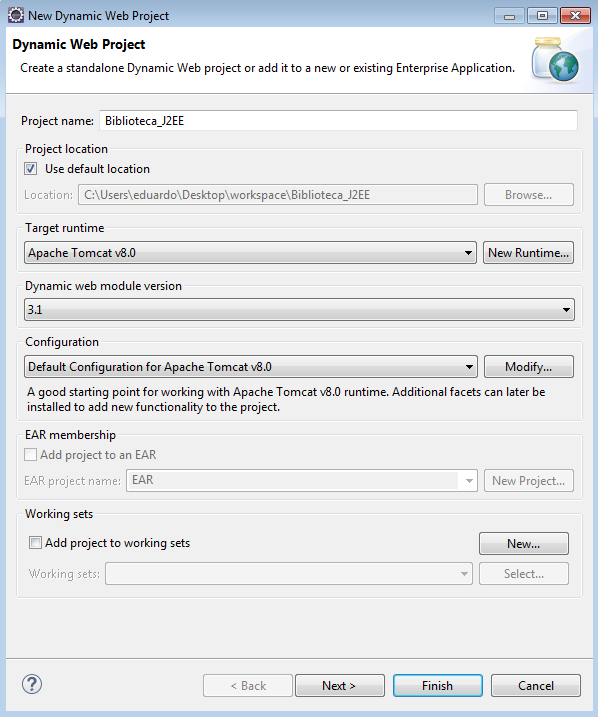
Si no tuviésemos definido ningún servidor de aplicaciones en Netbeans, lo podemos agregar en este mismo momento (durante la creación del proyecto) o posteriormente cuando ya está creado:

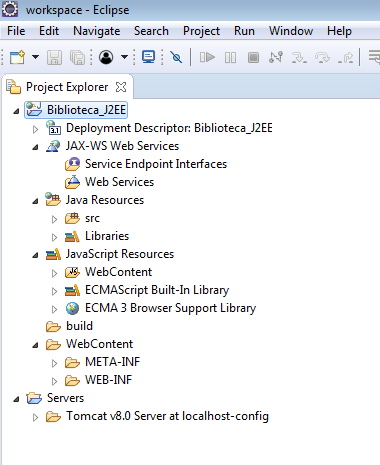
|  |
| --- |
| Agregar servidor en el momento de creación del proyecto |
|  |
|  |



**ECLIPSE**



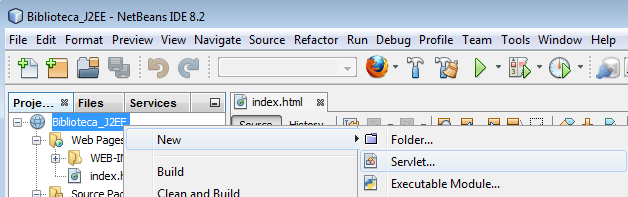


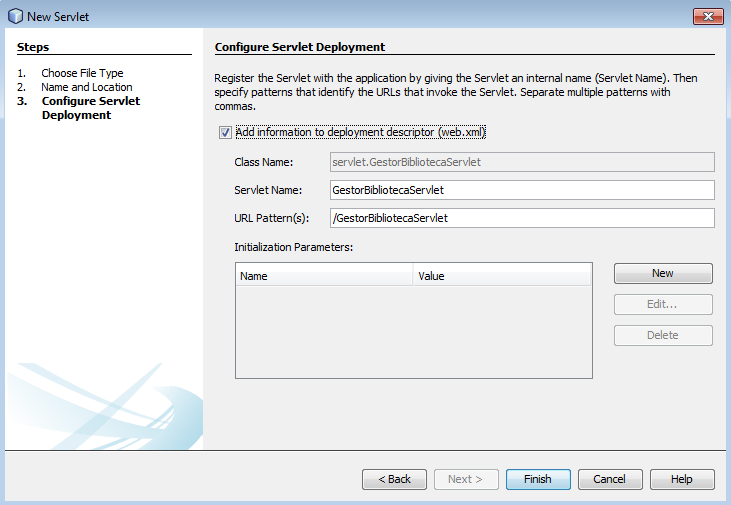
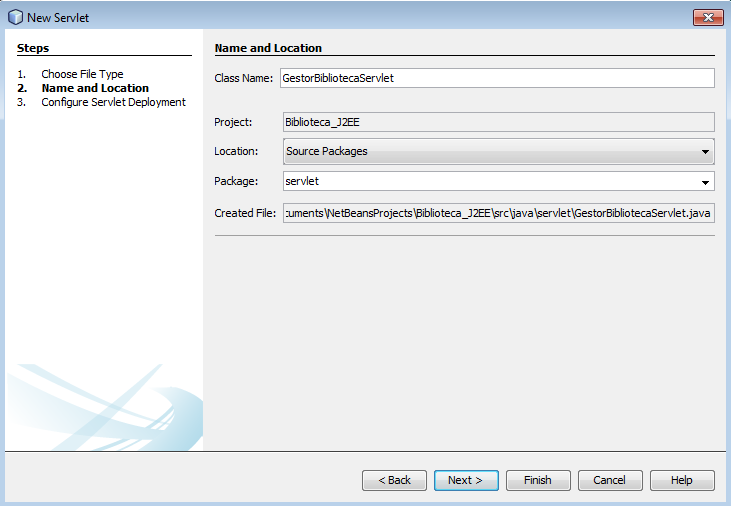


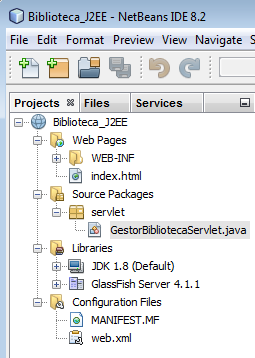
**EJERCICIO 2**

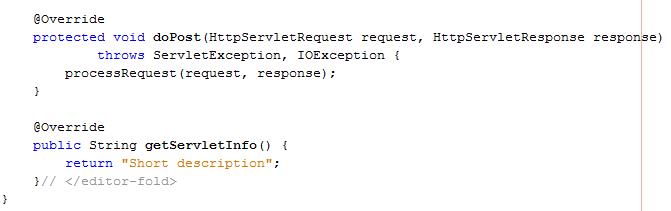
Crea un Servlet al que llamaremos “GestorBibliotecaServlet”, dentro del package servlet.

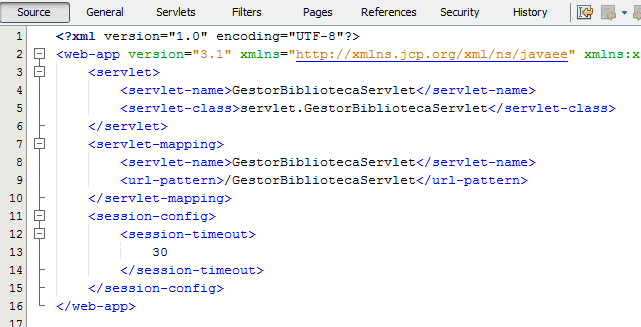
**NETBEANS**



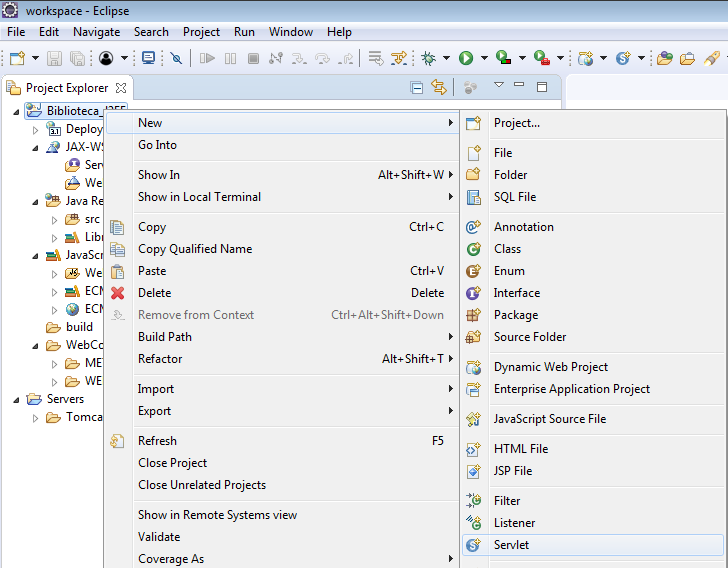


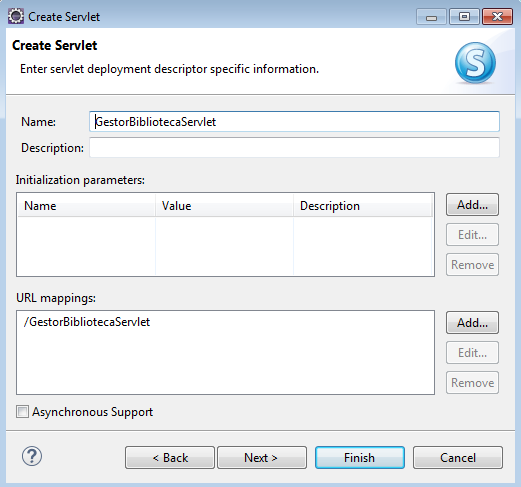
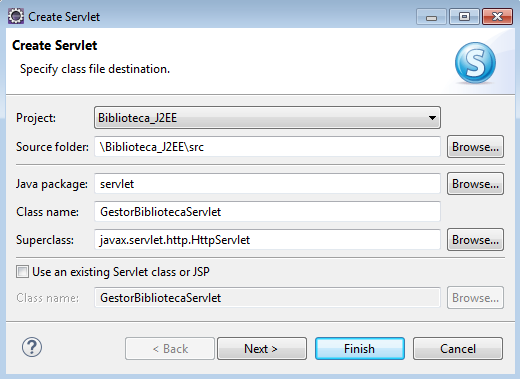


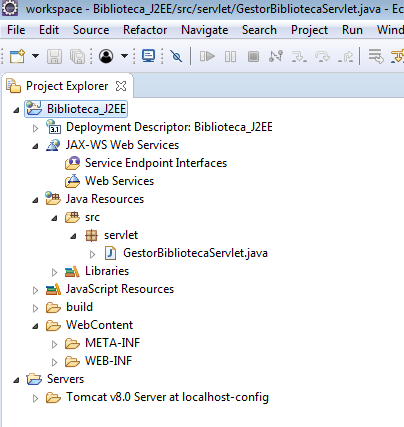
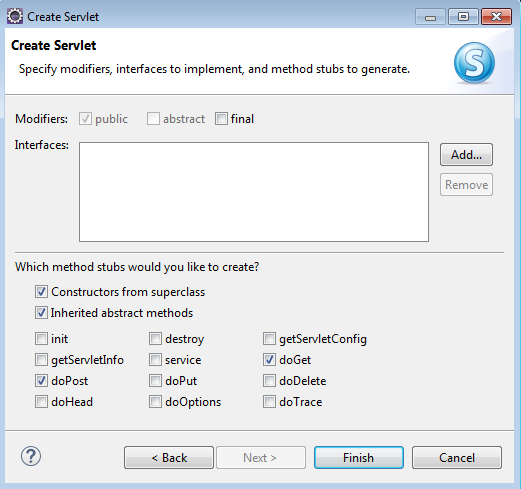


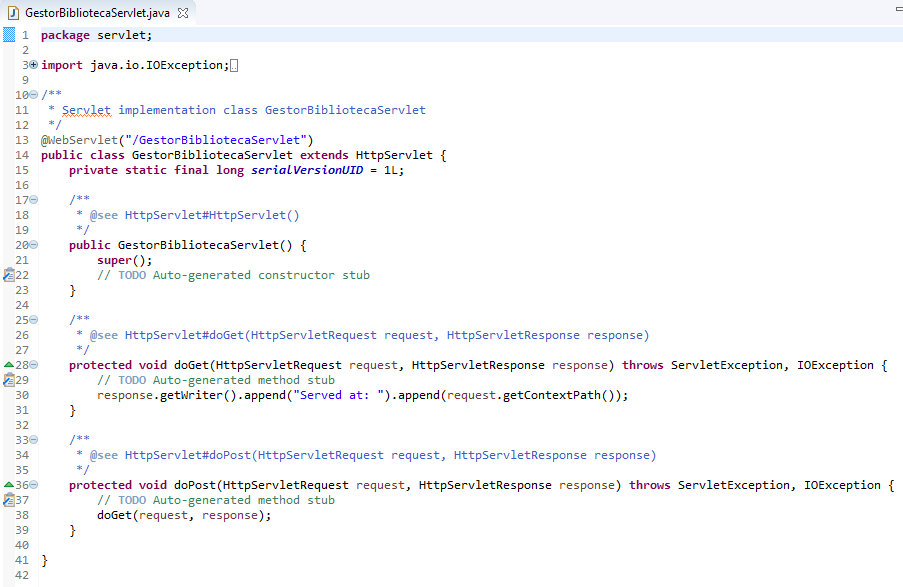


**ECLIPSE**









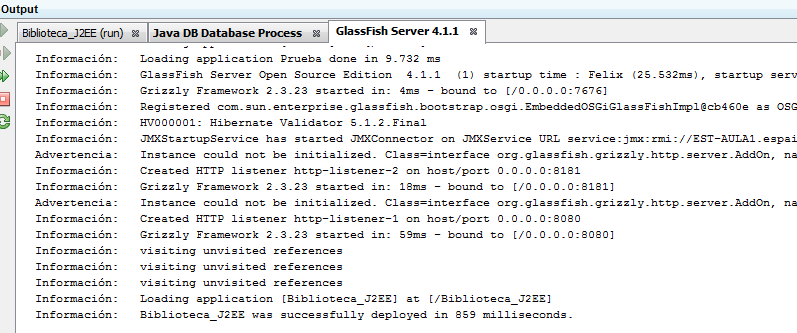
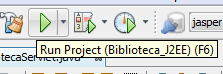
**EJERCICIO 3**

Ejecuta la aplicación web e inicia el servidor web. Desde un navegador pon la siguiente url:

http://localhost:8080/Biblioteca\_J2EE/GestorBibliotecaServlet

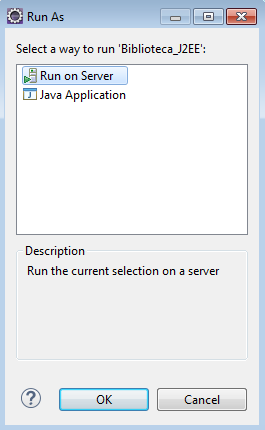
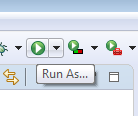
Podemos comprobar que el servlet queda configurado como un servicio web que está preparado para atender peticiones a través del protocolo HTTP.

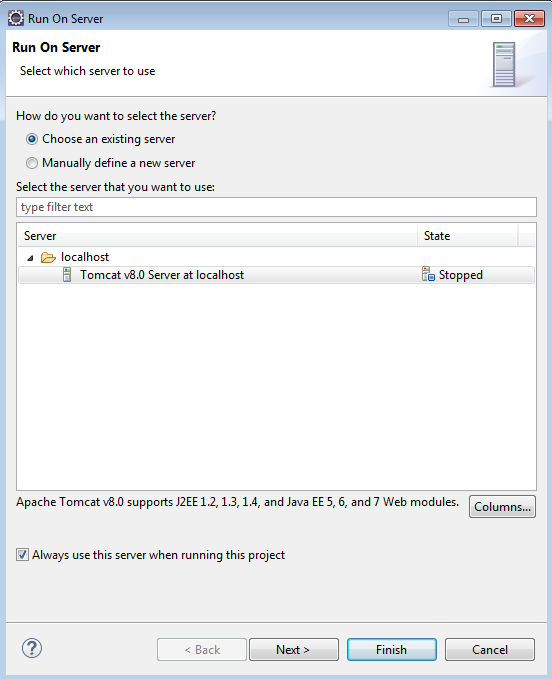
**NETBEANS**

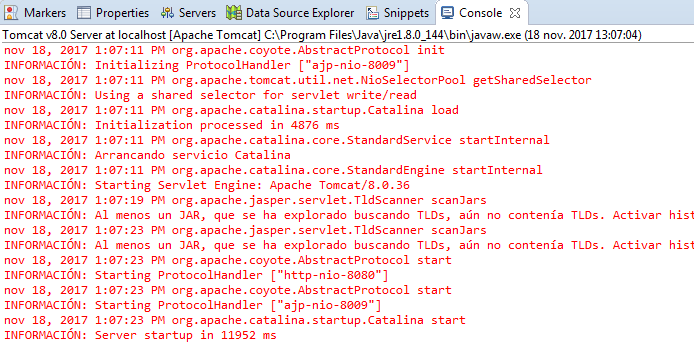


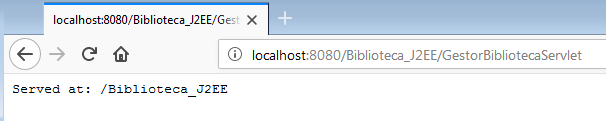


**ECLIPSE**









**EJERCICIO 4**

Modifica la respuesta del servlet GestorBibliotecaServlet para que pueda dar mensajes diferentes en función del tipo de petición:

* “Hola Mundo POST!!” si recibe una petición POST
* “Hola Mundo GET!!” si recibe una petición GET.

Sigue el modelo que utiliza Eclipse en la creación por defecto de la plantilla de un servlet, el cual hace las respuestas de doGet y de doPost independientes mediante la instrucción:

response.getWriter().append(mensaje).

**NETBEANS**



**ECLIPSE**



**EJERCICIO 5**

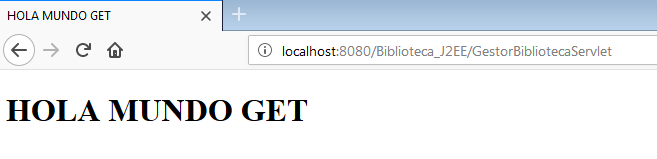
Repite el ejercicio 3. Inicia la aplicación web y desde un navegador pon la siguiente url:

http://localhost:8080/Biblioteca\_J2EE/GestorBibliotecaServlet

¿Qué mensaje da el servlet?

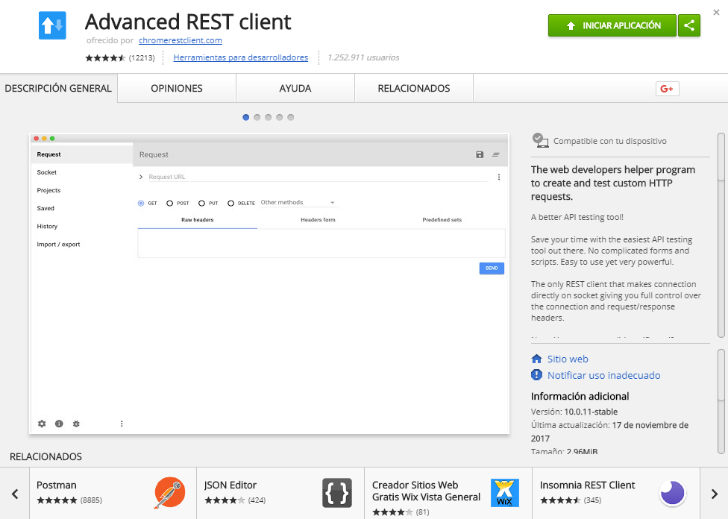
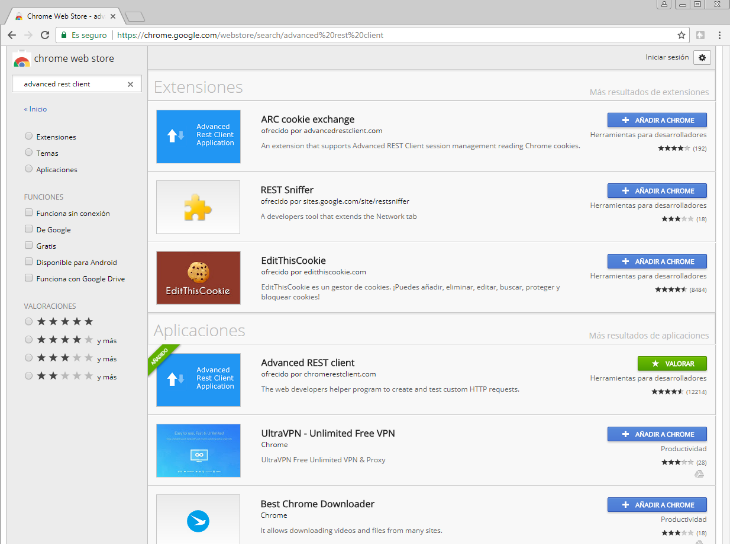
Indica dos sistemas que nos permitirían obtener los dos tipos de mensajes: GET y POST.

Idea: Mira el complemento Advanced Rest Client del navegador Chrome.

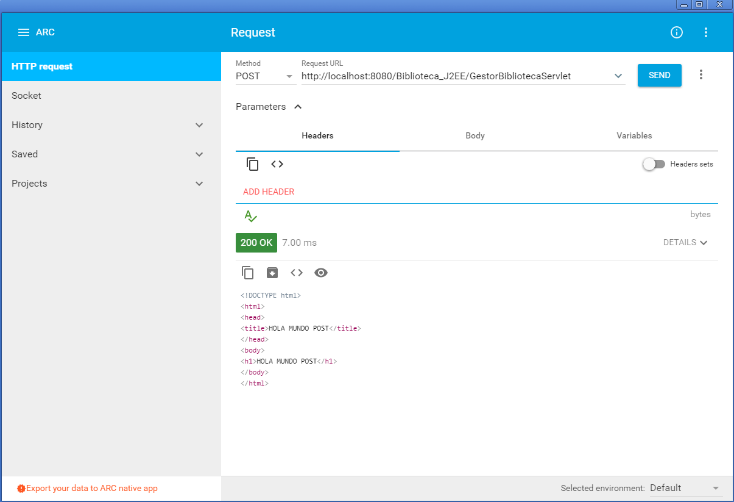
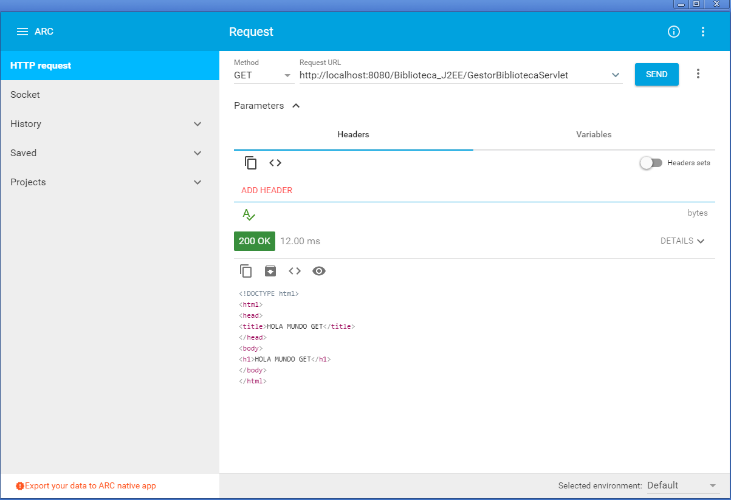


El mensaje que nos da el servlet es “Hola Mundo GET”, que es debido a que todas las llamadas realizadas directamente con una url desde un navegador producen un mensaje tipo GET hacia el servlet. Para conseguir realizar peticiones HTTP de los dos tipos, tanto GET como POST, se deben de hacer con formularios HTML o con algún plugin de un navegador.

En el caso del navegador Chrome podemos descargarnos e instalar el Advanced Rest Client:



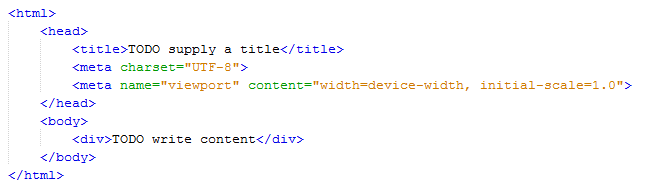
Podemos generar peticiones tanto GET como POST gráficamente hacia nuestro servlet:

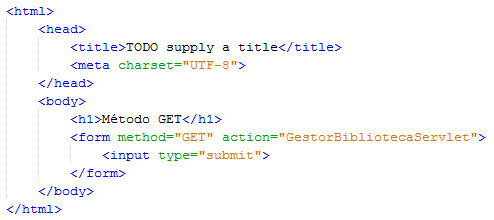


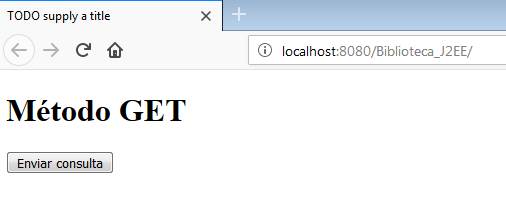
**EJERCICIO 6**

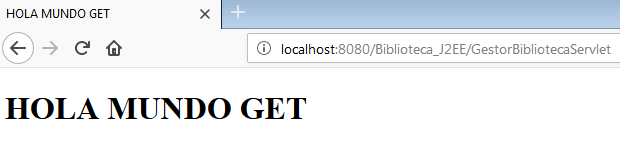
Utiliza la página web index.html para crear un formulario que contenga un único botón de submit. Este formulario debe llamar al servlet GestorBibliotecaServlet a través del método GET.

Ejecuta la aplicación y llama al servlet desde el formulario. Indica cual es el mensaje de salida que se obtiene.



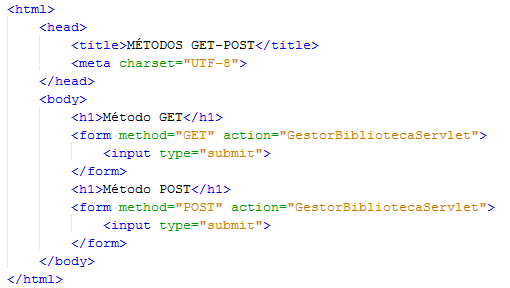


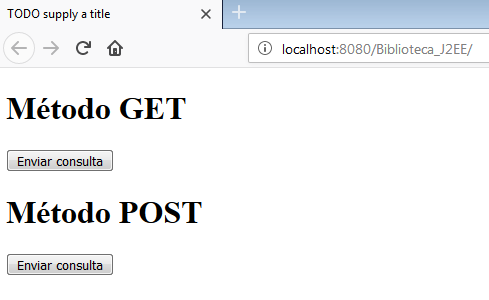




**EJERCICIO 7**

Agrega otro formulario en el archivo index.html de manera que llame al servlet GestorBibliotecaServlet a través del método POST. Ejecuta la aplicación y mira el mensaje de salida que da el servlet.



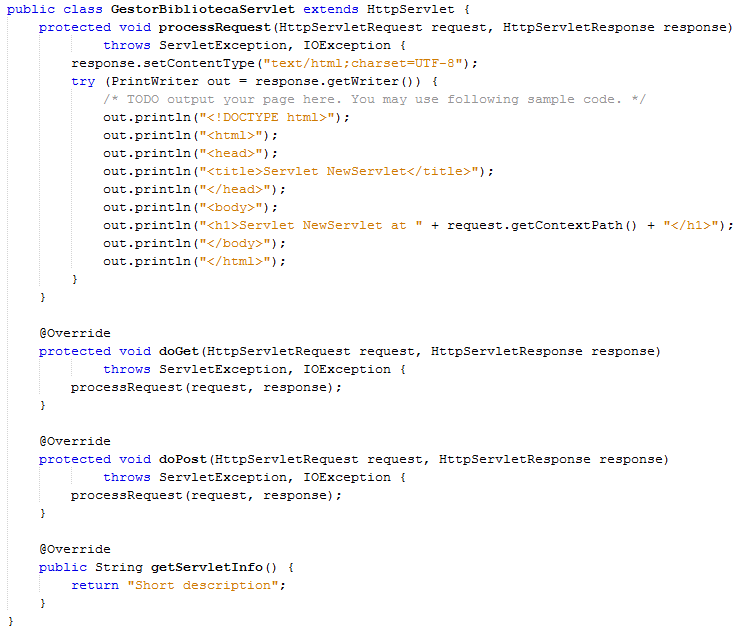


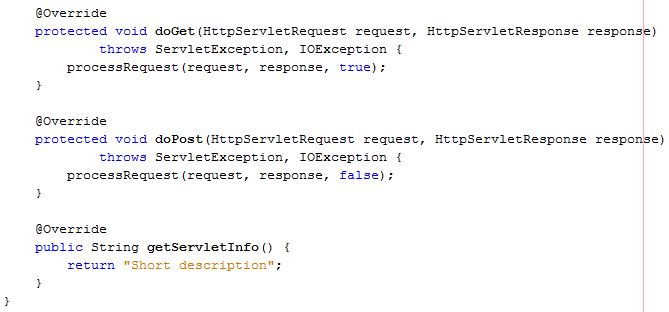


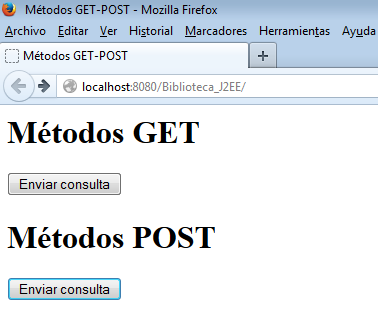
**EJERCICIO 8**

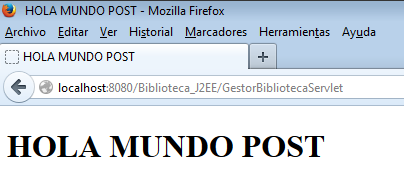
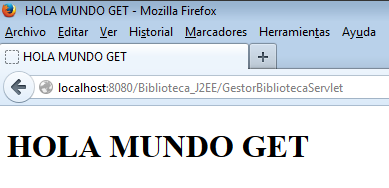
Aprovechando la estructura de la función processRequest, que crea por defecto Netbeans cuando crea la plantilla de un servlet, haz que el servlet GestorBibliotecaServlet siga haciendo lo mismo, es decir dar un mensaje de “Hola Mundo POST!!” si recibe una petición POST y “Hola Mundo GET!!” si recibe una petición GET.

Idea: Para lograrlo se puede incluir un nuevo parámetro booleano dentro de la función processRequest.



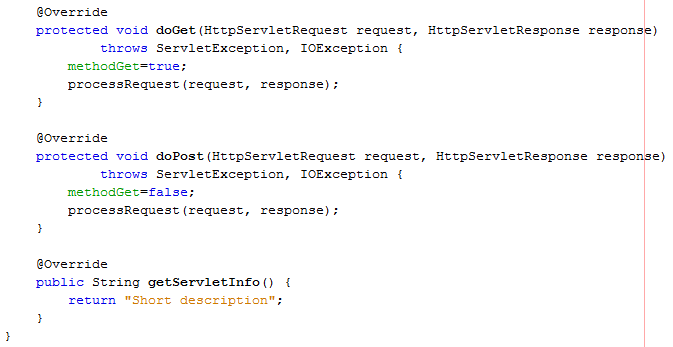






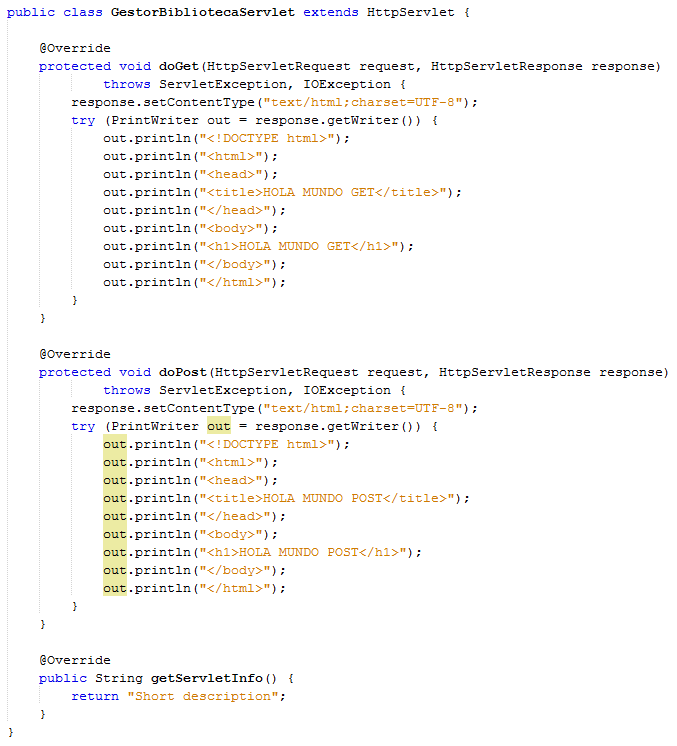
**EJERCICIO 9**

Realiza la misma operativa que en el ejercicio anterior, usando la estructura de la función processRequest, pero ahora utiliza una variable privada booleana para diferenciar entre llamadas GET y POST.



**EJERCICIO 10**

Divide la función processRequest en dos partes, la parte GET y la parte POST, y asigna cada parte a las funciones doGet y doPost, de forma que funcionen de forma independiente y no se tenga que utilizar variables para diferenciar entre ambas llamadas.

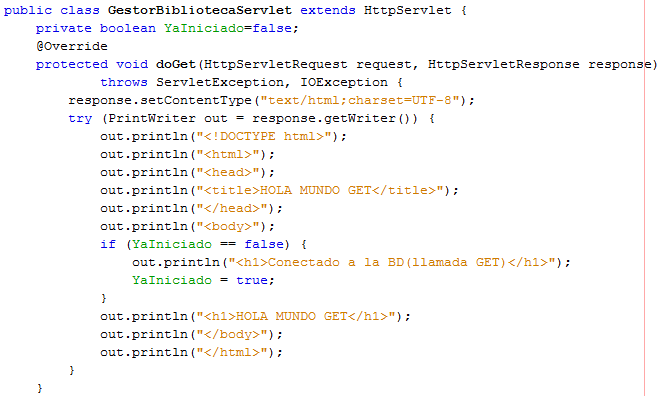


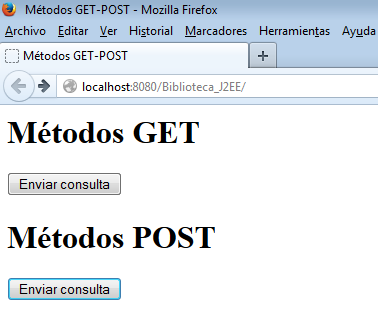
**EJERCICIO 11**

Trataremos la conexión con la base de datos en capítulos posteriores, pero como “adelanto” debemos saber que ésta se iniciará en cuanto se cree el Servlet en memoria.

Modifica el ejercicio para que el usuario reciba el mensaje “Conectado a la BD” sólo la primera vez que acceda al Servlet.

Lo haremos con una variable booleana YaIniciado.

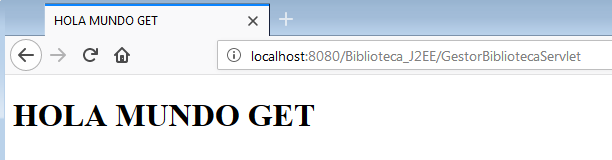




Primera llamada:



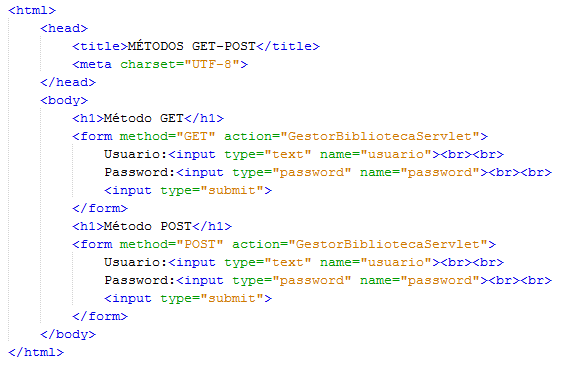
Segunda llamada (tirando hacia atrás en el navegador):



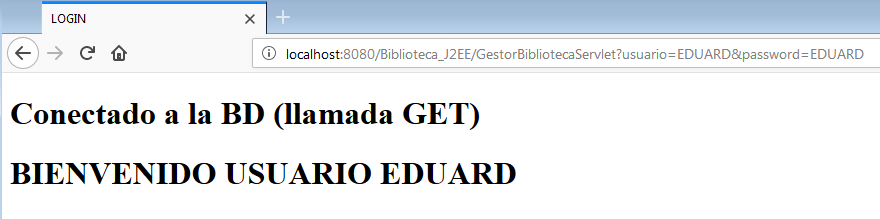
**EJERCICIO 12**

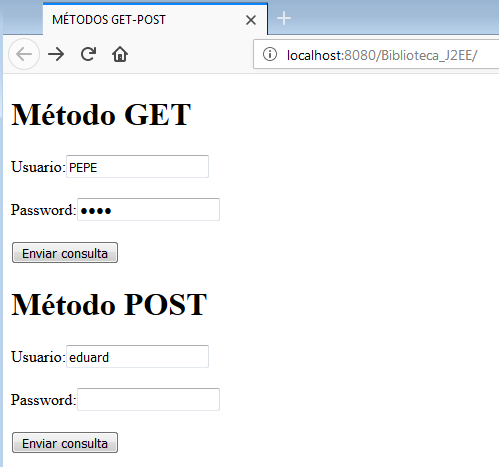
Nos interesa saber quién se conecta a nuestra biblioteca. Para ello, pasaremos el usuario y su contraseña al servlet para que ésta valide si tenemos permisos para acceder.

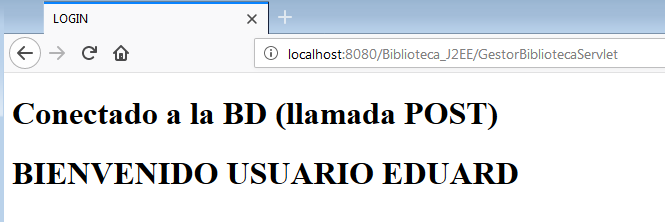
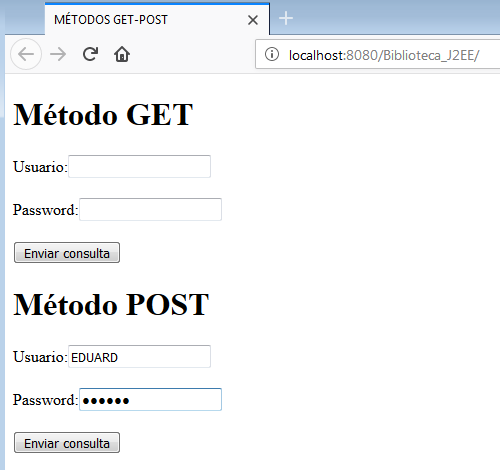
Añade 2 campos a los formularios de la página web index.html que se llamarán usuario y password. El servlet tiene que contestar con “Bienvenido [usuario]” si se valida el acceso del usuario o “Acceso no permitido” en caso contrario.

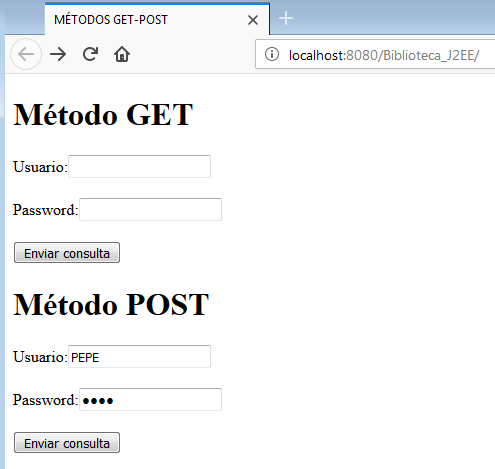








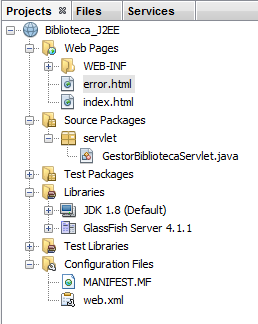


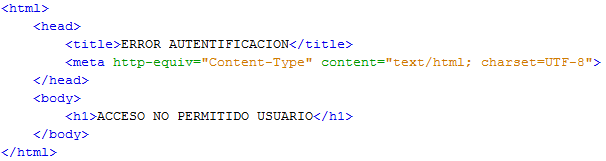


**EJERCICIO 13**

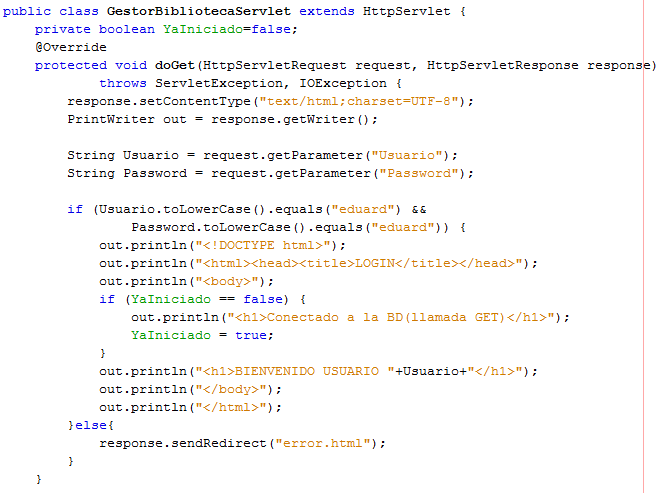
Ahora crea el fichero error.html, el cual será redireccionado por el servlet cuando la identificación sea incorrecta. Por lo tanto el servlet ya no dibujará directamente el resultado de la identificación errónea, sino que mostrará un fichero html donde se muestre el error. ¿Se le puede pasar un parámetro a un fichero html?

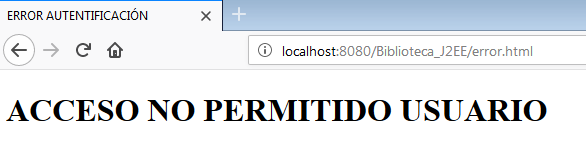
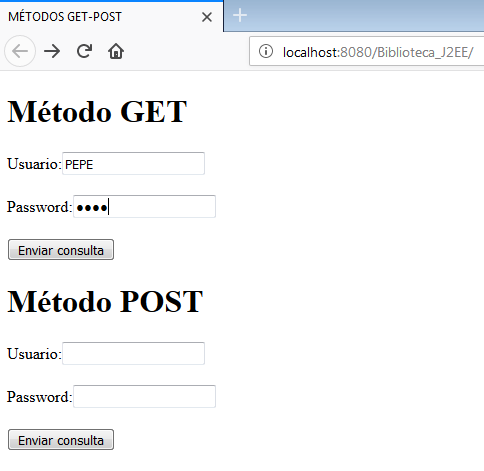
NOTA: Utiliza la función response.sendRedirect(“”).





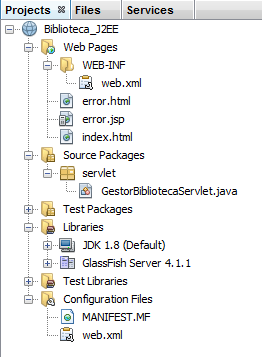
El problema es que no podemos pasarle a un fichero HTML un parámetro desde un servlet, o dicho de otra manera, un HTML no puede recoger parámetros HTTP. Si en lugar de un HTML fuera un JSP, entonces sí que podría.

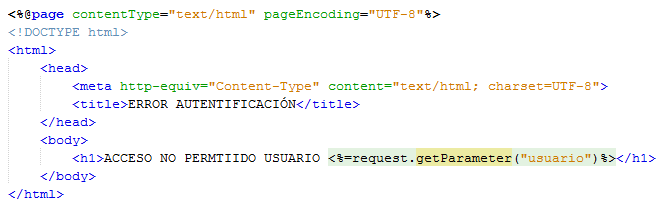




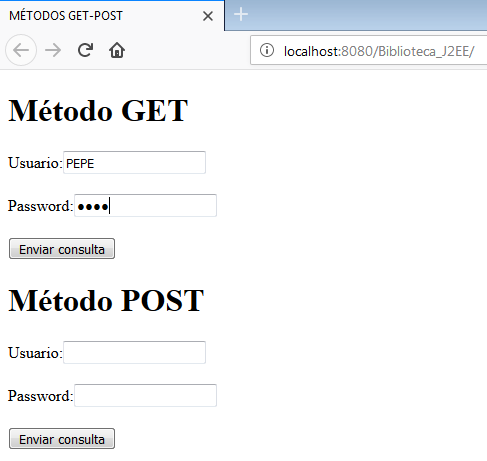
**EJERCICIO 14**

Crea el fichero error.jsp, que sustituirá a error.html en el redireccionado por parte del servlet GestorBibliotecaServlet cuando la identificación sea incorrecta. Le pasaremos el parámetro usuario para que lo muestre por pantalla, cosa que con un fichero html no se podía hacer.







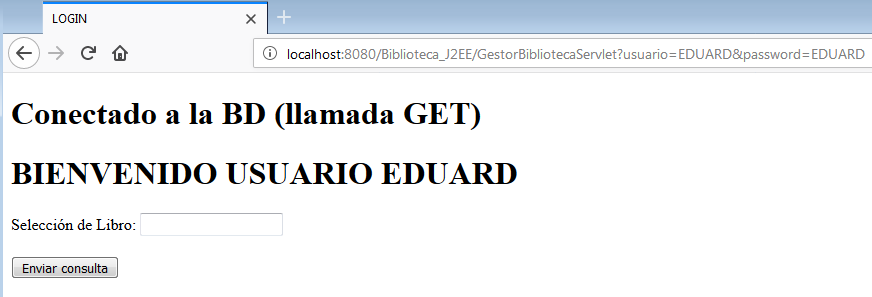




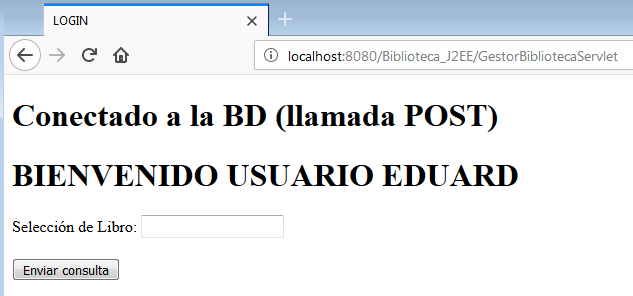
**EJERCICIO 15**

Una vez estemos conectados, nos interesa mostrar una página con un listado de libros disponibles. Para ello, en la respuesta de bienvenida del servlet GestorBibliotecaServlet, añadiremos un formulario con un campo de texto y un botón, que nos permitirá buscar una lista de libros por título. Este formulario debe apuntar a un nuevo servlet que llamaremos “ConsultaLibrosServlet”.





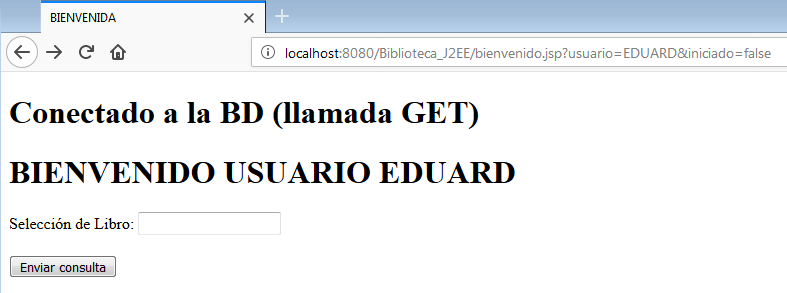


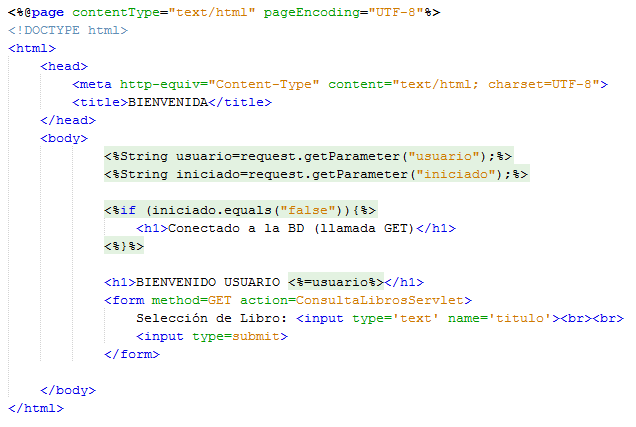


**EJERCICIO 16**

Modifica el fichero GestorBibliotecaServlet para que redireccione y delegue su funcionalidad en caso de bienvenida hacia el archivo bienvenido.jsp. Este archivo debe garantizar exactamente la misma funcionalidad del ejercicio anterior. Utilizar sendRedirect con paso de parámetros.







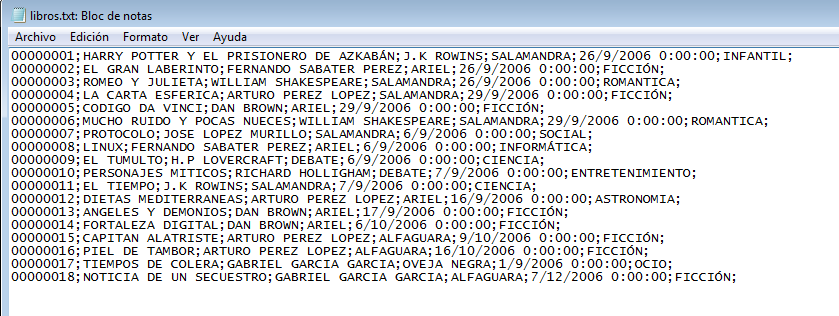


**EJERCICIO 17**

Crea el servlet ConsultaLibrosServlet. Estará preparado para recibir el título solicitado por el usuario, y buscará en el fichero de texto “libros.txt” (que se puede descargar del curso) todas las entradas que contengan dicha cadena de texto. Como respuesta, enviará al navegador un listado de libros separados por saltos de línea (etiqueta <br> de HTML). Considera los siguientes puntos:

* Utiliza el modelo processRequest para el servlet ConsultaLibrosServlet, para unificar los métodos doGet y doPost.
* Sitúa el fichero libros.txt en el directorio build/web de tu proyecto. Utiliza la api getServletContext().getRealPath("/") para obtener el path de este fichero

El contenido del fichero libros.txt es el siguiente:



Primero de todo creamos el servlet ConsultaLibrosServlet:

