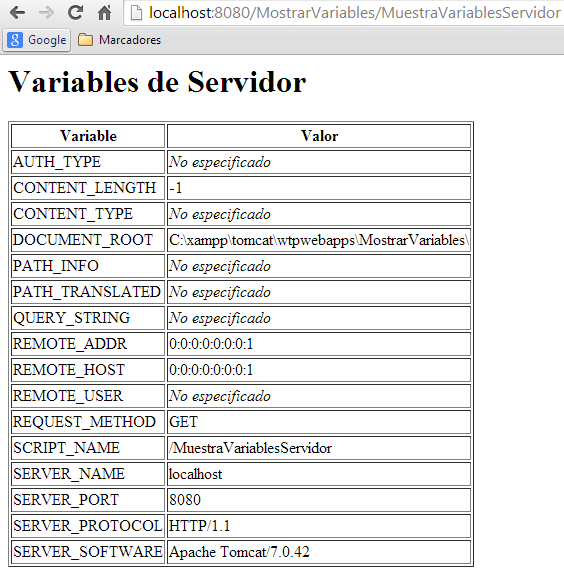
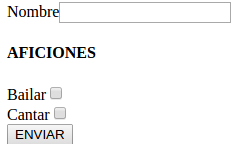
Ejercicios de desarrollo web con J2EE

1. Instalar y arrancar Tomcat. Arrancar el servicio. Acceder a la “home page” en localhost a través del puerto 8080.
2. Desde el “Home Page” de Tomcat, entrar en “First web application” → “Example app” y descargar la aplicación “sample.war”. Desplegarla (copiarla) en el directorio “webapps” de Tomcat y observar qué ocurre. Acceder a la aplicación a través de la URI “[**http://localhost:8080/sample**](http://localhost:8080/sample)”
3. Crear una aplicación J2EE (Dynamic web project) que muestre un “Hola mundo”. Observar dónde está desplegado. Crear un war y desplegarlo en el directorio webapps estándar de Tomcat.
4. Crear un servlet que muestre un formulario requiriendo un nombre y al pulsar “submit” se invoque a sí mismo, saludando al nombre introducido anteriormente.
5. Crear un servlet que muestre información de algunas de las variables manejadas por el servidor.



1. Crear un servlet que muestre en su versión get, un formulario con este aspecto



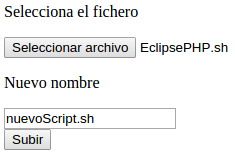
Al enviar el formulario, el mismo servlet recibirá vía post los datos, pudiendo ocurrir una de estas tres cosas:

* Que no se envíe ningún nombre, a lo que se responderá “Debes introducir un nombre”
* Que se envíe nombre, pero ninguna afición, a lo que se responderá “A <nombre> no le gusta nada”
* Que se envíe nombre y alguna/s afición/es, a lo que se contestará “A <nombre> le gusta <afición1> <afición2> …”

1. Realizar los mismos scriptlets que se realizaron en PHP: generaRadio(...), generaCheckBox(...) y generaSelect(...). NOTA: Utilizar Map<String,String> para suplir los arrays asociativos de PHP.
2. Hacer un formulario con dos campos de texto nombre, clave y país (a elegir entre España, Italia, Francia y Portugal). Al enviar se comprobará si todo ha ido bien en cuyo caso se mostrarán los datos enviados. Si hubo algún error, según la lógica de negocio (nombre no vacío, clave entre 6 y 12 caracteres), se informará del error y se repintará el formulario con los datos anteriores.

Archivos

1. Hacer un formulario que seleccione un archivo local, y lo suba a una carpeta en el servidor (NOTA: La carpeta debe existir previamente). El formulario permitirá renombrar el archivo si se desea (si se deja el nombre en blanco, mantendrá el nombre anterior).



Cookies y sesiones

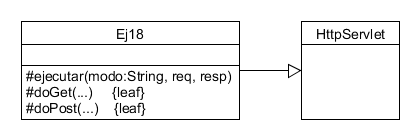
1. Hacer un programa que muestre un formulario nombre/clave. Cuando el usuario se valide con éxito (de entre dos usuarios al menos). El programa informará de hace cuántos segundos que no se validaba. A partir de un minuto, el programa no monitorizará este tiempo, y simplemente informará de que ha pasado más de un minuto. NOTA: Utilícense cookies. Puede ser útil el método *System.currentTimeMillis()*
2. Implementar, utilizando sesiones, un contador de visitas por usuario (para un par de usuarios distintos)

Acceso a SGBD y JSP

1. Para la tabla **libros(isbn, titulo, categoria)**, hacer un formulario de inserción y una página que muestre el catálogo. El primero ha de redirigir al segundo.
2. Realizar el ejercicio anterior, utilizando JSP.
3. Desdoblar a una clase externa “Libro” el objeto de negocio “Libro” y todas las operaciones de modelo relacionadas con libros. Implementar dichas operaciones con métodos estáticos.
4. Separar la clase Libro en dos clases: “Libro” y “LibroDAO”, dejando en ésta última las operaciones propias de modelo que teníamos en la otra.

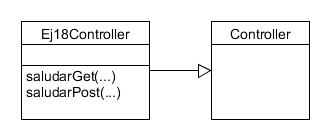
MVC

1. Crear una clase org.mvc.Controller con la siguiente estructura



* Asegurarse de no añadir la anotación @WebServlet a la clase Controller
* doGet(...) y doPost(...) lo único que hacen es llamar a ejecutar(“get”,request,response) o ejecutar(“post”,request,response) respectivamente.
* ejecutar analiza las cadenas “extras” de la URI para averiguar la acción a realizar sobre el controlador (para esta versión sólo se analizará la acción y no argumentos “extra”)
* Si no se ha indicado ninguna acción en la URI, se considerará que ésta será la cadena “index”
* Una vez hallada el String correspondiente a la acción se ejecutará utilizando “reflection” un método de este objeto cuyo nombre sea <accion>Get(request,response) o <accion>Post(request,response)
* Si el método no existe se lanzará una excepción personalizada de la clase org.mvc.exception.MetodoInexistenteException

1. Empaquetar la clase Controller y MetodoInexistenteException en un jar (empaquetando también el código fuente para no perderlo). Borrar las clases originales y añadir el jar a la carpeta lib del proyecto para poder utilizar las clases que contiene.
2. Probar la clase Controller de la siguiente manera.



* Asegurarse de anotar Ej18Controller con WebServlet para que responda a las URIs /Ej18 /Ej18/ y /Ej18/\*
* saludarGet() se activará con la uri “/Ej18/saludar” y desplegará un formulario en el que pedirá un nombre al usuario
* saludarPost() también se activará con la uri “/Ej18/saludar” (cuando se invoque vía POST) y saludará al nombre que reciba vía post

1. Añadir un par de métodos despedirseGet() y despedirsePost() similares a los anteriores pero que se despidan del usuario.

* Observar que lo único que hay que hacer para que funcione el nuevo Caso de Uso “despedir” es añadir los métodos (no hay que tocar URI’s ni anotaciones.
* Observar qué ocurre cuando se pone un nombre de accion inexistente. Capturar la excepción y mandar una página HTML de error personalizada.
* Observar qué ocurre cuando no se indica ninguna acción en concreto. Rescribir index para que ejecute por defecto saludarGet()

1. c