**Cliente – Servidor**

**HTTP:**

El protocolo HTTP es el protocolo utilizado en Internet para realizar la “conversación” entre diferentes máquinas, denominadas cliente y servidor.

Sus siglas significan HyperText Transfer Protocol, y tiene como objetivo envolver el lenguaje HTML (HyperText Markup Language) y otras especificaciones para luego ser transportado hacia su destino.

**WEB SERVER:**

Representa al servidor que se encarga de procesar los pedidos de los distintos clientes. Tiene la capacidad de recibir y administrar pedidos, para determinar la respuesta a enviar .

Cuando decimos **cliente** nos referimos a un navegador web determinado.

En nuestro caso utilizamos el Servidor Apache Tomcat.

**Apache TOMCAT como WEB SERVER:**

<http://tomcat.apache.org/>

Les adjunto una guía para instalar e ingresar a la consola de Tomcat

<https://www.youtube.com/watch?v=yunGR3hvxX4>

**Navegador Web:**

Es el otro componente de la estructura CLIENTE-SERVIDOR.

* Representa al cliente que hace pedidos al servidor.
* Tiene la responsabilidad de comprender las respuestas provenientes del Servidor, interpretarlas y visualizarlas en pantalla de modo grafico.

**Responsabilidades de un browser o Navegador**

Es interpretar lo que me envia el Servidor. Por ejemplo creamos un HTML sencillo con un párrafo “Hola Mundo”.

Opciones:

1. Descargar Sublime. Opcion elegida.
2. Realizarlo via txt.

Pasos:

1. Crear un archivo html (sublime) o txt.
2. Colocar el siguiente codigo:

<!DOCTYPE html>

<html>

<head>

<title>Hola Mundo</title>

</head>

<body>

<p>Hola Mundo</p>

</body>

</html>

1. Lo abrimos con un navegador (recomendación Google Chrome).
2. Y vemos como el navegador interpreta el html a traves del protocolo HTTP lo que envia el Servidor (en este caso nuestra PC).

**Responsabilidades del Servidor**

Responderme un HTML de acuerdo a las solicitudes que realiza el browser.

BROWSER (usuario) > SERVIDOR (servlets) > BACKEND (BBDD, APPS)

FRONTEND intermedia (servicios middleware) o backend

**Request:**

Solicitud o pedido que hace un cliente a un servidor. En ese pedido el cliente le envia datos al servidor.

A partir de este pedido el servidor recibe los datos, los procesa y genera una respuesta.

**Response:**

En general retorna un codigo HTML. Aunque puede retornar diverso tipo de información como por ejemplo un XML, imágenes. O alguna otra función especifica. Por ej: enviar un mail, etc.

En caso de querer probarlo en Netbeans dejamos los pasos para **crear un proyecto web** en dicho IDE:



**SERVLETS**

Que es un Servlet?

* Es parte de la tecnología Java junto con JSP, y pertenece a la Edicion Empresarial (JEE).
* Un servlet es un objeto java que se ejecuta en un servidor (contenedor J2EE), especialmente diseñado para ofrecer contenido dinamico desde un servidor Web generalmente HTML.
* Es una unidad de funcionalidad que se ejecuta del lado del servidor, y genera resultados que son enviados al cliente.

Hace de **intermediario** entre la **capa web** y **la capa de acceso a la BDD**.

* Un Servlet debe ser desplegado dentro de un Servlet Container para su correcto funcionamiento. No puedo colocarlo en cualquier servidor, sino en un servidor que pueda procesar codigo Java.

Por ejemplo paginas como Amazon.com esta llena de links y formularios que lo que hacen son pedidos al servidor. Tanto los links como los formularios utilizan peticiones que van a través de distintos métodos. Ej: los **links** utilizan el método **GET** y los **formularios** pueden utilizar tanto **GET** como **POST** para enviarse.

Lo que vamos a hacer ahora es crear una pagina HTML pero dentro de Netbeans para poder llamar un SERVLET que vamos a crear. Es decir vamos a conectar el mundo de HTML con el mundo de Java WEB.

Para comenzar vamos a crear primero una pagina de HTML:

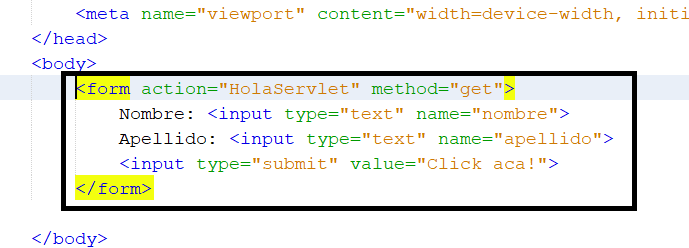


Luego vamos a insertar un formulario desde nuestro HTML para poder enviar datos a nuestro SERVLET de JAVA.

**Ejercicio A: Enviamos la info de un formulario a un SERVLET.**

**Codigo:** Clase10\_A

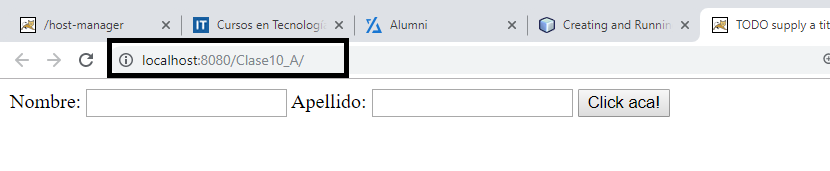
1. Nos vamos a Web pages.
2. Index.html
3. Abrimos el Index.html.
4. Creamos un formulario:



El formulario va a tener checkbox, radiobuttons, botones, listas, campos, es decir todo lo que el HTML puede contener y ser enviado a un SERVLET.

Los dos métodos principales son get y post. En este caso usamos **get** y la información va a viajar en la barra que tienen los browser arriba. En cambio si usamos el método **post** el envio se realizara via headers (cabeceras) en el cuerpo del request. En **action** del htmldebo colocar el **serlvet** de java que va a recibir toda esta información.

1. Ejecutamos el proyecto. Vemos que se levanta en el servidor de Tomcat.

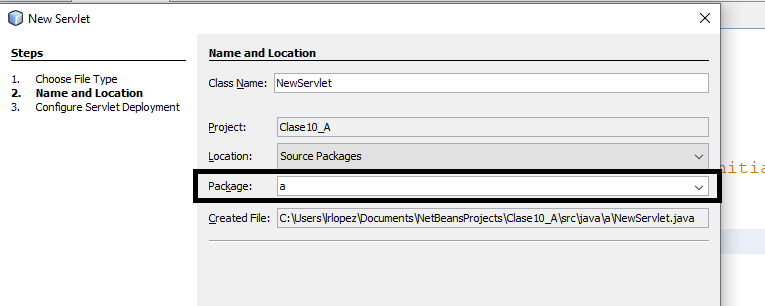


Esto se puede abrir desde cualquier browser copiando la url.

Esta pagina esta generando un pedido (request) a un servidor.

**EJEMPLO DE CREACION DE SERVLET.**

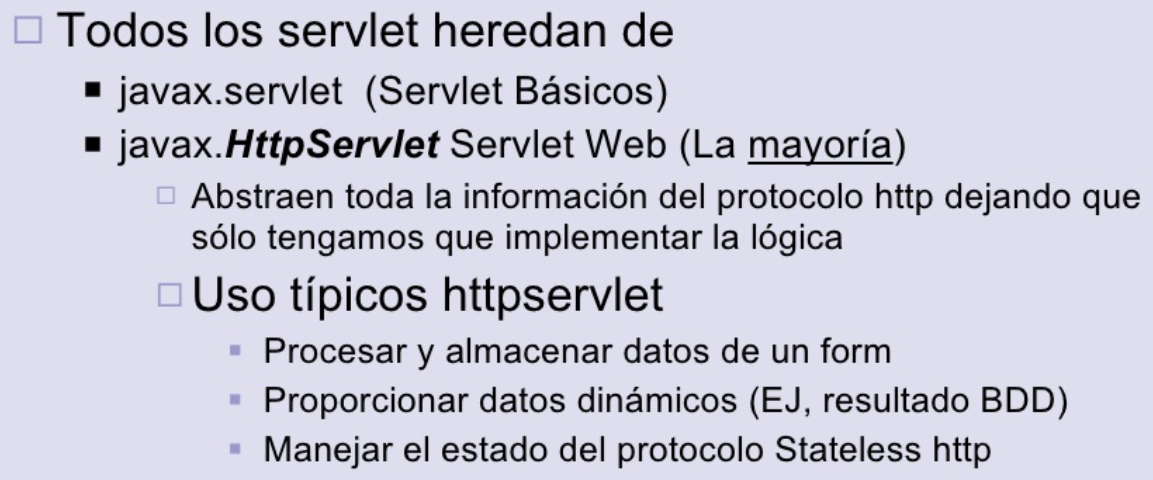
1. Source Packages => New => Servlet
2. Importante: Los servlet tienen que tener un paquete. Le ponemos cualquier nombre.



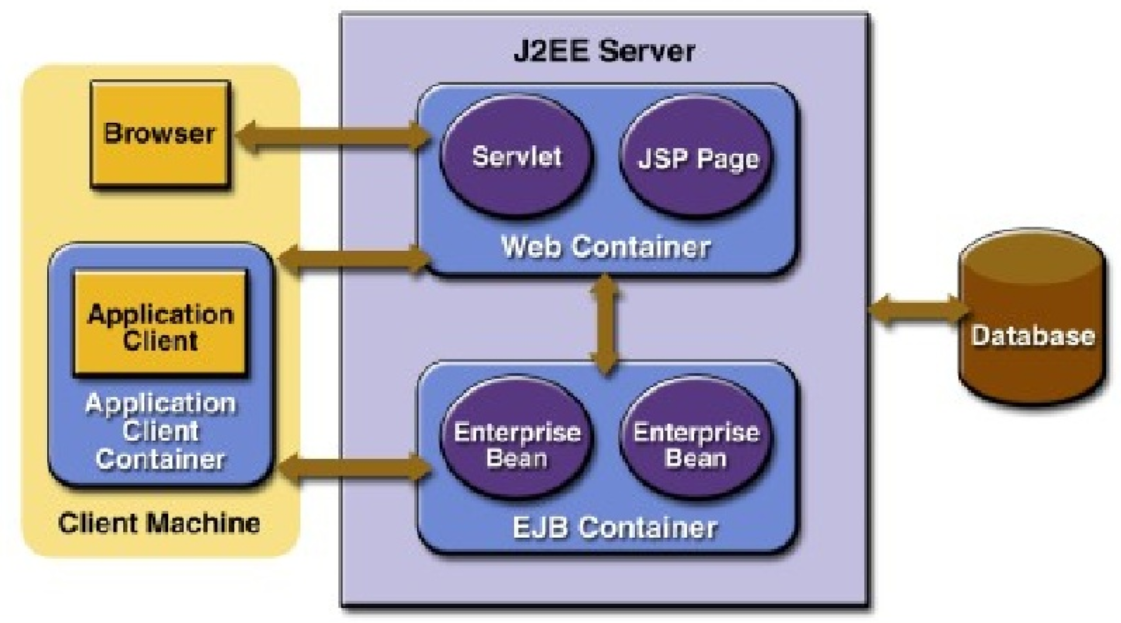
1. ClassName: HolaServlet. Igual a como lo habíamos llamado.
2. FINISH.

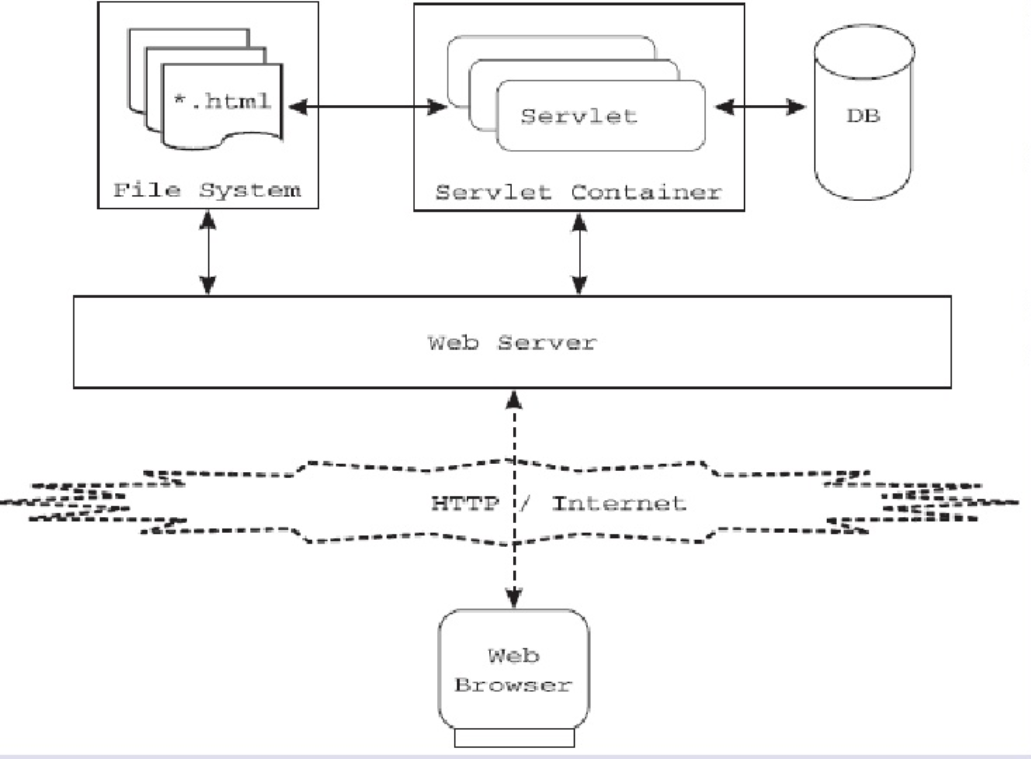
#### 1.1.2. Arquitectura del paquete servlet

Dentro del paquete **javax.servlet** tenemos toda la infraestructura para poder trabajar con servlets. El elemento central es la interfaz **Servlet**, que define los métodos para cualquier servlet. La clase **GenericServlet**es una clase abstracta que implementa dicha interfaz para un servlet genérico, independiente del protocolo. Para definir un servlet que se utilice vía web, se tiene la clase **HttpServlet** dentro del subpaquete **javax.servlet.http**. Esta clase hereda de *GenericServlet*, y también es una clase abstracta, de la que heredaremos para construir los servlets para nuestras aplicaciones web.



SERVLET Y CONTENEDORES





1. En el método doGet (ya que estamos usando este método) colocamos el siguiente codigo:

//Ejercicio 1

response.getWriter().append("Hola Mundo"); //append: concatena el string al printWriter

//getWriter: retorna un printwriter el

//cual podra enviar una cadena de caracteres al cliente

Esto lo que hace es que cuando oprimimos el botón desde nuestra pagina html, el servlet devuelve un “Hola Mundo”.

**Ejercicio 2**: **Objetivo**: Que el nombre y apellido que ingresa el usuario desde el html sea lo que salga por pantalla, es decir lo que devuelva el Servlet.

**Codigo**: Mismo codigo. Clase10\_A



Para hacer esto debemos prestar atención a los objetos que recibe como parámetro el método doGet=> Esto es el **request** que va a tener todos los datos del usuario y el **response** el cual me va a permitir responder desde el servidor. Son objetos de tipo interfaz.

1. Hago una modificación al doGet por lo siguiente:

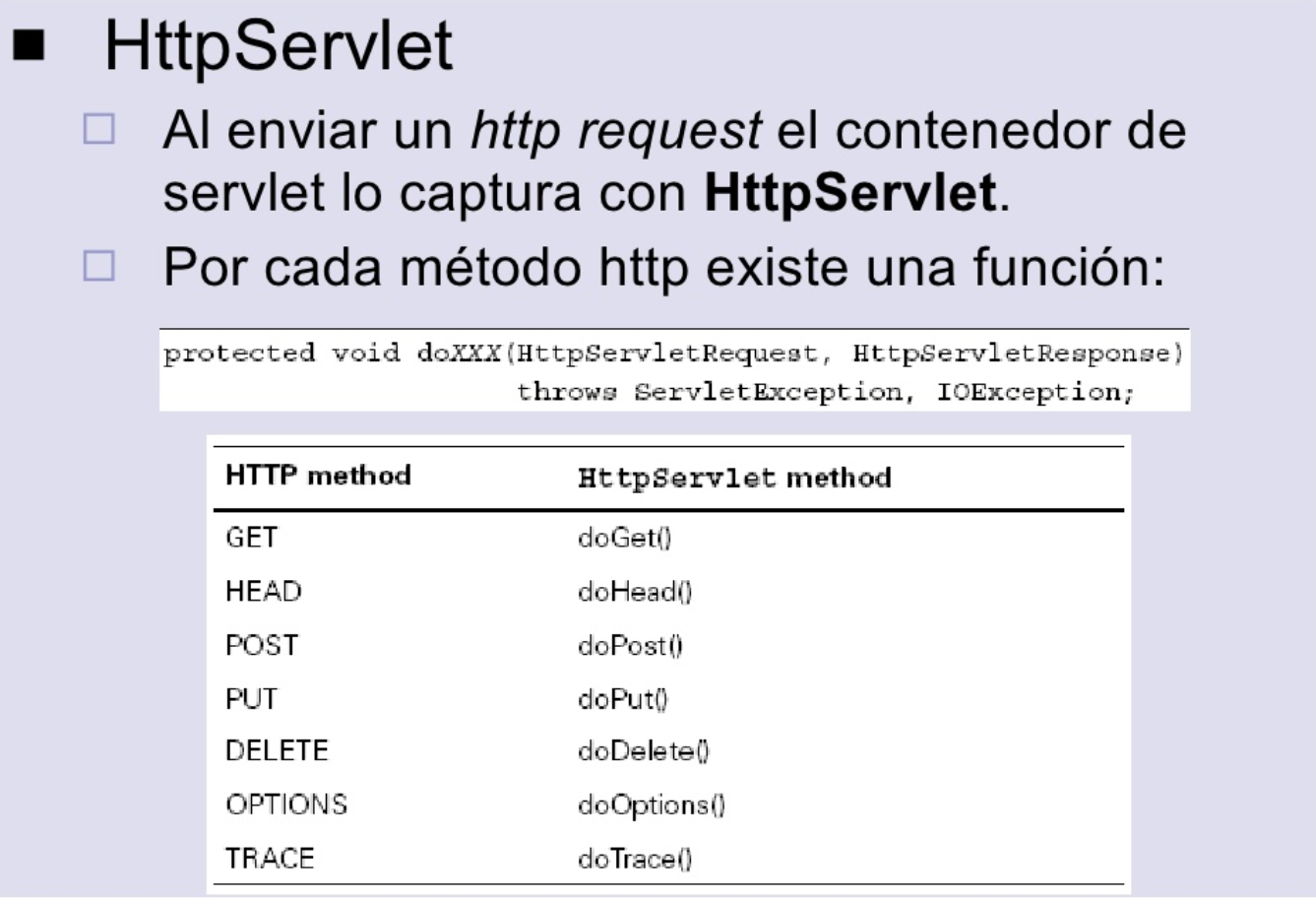
//Ejercicio 2

String nombre = request.getParameter("nombre"); //obtiene el dato del nombre en html

String apellido = request.getParameter("apellido"); //idem apellido

response.getWriter().append(nombre + " " +apellido); //el servlet devuelve el nombre y apellido

El getWriter() es un método de la interface ServletResponse y devuelve un objeto del tipo PrintWriter en donde contiene toda la text-data (character data). Entonces básicamente responde con el texto del nombre y apellido.



**Ejercicio 3: Ejercicio de la clase 10**

Consignas:



Codigo: **Clase10\_B**



1. Construimos el Form en index.html.

Incluimos el siguiente codigo en el Body:

<h1>Formulario</h1>

<form action="PersonaServlet" method="post">

<label>Nombre:<br> <input type="text" name="nombre"></label> <br>

<label>Apellido: <br> <input type="text" name="apellido"></label><br>

<label>Domicilio:<br> <input type="text" name="domicilio"></label><br>

<label>Pais: <br> <input type="text" name="pais"></label><br><br>

<input type="submit" name="boton" value="Registrarme!">

</form>

1. Agrego en el método **doGet** lo siguiente:

String respuesta = request.getParameter("nombre") + "\n" + request.getParameter("apellido") + "\n"

+ request.getParameter("domicilio") + "\n" + request.getParameter("pais");

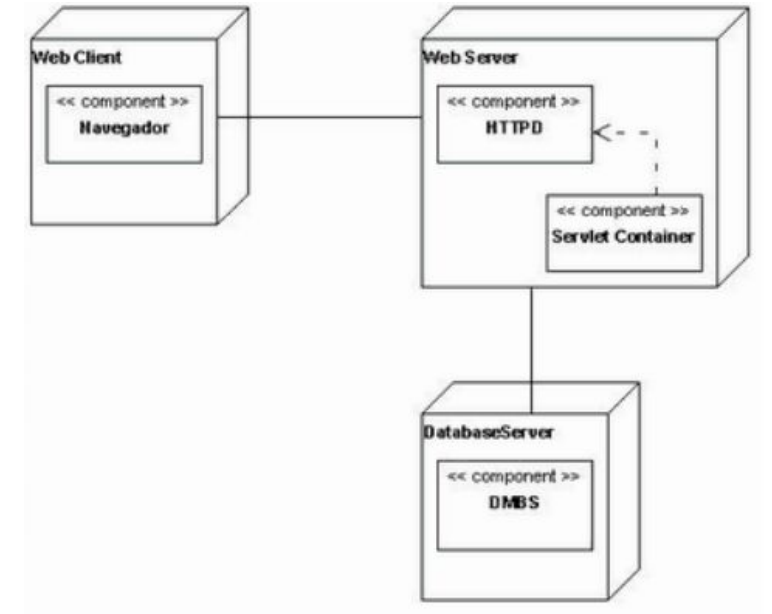
response.getWriter().append(respuesta);

1. Cerciorarse que en el metodo **doPost** tengamos lo siguiente (el metodo doPost esta llamando a doGet):

doGet(request, response);

**Observaciones**:

Cabe destacar que si quiero registrar los datos en la BDD puedo hacer referencia a **PersonaDAO** creando un objeto de tipo **Persona**. Eso lo vamos a ver en la próxima clase.



El método **doPost** procesa request de tipo **post**. Solo que en el caso previo la inteligencia la derivo al método **get**.

**Ejercicio 4:** El objetivo es cuando al presionar el botón el servidor nos devuelva el siguiente mensaje de salida.

# Bienvenido a Java EE.

# Hoy es (agrego fecha y hora de hoy)

# y ha creado su primer servlet

**Codigo: Clase10\_C**

****

1. Ingreso el siguiente botón en mi HTML (index).

<body>

<form action="Servlet" method="get">

<input type="submit">

</form>

</body>

1. Crear un servlet llamado Servlet y en el método **processRequest** ingresar el siguiente codigo a la altura del BODY:

out.println("<h1>Bienvenido a Java EE. Hoy es "+Calendar.getInstance().getTime()+ "y ha "

+ "creado su primer servlet</h1>");

**Ejercicio 5: Objetivo:** Ver cada uno de los componentes HTML para conocer los elementos mas importantes en dicho lenguaje. Ver el siguiente codigo.

Codigo: **Clase10\_D**.



1. Plantear el index.html de acuerdo al codigo previo.

**Nota:** <input type=”hidden”>

El elemento input, teniendo el valor "hidden" en su atributo type, representa cualquier cadena de texto arbitraria que no está pensada para ser vista o editada por el usuario.

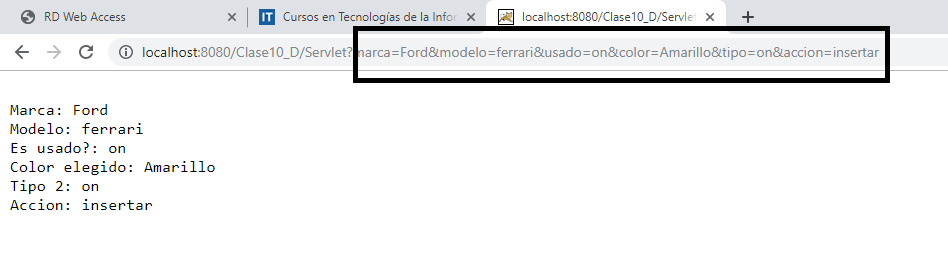
**Metodo GET:** Envia la información del usuario adjuntada al request. La URL y la información codificada es separada con el carácter”**?”**.

Esta información es enviada utilizando el mencionado **QUERY\_STRING header** y será accesible a través del QUERY STRING variable de entorno a través de los métodos **getQueryString**() y **getParameter**() del **objeto** **request**.

**Metodo POST:** Mas confiable que el GET para el envio de información. La información es enviada a través de un **mensaje.**

**JSP** maneja este tipo de **requests** a través del **getParameter()** para leer parámetros simples y **getInputStream()** para leer data binary stream que viene del cliente.

**Querystring**: En el ejercicio vemos que todo lo que esta después del ? es lo que enviamos via método Get con el formulario. Lo enviamos desde el cliente al servidor. Está compuesta de pares de nombre/valor separados por un "&".



Mismo ejercicio usando **POST**

Modificamos el index.html (method=POST) y en el Servlet en el método doPost() borramos la llamada al método **processRequest()** y llamamos al método **doGet().** Para que la inteligencia de respuesta continue estando allí.

La diferencia que vemos ahora es que la información va inmersa en el cuerpo del **request. No figura mas en el Address bar.**

**Web.xml: Descriptor de Despliegue.**

Esta localizado en la ruta YOURPROJECT\web\WEB-INF.

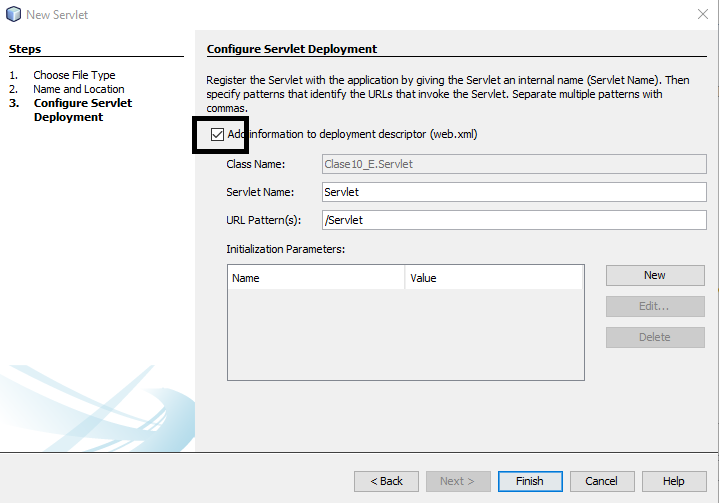
Es opcional a partir de Java EE 6. Por default no es cargado en Netbeans. Hay que crearlo en caso de necesitarlo.

Creacion:

New -> Other-> Web -> Standard Deployment Descriptor (web.xml) -> Next -> Finish

El archivo web.xml se denomina descriptor de despliegue y es opcional cuando se utilizan servlet. Con él se define su nombre y se identifica al paquete y clase que lo contiene, entre otras cuestiones. Tambien tiene otros usos, tales como los filtros, los parametros de contexto, los parametros iniciales, etc…

También se puede crear cuando se crea un Servlet:

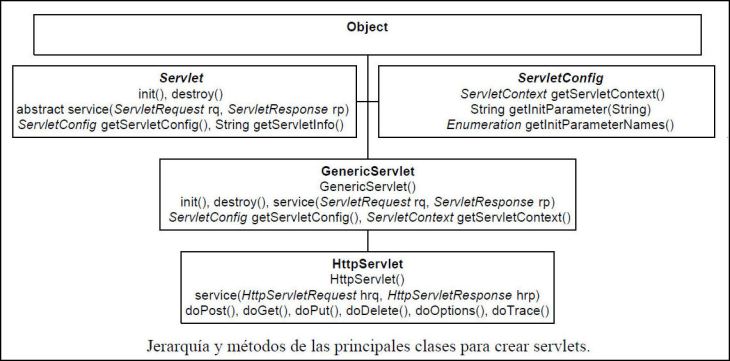


**Ciclo de Vida del Servlet:**

**El API Servlets y el ciclo de vida.**

Los servlets usan clases e interfaces de dos paquetes:

1. javax.servlet que contiene clases para servlets genéricos (independientes del protocolo que usen) y,
2. **javax.servlet.http** (que añade funcionalidad particular de http). El nombre javax indica que los servlets son una extensión.

[](https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/image1-53/)

Los servlets no tienen el método main() como los programas Java, sino que se invocan unos métodos cuando se reciben peticiones. A esta metodología se le llama ciclo de vida de un servlet y viene dado por tres métodos: **init**, **service**, **destroy**:

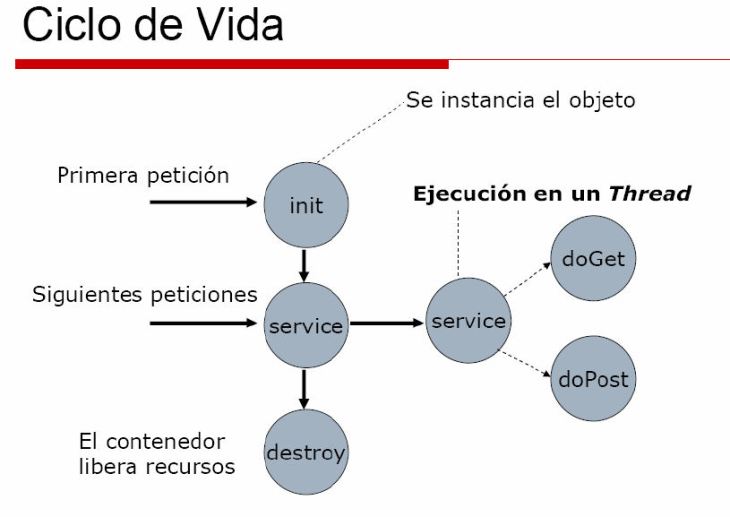
**INICIALIZACIÓN**: Una única llamada al método “init” por parte del servlet. Incluso se pueden recoger unos parámetros concretos con “getInitParameter” de “ServletConfig” iniciales y que operarán a lo largo de toda la vida del servlet.

**init(ServletConfig config)**: Es el método utilizado para crear una nueva instancia del servlet (análogo al constructor). [Ver el ciclo de vida](https://users.dcc.uchile.cl/~jbarrios/servlets/vida.html). Este método puede ser sobreescrito para realizar tareas como crear una conexión a una BD que se mantendrá mientras el servlet se mantenga cargado y puede ser utilizada por cada petición. **ServletConfig** contiene los parámetros de inicialización que entrega el servidor al servlet.

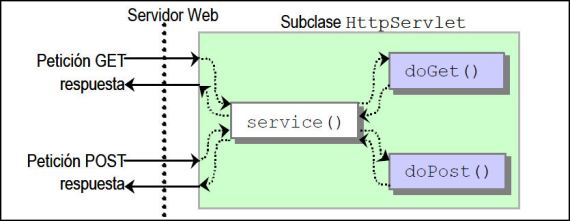
**SERVICIO**: una llamada a service() por cada invocación al servlet para procesar las peticiones de los clientes web.

**service(ServletRequest req, ServletResponse res)**: Este método es el que se llama cuando se recibe una petición de un cliente y en su implementación normal para HTTP verifica el tipo de solicitud GET, POST, etc. y la redirige a los métodos respectivos. En general no es necesario reimplementar este método.

**DESTRUCCIÓN**: Cuando todas las llamadas desde el cliente cesen o un temporizador del servidor así lo indique o el propio administrador así lo decida se destruye el servlet. Se usa el método “destroy” para eliminar al servlet y para “recoger sus restos” (garbage collection).

[](https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/image2-45/)

Cada vez que el servidor pasa una petición (distinta a la primera) a un servlet se invoca el método service(), este método habrá que sobreescribirlo (override). Este método acepta dos parámetros: un objeto petición (request) y un objeto respuesta. Los servlets http, que son los que vamos a usar, tienen ya definido un método service() que llama a doXxx(), con Xxx el nombre de la orden que viene en la petición al servidor web. Estos dos métodos son doGet() y doPost() y nos sirven para atender las peticiones específicamente provinientes de métodos GET o POST respectivamente,

[](https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/image1-51/)

De esta manera, una vez se carga el servlet, es muy eficiente, pues sólo hay una copia cargada en memoria (se ejecutan uno o varios hilos), no hay que crear nuevos objetos (un solo objeto servlet), y tiene persistencia: puede guardar información entre peticiones, como contadores o conexiones a una base de datos. Esto último puede ser muchísimo más eficiente hacerlo en el método init que abrir y cerrar la conexión con la base de datos en cada petición.

**Ejercicio 6:**

Usamos el mismo formulario del ejercicio anterior y utilizamos el método **processRequest** para ver como se arma un html de respuesta. Incluimos los datos de respuesta y la lógica en el mismo.



**Fuentes de Informacion (No sustituyen sino complementan al Alumni):**

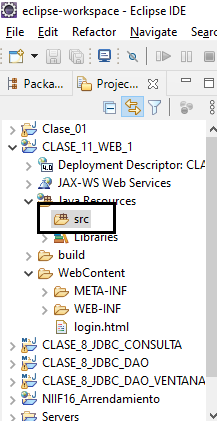
<https://flanagan.ugr.es/docencia/2005-2006/2/servlets/ejemplos/Alumno.java>

<https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/>

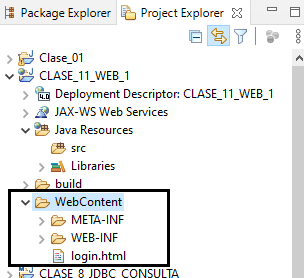
<https://users.dcc.uchile.cl/~jbarrios/servlets/clases.html>

**CLASE\_11\_EJEMPLO\_1 (ECLIPSE)**

1. **Creamos el proyecto**. File=>New=>DynamicWebProject.
2. ProjectName: CLASE\_11\_WEB\_1.=>NEXT>NEXT>y le decimos que **genere el archivo web.xml (deployment description)**=>FINISH.
3. Nuestro **codigo fuente** estará en:



1. Los **archivos web** estarán en:



**WebContent** es la **raíz**. A medida que se vayan creando paginas nuevas en subcarpetas representa una “barra” a partir del WebContent.

1. **web.xml:** es un archivo importante, aquí se van a ir definiendo los servlets automáticamente y el ejecutable de nuestro proyecto.
2. Creamos **login.html:**



Y agregamos el siguiente codigo:

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Login Web</title>

<script type="text/javascript" languaje="JavaScript" src="js/codigo.js"></script>

<link rel="stylesheet" href="css/style.css">

</head>

<body onload="carga();">

<p>HTML</p>

<p>CSS</p>

<p>JS</p>

<p>JSP</p>

<p>SERVLETS</p>

<p>Servidor Web </p>

<p>Servidor de Aplicaciones </p>

<form action="LoginWeb.jr" method="POST" onsubmit="return validaDatos();">

<table class="center">

<tr>

<td colspan="2" style="font-weight: bold; size:20px;">Ingresa tu Usuario y Contraseña </td>

</tr>

<tr>

<td style="font-weight: bold; size:12px; color: blue; text-align:right;" >Usuario: </td>

<td> <input type="text" name="usuario" id="usuario"> </td>

</tr>

<tr>

<td style="font-weight: bold; size:12px; color: blue; text-align:right;">Password: </td>

<td> <input type="password" name="password" id="password" > </td>

</tr>

<tr>

<td> <input type="button" value="Cancelar" class="boton"> </td>

<td> <input type="submit" value="Aceptar" class="boton"> </td>

</tr>

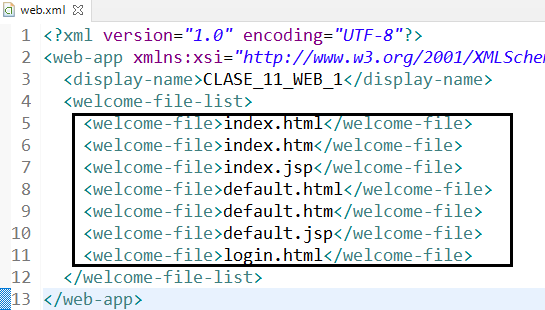
</table>

</form>

</body>

</html>

1. **Ejecutamos: web.xml** indicamos con que archivo front se ejecutara nuestro proyecto.



Borramos todos los archivos y agregamos **login.html**.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"* id=*"WebApp\_ID"* version=*"4.0"*>

<display-name>EJERCITACION</display-name>

<welcome-file-list>

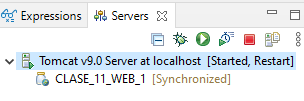
<welcome-file>login.html</welcome-file>

</welcome-file-list>

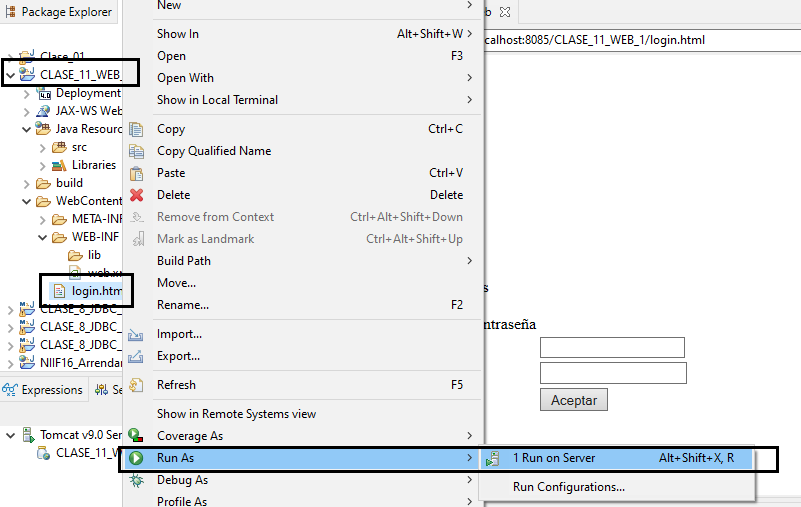
</web-app>

1. **DEPLOY y EJECUCION (forma 1)**

Click sobre **login.html=>run as => Run on Server**. (Previamente tenemos que tener instalado Tomcat y levantarlo desde Eclipse de forma tal que lo veamos asi).



Luego:



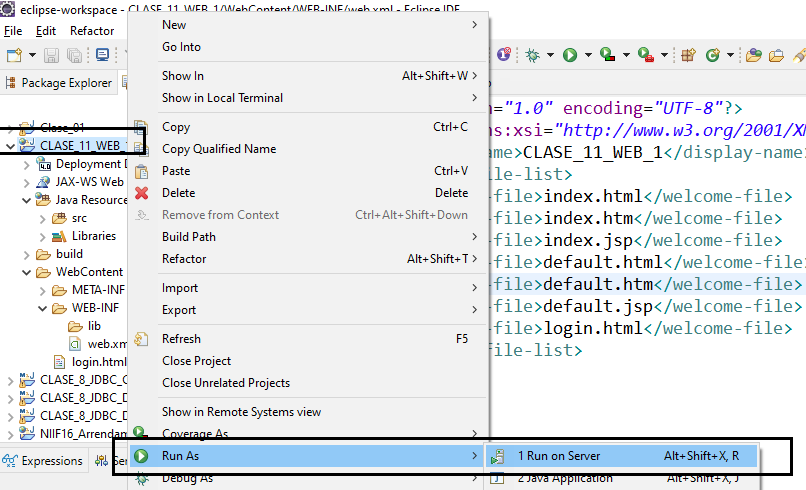
Vemos la salida desde ECLIPSE o bien <http://localhost:8085/CLASE_11_WEB_1/login.html>

O bien <http://localhost:8089/CLASE_11_WEB_PRACTICA_1/login.html>

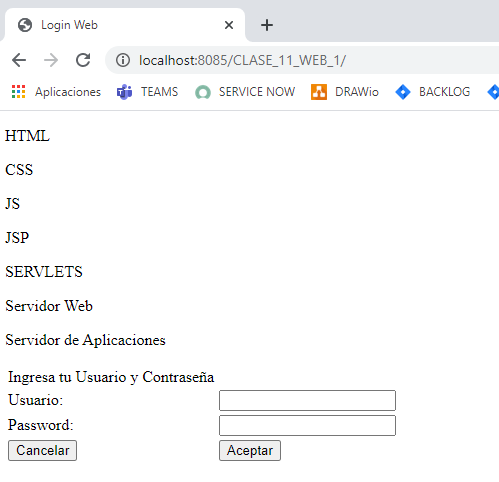
dependiendo el **puerto** configurado.

1. **DEPLOY y EJECUCION (forma 2)**:

Teniendo configurado el **login.html** en el **web.xml**, ejecutamos previo reinicio del servidor de Tomcat:



Desde el browser:



1. Que pasa si comentamos la siguiente línea en el **web.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"* id=*"WebApp\_ID"* version=*"4.0"*>

<display-name>EJERCITACION</display-name>

<welcome-file-list>

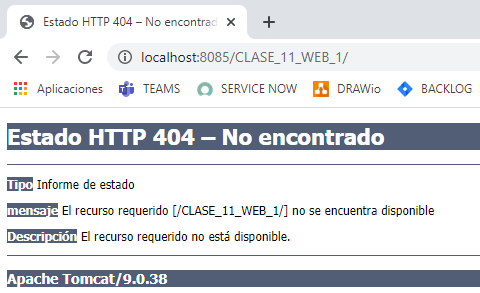
<welcome-file>login.html</welcome-file>

</welcome-file-list>

</web-app>

**Reiniciamos** el servidor.

Obtendremos un **404**.



**Causa:** En la lista de archivos del **web.xml** no encontró ningún html o jsp ejecutable.

1. **css/style.css**:

table.center{

margin-left:auto;

margin-right:auto;

}

.boton{

font-size: 14px;

color: blue;

}

.parrafo{

font-weight: bold;

font-size: 20px;

}

**Chequeemos que este linkeado al html.**

<link rel=*"stylesheet"* href=*"css/style.css"*>

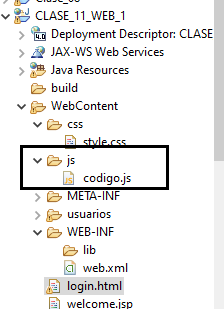
Centra el formulario y le agrega color azul y negrita a los labels.

1. Para que el **cursor** se focalice en el campo **usuario** teniendo el atributo **onload** en **login.html**

</head>

<body onload="carga();">

ll. **js/codigo.js** (JavaScript Source File)



Linkeamos en el **head** de **login.html**

<script type="text/javascript" languaje="JavaScript" src="js/codigo.js"></script>

**js/codigo.js => autofocus + validaciones**

function **carga()** {

document.getElementById("usuario").value = "";

document.getElementById("usuario").focus();

}

//Validamos si no ingreso el usuario y/o la password

function validaDatos() {

var usuario = document.getElementById("usuario");

var password = document.getElementById("password");

if (

(usuario.value == "" || usuario.value == '' || usuario.value == null) &&

(password.value == "" || password.value == '' || password.value == null)

) {

alert("Ingresa tu usuario y password");

usuario.focus();

return false;

}

if (usuario.value == "" || usuario.value == '' || usuario.value == null) {

alert("Ingresa tu usuario");

usuario.focus();

return false;

} else if (password.value == "" || password.value == '' || password.value == null) {

alert("Ingresa tu password");

password.focus();

return false;

}else{

return true;

}

}

//Guardamos, actualizamos y borramos

function guardar(){

var forma = document.getElementById("formaUsuarios"); //form."...." hago referencia a los atributos del formulario, ejmplo a "action".

forma.action = "adminUsuarios.servlet"; //referencia al formulario atributo action=>referencia al Servlet

var accion = document.getElementById("accion");

accion.value = "guardar";

forma.submit(); //lo submitea

}

function actualizar(){

var forma = document.getElementById("formaUsuarios"); //form."...." hago referencia a los atributos del formulario, ejmplo a "action".

forma.action = "adminUsuarios.servlet"; //referencia al formulario atributo action=>referencia al Servlet

var accion = document.getElementById("accion");

accion.value = "actualizar";

forma.submit(); //lo submitea

}

function borrar(){

var forma = document.getElementById("formaUsuarios"); //form."...." hago referencia a los atributos del formulario, ejmplo a "action".

forma.action = "adminUsuarios.servlet"; //referencia al formulario atributo action=>referencia al Servlet

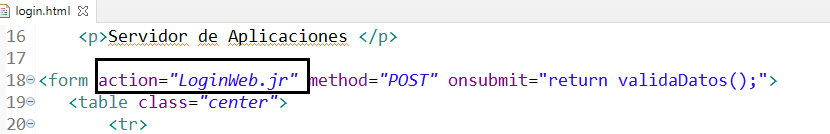
var accion = document.getElementById("accion");

accion.value = "borrar";

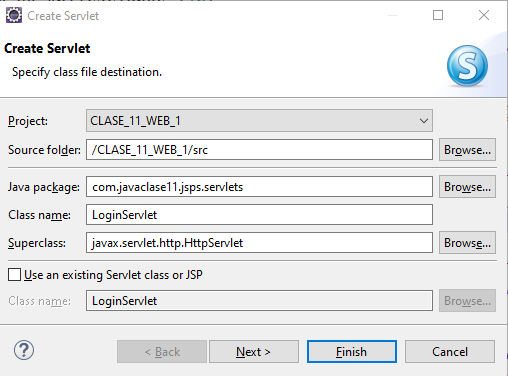
forma.submit(); //lo submitea

}

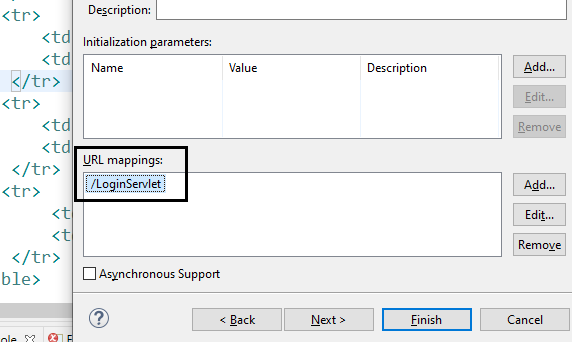
1. **CREAMOS** un **SERVLET** que recibirá el request LoginWeb.jr del formulario **login.html** (action).



ClASE\_11\_WEB\_1=> Java Resources=>src=> botón derecho => new=> Servlet



Next=> damos doble click en **url Mappings**

****

Y lo cambiamos por **/LoginWeb.jr**

**FINISH**

1. Omitimos la annotation

@WebServlet("/LoginWeb.jr"). la cual es una opción para reconocer un servlet.

La opción B es incluirlo en el **web.xml** como haremos en nuestro caso.

Como dijimos previamente

**BORRAMOS o COMENTAMOS EL ANOTATION** del servlet @WebServlet("/LoginWeb.jr")

1. Y agregamos en el **web.xml** el **servlet** lo marcado en negrita:

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

id="WebApp\_ID" version="4.0">

<display-name>CLASE\_11\_WEB\_1</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

<welcome-file>login.html</welcome-file>

</welcome-file-list>

**<servlet>**

**<!-- Ruta de nuestro Servlet con el nombre de nuestro Servlet -->**

**<display-name>LoginServlet</display-name>**

**<servlet-name>loginServlet</servlet-name>**

**<servlet-class>com.javaclase11.jsps.servlets.LoginServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>loginServlet</servlet-name>**

**<url-pattern>/LoginWeb.jr</url-pattern> <!-- Es donde va a estar escuchando las solicitudes o requests. -->**

**</servlet-mapping>**

</web-app>

1. Solucionamos los errores que se observan en el Servlet si los hubiere.



En este caso, no esta encontrando el path del Servlet.

Para solucionarlo agregamos la librería de Apache Tomcat a nuestro proyecto en caso de no tenerla agregada ya.

Click derecho sobre nuestro proyecto=>properties=>Java Build Path=>Libraries=>Add Library=>ServerRuntime =>Apache Tomcat9.0 (o la versión correspondiente)

1. Probamos <http://localhost:8089/CLASE_11_WEB_1/login.html>
2. Una vez invocado nuestro servlet, el metodo **doPost()** será el llamado. Incluimos el siguiente codigo:

String usuario = request.getParameter("usuario");

String password = request.getParameter("password");

//validamos que el usuario ingresado sea "user" password "1234"

if(usuario.equals("user")&& password.equals("1234")) {

System.out.println("Welcome");

//Seteamos el atributo al request

request.setAttribute("usuarioLogueado", usuario);

//redireccionamos a welcome.jsp

request.getRequestDispatcher("welcome.jsp").forward(request, response);

}else {

System.out.println("Error en credenciales de acceso...");

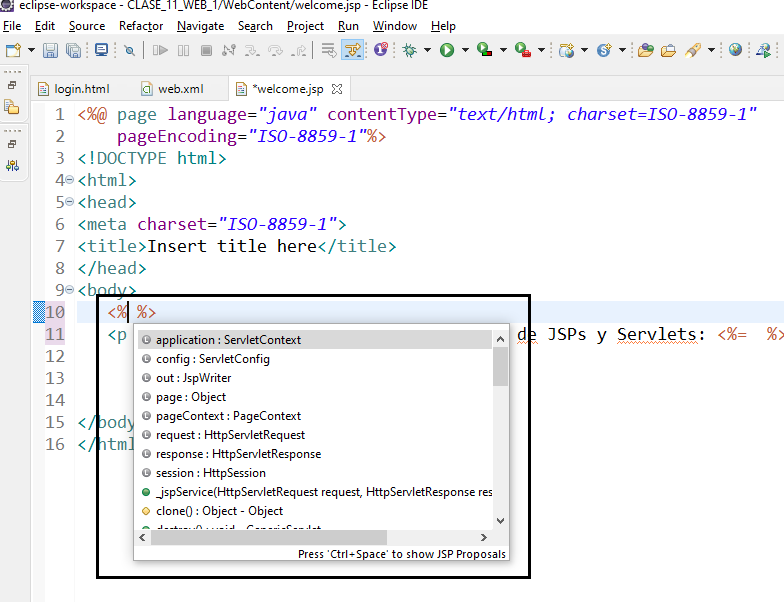
}

Y creamos un JSP para que reciba el redireccionamiento (getRequestDispatcher().forward) del Servlet.

1. **JSP. Creacion**

**WebContent=>**Click derecho=> New=> JPS File=>welcome.jsp=> next=> finish

Tengamos en cuenta que en los servlet /jsp tenemos disponibles los siguientes objetos implicitos



<https://emergentesdev.wordpress.com/2013/07/24/objetos-implicitos-en-jsp/>

1. **welcome.jsp**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

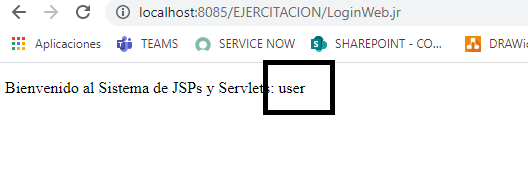
<body>

<p class="parrafo">Bienvenido al Sistema de JSPs y Servlets: <%= request.getAttribute("usuarioLogueado") %></p>

</body>

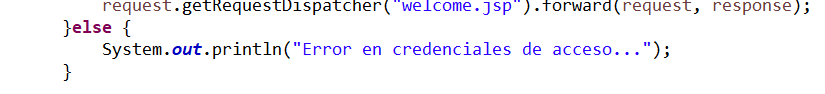
</html>

1. Redireccionamos y mostramos el nombre del usuario desde el jsp luego de haber pasado la validacion de credenciales correctas.

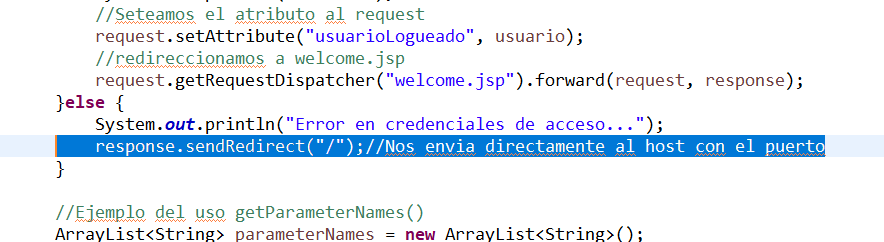


1. Cuando no pasa la **validacion** en lugar de mostrar por consola, redireccionamos desde el Servlet a welcome.jsp y mostramos el mensaje de error:

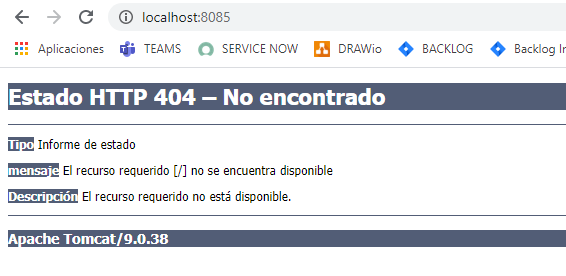
Al siguiente codigo:



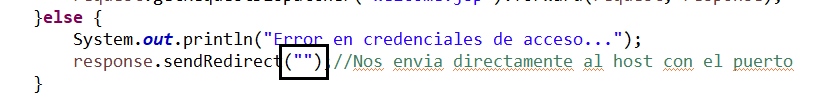
Le agregamos:



Obtenemos un 404 nos envia al host.



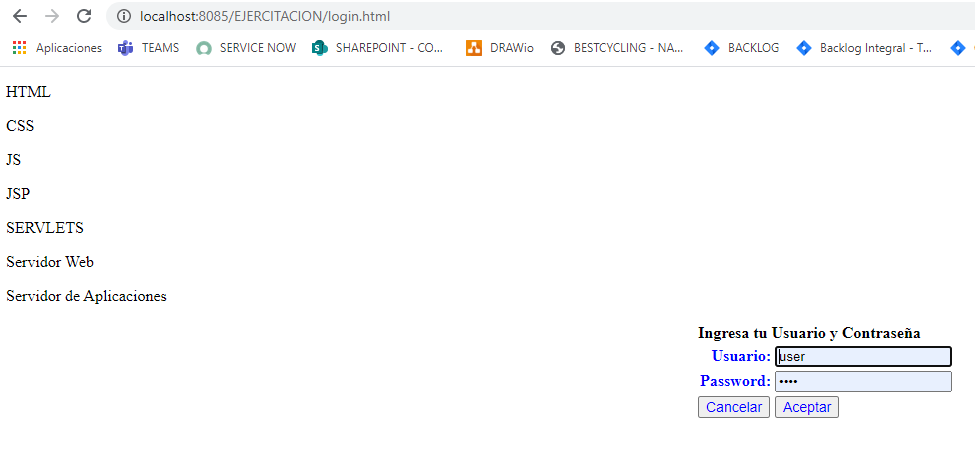
Si no le ponemos nada y si se ingresan mal las credenciales, nos regresa al **contexto** (**WebContent** o **raíz** principal web):





Y si ponemos

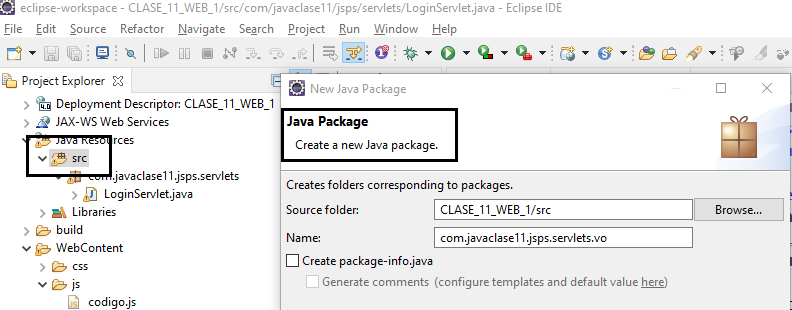
response.sendRedirect("login.html");//Nos envia a la pagina de inicio (recordar reiniciar el Server para que se vea la diferencia)



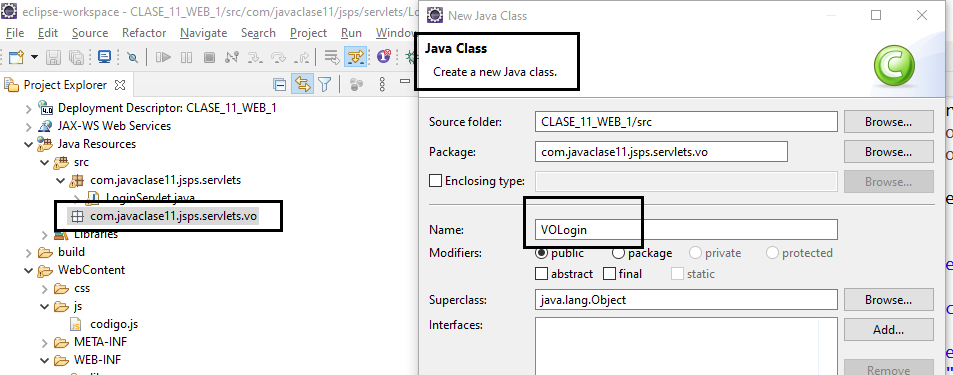
1. **DESAFIO 2** : Guardar el usuario en la sesión para que desde otra pagina podamos recuperarlo

Para ello creamos un simple Java bean:

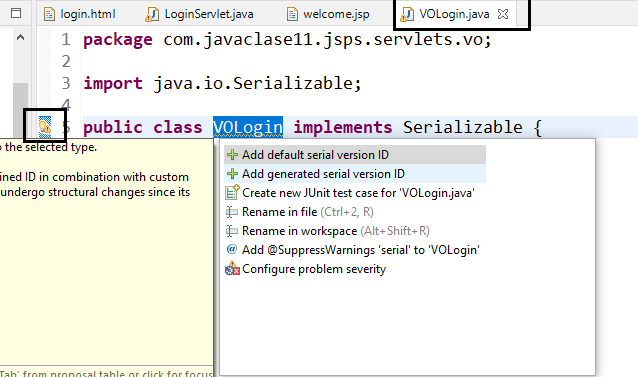
1. Creamos un nuevo paquete en **src**:



1. Creamos la clase VOLogin que actuara como un Java bean.



Implementamos Serializable



Alt+shift+s => getters y setters

De forma tal que el bean nos queda:

package com.javaclase11.jsps.servlets.vo;

import java.io.Serializable;

public class VOLogin implements Serializable {

/\*

\* Serializar consiste en convertir un objeto en una sucesion de bytes.

\* Con el objetivo de poder almacenar el mismo en un medio de almacenamiento (dis

\* co duro, pen drive, BBDD, etc) y en el futuro restaurar o recomponer ese

\* objeto al estado en el que se encontraba cuando lo serializamos.

\* Serializar = convertir en bytes.

\*

\* Tambien como segundo objetivo se puede distribuir ese objeto a ordenadores

\* remotos y desde los mismos poder reestablecer el objeto en el mismo estado

\* en el que se encontraba cuando se serializo.

\*

\* Interfaz Serializable: Las clases que la implementen podran generar instancias

\* u objetos suceptibles de convertir en bytes para transferir los mismos.

\*

\* Clase ObjectOutputStream y ObjectInputStream: La 1ra permite construir un

\* flujo de datos a traves de la cual es posible transferir un objeto desde

\* nuestro programa de Java hasta un medio de almacenamiento o hacia un ordenador

\* remotor (desde dentro hacia afuera). La 2da hace lo contrario, crea un flujo

\* de datos a partir de lo cual es posible que viaje un objeto pero desde fuera

\* hacia adentro de nuestro programa. Y desde ahi recomponer esos bytes que han entrado

\*

\* Metodos writeObjet() y readObject(): El 1ro crea el objeto (sucesion de bytes)

\* para lanzarlo de adentro hacia afuera y readObject() hace lo contrario.

\* Lee la sucesion de bytes.

\*

\*/

private static final long serialVersionUID = -3617155887382263126L;

private String usuario;

private String nombre;

private Integer edad;

private String password;

public String getUsuario() {

return usuario;

}

public void setUsuario(String usuario) {

this.usuario = usuario;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public Integer getEdad() {

return edad;

}

public void setEdad(Integer edad) {

this.edad = edad;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public static long getSerialversionuid() {

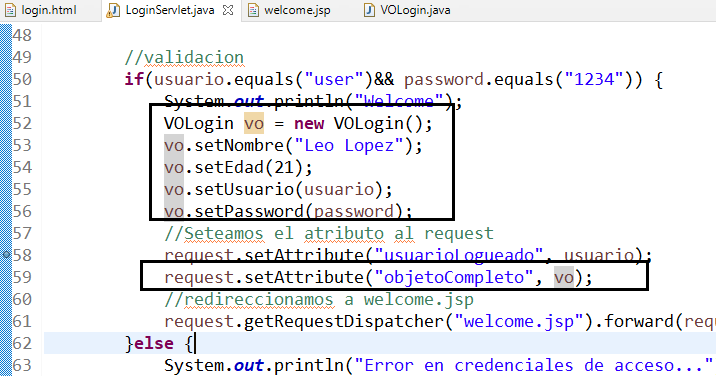
return serialVersionUID;

}

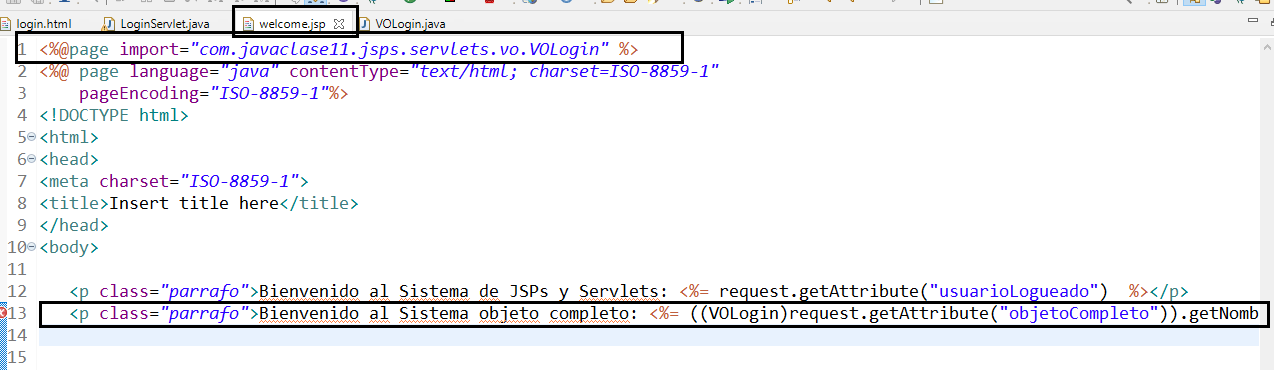
}

1. **Desde LoginServlet** Creamos un objecto y lo guardamos en el request:

Hacemos los siguientes agregados al servlet.



Y desde **welcome.jsp** lo recuperamos**:**

****

Codigo:

<%@page import=*"com.javaclase11.jsps.servlets.vo.VOLogin"*%>

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<p class=*"parrafo"*>Bienvenido al Sistema de JSPs y Servlets: <%= request.getAttribute("usuarioLogueado") %></p>

<p class=*"parrafo"*>Objeto completo: <%

VOLogin aux = ((VOLogin)request.getAttribute("objetoCompleto"));

out.print(aux.getNombre());

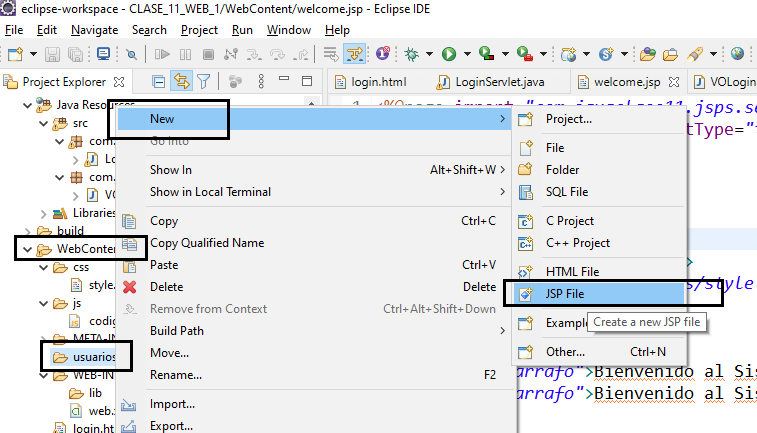
%>

</p>

</body>

</html>

1. **Luego** Creamos la carpeta **usuarios** con un nuevo **usuarios.jsp**:



Y colocamos lo siguiente:

<%@page import="com.javaclase11.jsps.servlets.vo.VOLogin"%>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Usuarios</title>

</head>

<body>

<p class="parrafo">Bienvenido al Sistema como objeto completo: <%= ((VOLogin)request.getAttribute("objetoCompleto")).getNombre() %></p>

</body>

</html>

Recordemos que con CTRL+ESPACIO al lado de VOLogin se importa automáticamente la clase.

1. Desde **welcome.jsp** redireccionamos a **usuarios.jsp** con un link. Como a esa altura el **request** se pierde tenemos que generar un objeto de tipo **HttpSession** de tal manera que nos quede:

**Welcome.jsp:**

<%@page import="com.javaclase11.jsps.servlets.vo.VOLogin" %>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

<link rel="stylesheet" href="css/style.css">

</head>

<body>

<p class="parrafo">Bienvenido al Sistema de JSPs y Servlets: <%= request.getAttribute("usuarioLogueado") %></p>

**<a href="usuarios/usuarios.jsp">Administracion de Usuarios</a>**

</body>

</html>

**usuarios.jsp**

<%@page import="com.javaclase11.jsps.servlets.vo.VOLogin"%>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<p class="parrafo">Bienvenido al Sistema como objeto completo: <%= ((VOLogin)session.getAttribute("objetoCompleto")).getNombre() %></p>

</body>

</html>

Y desde el **LoginServlet.java** creamos el objeto **httpsession**:

. . . .

vo.setPassword(password);

//Seteamos el atributo al request

request.setAttribute("usuarioLogueado", usuario);

**//Generamos una sesion para poder guardar el atributo "objetoCompleto" y luego recuperarlo desde usuario.jsp**

**HttpSession session = request.getSession(true); //con un true decimos que nuestra sesion la estamos inicializando**

**//Le cargamos el objeto vo a la sesion**

**session.setAttribute("objetoCompleto", vo); //no lo podemos guardar en un request porque al utilizar un ancla desde el jsp se perdio el request en ese momento**

. . . .

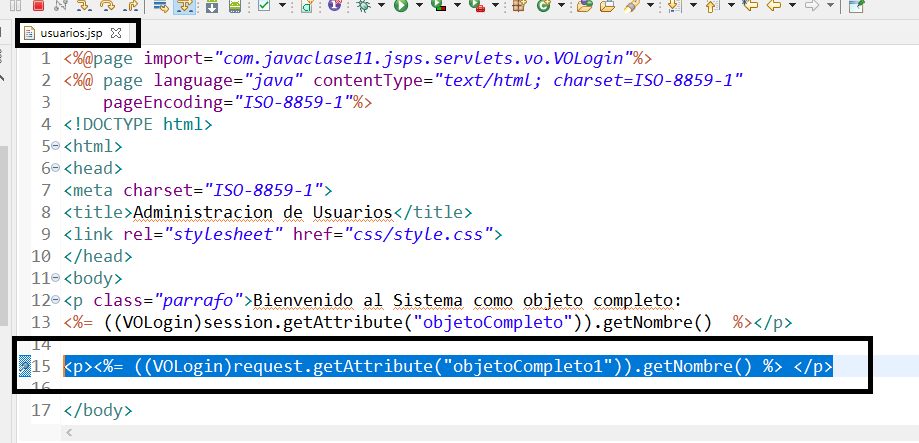
}

Entonces la diferencia entre un **request** y un **session** es que el request es por petición de una pagina a otra. Y el session se conserva a lo largo de toda mi sesión, de toda mi aplicación.

Cuando se destruye la sesión ya no podre utilizar esos atributos. El sistema enviara un **null Pointer exception**.

De hecho si le agregámos la siguiente línea a **usuarios.jsp**:

<p><%= ((VOLogin)request.getAttribute("objetoCompleto1")).getNombre() %> </p>



Veremos que obtendremos un **java.lang.nullPointerException**.

1. Cargamos una **lista** en **usuarios.jsp**

Diferencia entre **button** y **submit**. Submit envia el formulario mientras que button no. Aunque este ultimo puede ser activado por JavaScript.

1. Creamos un servlet **UsuariosServlet** el cual mapea con **/adminUsuarios.servlet**

Damos doble click en **URL mappings**.



Mas abajo sumaremos el codigo correspondiente a UsuariosServlet.java.

1. En definitiva los códigos nos deben quedar en orden de aparicion

**Login.html**: listo

**LoginServlet: completamos algún codigo**

package com.javaclase11.jsps.servlets;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Enumeration;

import java.util.Iterator;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import com.javaclase11.jsps.servlets.vo.VOLogin;

/\*\*

\* Servlet implementation class LoginServlet

\*/

//@WebServlet("/LoginWeb.jr")

public class LoginServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* Default constructor.

\*/

public LoginServlet() {

// TODO Auto-generated constructor stub

}

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

String usuario = request.getParameter("usuario");

String password = request.getParameter("password");

System.out.println("Usuario: "+usuario);

System.out.println("Password: "+password);

//validacion

if(usuario.equals("user")&& password.equals("1234")) {

System.out.println("Welcome");

VOLogin vo = new VOLogin();

vo.setNombre("Leo");

vo.setEdad(21);

vo.setUsuario(usuario);

vo.setPassword(password);

//Seteamos el atributo al request

request.setAttribute("usuarioLogueado", usuario);

//Generamos una sesion para poder guardar el atributo "objetoCompleto" y luego recuperarlo desde usuario.jsp

HttpSession session = request.getSession(true); //con un true decimos que nuestra sesion la estamos inicializando

//Metemos el request dentro de la sesion

session.setAttribute("objetoCompleto", vo); //no lo podemos guardar en un request porque al utilizar un ancla desde el jsp se perdio el request en ese momento

**//Creamos una lista que le pasaremos a usuarios.jsp**

**List<VOLogin> lista = new ArrayList<VOLogin>();**

**//y la guardamos en un atributo**

**session.setAttribute("listaUsuarios", lista);**

//redireccionamos a welcome.jsp

request.getRequestDispatcher("welcome.jsp").forward(request, response);

}else {

System.out.println("Error en credenciales de acceso...");

response.sendRedirect("login.html");//Nos envia a la pagina de inicio

}

**//Ejemplo del uso getParameterNames()**

**ArrayList<String> parameterNames = new ArrayList<String>();**

**//Me devuelve un objeto de tipo Enumeration, lo recorro y lo agrego a una collection**

**Enumeration enum1 = request.getParameterNames();**

**while(enum1.hasMoreElements()) {**

**String parameterName = (String) enum1.nextElement();**

**parameterNames.add(parameterName);**

**}**

**//Itero la collection y luego la muestro por consola**

**Iterator iter = parameterNames.iterator();**

**int cont = 1;**

**while(iter.hasNext()) {**

**System.out.println("ParName "+cont +": "+iter.next());**

**cont++;**

**}**

}

}

**welcome.jsp**

<%@page import="com.javaclase11.jsps.servlets.vo.VOLogin" %>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

<link rel="stylesheet" href="css/style.css">

</head>

<body>

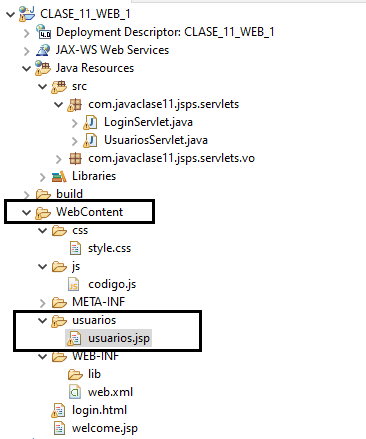
<p class="parrafo">Bienvenido al Sistema de JSPs y Servlets: <%= request.getAttribute("usuarioLogueado") %></p>

<a href="usuarios/usuarios.jsp">Administracion de Usuarios</a>

</body>

</html>

**Usuarios.jsp**

****

**usuarios.jsp**

<%@page import="java.util.List"%>

<%@page import="com.javaclase11.jsps.servlets.vo.VOLogin"%>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

**<link rel="stylesheet" href="../css/style.css">**

**<script type="text/javascript" languaje="JavaScript" src="../js/codigo.js"></script>**

</head>

<body **onload="carga();"**>

<p class="parrafo">

Bienvenido al Sistema como objeto completo:

<%= ((VOLogin)session.getAttribute("objetoCompleto")).getNombre() %></p>

**<!-- Creamos una tabla en un formulario-->**

**<!-- El action se populara via Javascript -->**

**<form action="" method="post" id="formaUsuarios">**

**<input type="hidden" name="accion" id="accion">**

**<table>**

**<tr >**

**<td**

**style="font-weight: bold; size: 12px; color: blue; text-align: right;">Usuario:**

**</td>**

**<td><input type="text" name="usuario" id="usuario"></td>**

**</tr>**

**<tr>**

**<td**

**style="font-weight: bold; size: 12px; color: blue; text-align: right;">Password:**

**</td>**

**<td><input type="password" name="password" id="password">**

**</td>**

**</tr>**

**<tr>**

**<td**

**style="font-weight: bold; size: 12px; color: blue; text-align: right;">Nombre(s):**

**</td>**

**<td><input type="text" name="nombre" id="nombre"></td>**

**</tr>**

**<tr>**

**<td**

**style="font-weight: bold; size: 12px; color: blue; text-align: right;">Edad:**

**</td>**

**<td><input type="text" name="edad" id="edad"></td>**

**</tr>**

**<tr colspan="3">**

**<!-- La invocacion a nuestro servlet lo hacemos via Javascript -->**

**<td colspan="2">**

**<input type="button" value="Guardar" class="boton" onclick="guardar();">**

**<input type="button" value="Actualizar" class="boton" onclick="actualizar();">**

**<input type="button" value="Eliminar" class="boton" onclick="borrar();">**

**</td>**

**</tr>**

**</table>**

**<table>**

**<tr>**

**<th>Usuario</th>**

**<th>Nombre(s)</th>**

**<th>Edad</th>**

**</tr>**

**<%**

**// Creamos la lista, la englobamos en un try-catch para cuando en la ida salte un javaNullPointerException**

**List<VOLogin> lista = null;**

**try{**

**lista = (List<VOLogin>)session.getAttribute("listaUsuarios");**

**}catch(Exception e){**

**System.out.println("nullPointerException");**

**}**

**if(lista == null){**

**}else{**

**// Iteramos la lista**

**for(VOLogin obj: lista){**

**%>**

**<tr>**

**<td> <%= obj.getUsuario() %></td>**

**<td> <%= obj.getNombre() %></td>**

**<td> <%= obj.getEdad() %></td>**

**</tr>**

**<%**

**}**

**}**

**%>**

**</table>**

**//valida que el nombre exista, si existe avisa al usuario y en el servlet**

**//no permite cargarlo a la lista.**

**<p> <%**

if**(request.getAttribute("mensajeExiste")==**null**){**

**System.out.println("es null");**

**}**else**{**

**out.print(request.getAttribute("mensajeExiste"));**

**}**

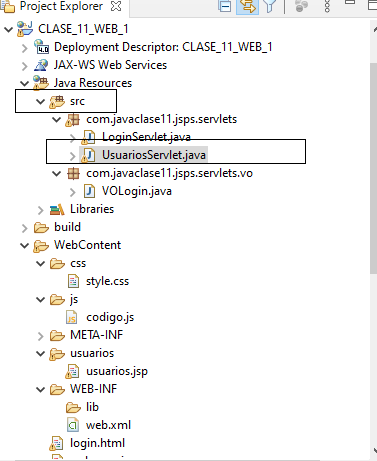
**%></p>**

**</form>**

</body>

</html>

**UsuariosServlet**



package com.javaclase11.jsps.servlets;

import java.io.IOException;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.javaclase11.jsps.servlets.vo.VOLogin;

/\*\*

\* Servlet implementation class UsuariosServlet

\*/

@WebServlet**("/usuarios/**adminUsuarios.servlet") //dejamos la annotation, equivale a url-pattern del web.xml. Debemos hacer referencia a la carpeta de usuarios porque el .jsp que esta invocando el servlet se encuentra en dicha carpeta

public class UsuariosServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public UsuariosServlet() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

response.getWriter().append("Served at: ").append(request.getContextPath());

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

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

int check=0, contr=0;

String accion = request.getParameter("accion"); //recupera el valor del atributo accion campo hidden el cual contendra "guardar", "actualizar" o "borrar".

System.out.println("Accion: "+accion);//Muestra por consola si es Guardar, Actualizar, Eliminar

//Obtenemos la sesion a traves del request y luego obtenemos la Lista de Usuarios

List<VOLogin> lista = (List<VOLogin>) request.getSession().getAttribute("listaUsuarios");

VOLogin vo = new VOLogin();

//Le pedimos los siguientes valores al request para guardarlos en vo

vo.setUsuario(request.getParameter("usuario"));

vo.setPassword(request.getParameter("password"));

vo.setNombre(request.getParameter("nombre"));

vo.setEdad(Integer.valueOf(request.getParameter("edad")));

//En base a la accion que llega

if(accion.equals("guardar")) {

//chequeamos si el nombre ya se encuentra en la lista

try {

for(VOLogin aux : lista ) {

if(aux.getNombre().equalsIgnoreCase(vo.getNombre())&& check==0) {

System.out.println("No se puede ingresar un nombre que ya existe");

request.setAttribute("mensajeExiste", "No es posible ingresar un nombre que ya existe");

contr = 2;

}

}

if(contr==0) lista.add(vo);

}catch(Exception e) {

System.out.println("Por si se da la excepcion concurrent modification exception");

}

if(lista.size()==0) {

lista.add(vo);

}

} else if(accion.equals("actualizar")) {

for(VOLogin aux: lista) {

//si accion tiene actualizar permite actualizar la password, edad y nombre

if(aux.getNombre().equals(vo.getNombre())) {

aux.setPassword(vo.getPassword());

aux.setEdad(vo.getEdad());

aux.setNombre(vo.getNombre());

}

}

}else if(accion.equals("borrar")) {

//si accion tiene borrar permite borrar el registro

for(VOLogin aux: lista) {

if(aux.getNombre().equals(vo.getNombre())) {

lista.remove(aux);

}

}

}

//redireccionamos al mismo usuarios.jsp

request.getRequestDispatcher("usuarios.jsp").forward(request, response); }

}

En este ultimo servlet usamos la **anotation** en lugar de **web.xml.**

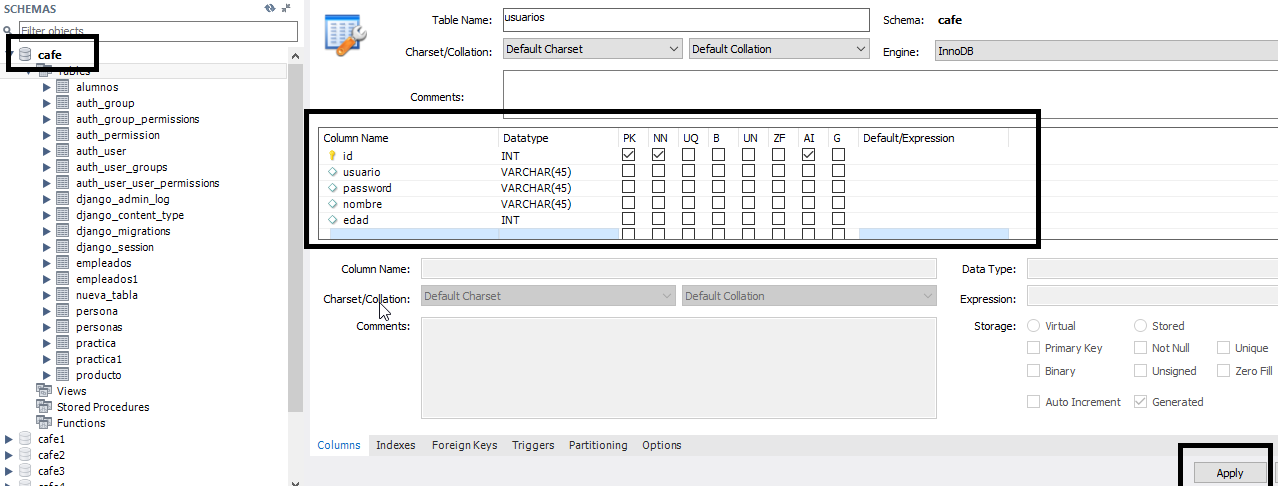
Y finalmente le responde a **usuarios.jsp.**

1. Codigo definitivo: Correr en Eclipse

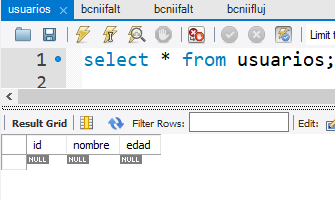


gh. Le agregamos BBDD:

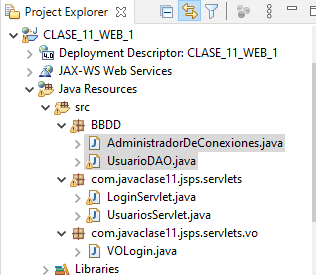
1. Creamos la tabla en Mysql Workbench:



Una vez que esta creada la tabla le hacemos un **select** para ver los campos:



2.Agregamos en un nuevo paquete **AdministracionDeConexiones** y **UsuarioDAO.**



**Mysql.jar** levantamos el conector al proyecto:

Boton derecho sobre el proyecto=> Properties=> Java Build Path=> Y levantamos el jar habiendolo descargado previamente



**UsuarioDAO.java**

package BBDD;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import Excepciones.CafeStoreException;

import servlets.vo.VOLogin;

public class UsuarioDAO {

public static void inserta(VOLogin vo) throws CafeStoreException {

Connection con = null;

PreparedStatement st = null;

try {

con = AdministradorDeConexiones.obtenerConexion();

String sql = "insert into cafe.usuarios(nombre, edad) values(?,?)";

st = con.prepareStatement(sql);

st.setString(1, vo.getNombre());

st.setInt(2, vo.getEdad());

st.execute();

}catch(Exception e) {

e.printStackTrace();

}finally {

try {

con.close();

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

public static void modifica(VOLogin vo) throws CafeStoreException {

PreparedStatement st = null;

Connection con = null;

//Chequeamos si el producto existe

if (getUsuario(vo)==null) {

throw new CafeStoreException("El usuario no existe");

}

try {

con = AdministradorDeConexiones.obtenerConexion();

String sql = "update usuarios set nombre=?, edad=? where nombre = ?";

st = con.prepareStatement(sql);

st.setString(1, vo.getNombre());

st.setInt(2, vo.getEdad());

st.setString(3, vo.getNombre());

st.execute();

} catch (Exception e) {

throw new CafeStoreException("Se dio un problema en la actualizacion. Escalar al equipo se soporte.");

} finally {

try {

st.close();

con.close();

} catch (SQLException e) {

e.printStackTrace();

throw new CafeStoreException("Se dio un problema en el cierre de conexion con la BBDD");

}

}

}

public static void elimina(VOLogin vo) throws CafeStoreException {

PreparedStatement st = null;

Connection con = null;

if (getUsuario(vo)==null) {

throw new CafeStoreException("El usuario no existe");

}

try {

con = AdministradorDeConexiones.obtenerConexion();

String sql = "delete from usuarios where nombre = ?";

st = con.prepareStatement(sql);

st.setString(1, vo.getNombre());

st.execute();

} catch (Exception e) {

throw new CafeStoreException("Hubo un problema en la eliminacion");

} finally {

try {

st.close();

con.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

public static VOLogin getUsuario(VOLogin vo) throws CafeStoreException {

Connection con = null;

ResultSet rs = null;

PreparedStatement st = null;

VOLogin vo1 = null;

try {

con = AdministradorDeConexiones.obtenerConexion();

String sql = "select \* from usuarios where nombre = ?";

st = con.prepareStatement(sql);

st.setString(1, vo.getNombre());

rs = st.executeQuery();

if (rs.next()) {

String nombre = rs.getString("nombre");

int edad = rs.getInt("edad");

vo1 = new VOLogin();

vo1.setNombre(nombre);

vo1.setEdad(edad);

} else {

throw new CafeStoreException("El usuario no existe");

}

} catch (SQLException e) {

e.printStackTrace();

} catch (Exception e) {

System.out.println(e.getMessage());

} finally {

try {

rs.close();

st.close();

con.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

return vo1;

}

public static List<VOLogin> getUsuarios() {

Connection con = null;

List<VOLogin> usuarios = new ArrayList<VOLogin>();

ResultSet rs = null;

PreparedStatement st = null;

try {

VOLogin vo = null;

con = AdministradorDeConexiones.obtenerConexion();

String sql = "select \* from usuarios"; //me traigo toda la tabla

st = con.prepareStatement(sql);

rs = st.executeQuery();

while (rs.next()) {

String user = rs.getString("usuario");

String pass = rs.getString("password");

String nombre = rs.getString("nombre");

int edad = rs.getInt("edad");

vo = new VOLogin();

vo.setUsuario(user);

vo.setPassword(pass);

vo.setNombre(nombre);

usuarios.add(vo);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

rs.close();

st.close();

con.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

return usuarios;

}

}

1. **Creamos package Excepciones: CafeStoreException:**

**package** Excepciones;

**public** **class** CafeStoreException **extends** Exception{

**private** String mensaje;

**public** CafeStoreException(String mens) {

**this**.mensaje = mens;

}

@Override

**public** String getMessage() {

**return** mensaje;

}

}

1. Conexión con BBDD: Agregar el mysql-connector-java-8.0.19.jar en la carpeta **WebContent=>WEB-INF=>lib**



**Codigo Eclipse:**

****