**Ejercicio 2: Servlets: creación y despliegue de servlets en apache-tomcat.**

**utilizando el IDE Eclipse.**

1. **Instalación de Eclipse.**

Eclipse es un entorno de desarrollo opensource basado en Java, para su funcionamiento necesita tener instaladas la máquina virtual de java (JRE) y para desarrollar JSPs y servlets se necesita la versión JDK de Java. Estas instalaciones ya las realizamos al instalar el contenedor web: apache-tomcat. Desde la página: <http://www.eclipse.org/downloads> descarga la versión **Eclipse IDE for Java EE Developers** que incluye la integración con los servidores de aplicaciones más utilizados como **tomcat, JBOSS, WebSphere,….** Una vez descargado el archivo, descomprímelo y crea un acceso directo en el escritorio.

1. **Configuración básica de Eclipse:**

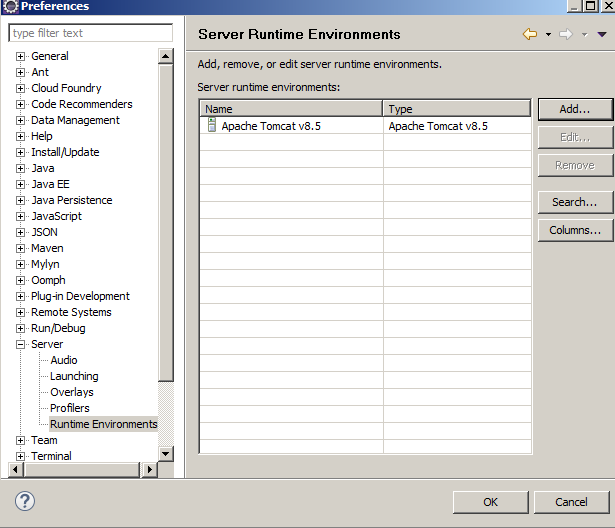
Al arrancar Eclipse debemos elegir un directorio de trabajo, (**workspace),** donde se guardarán los proyectos que vamos realizando y otras opciones de configuración de Eclipse, dejamos el que viene por defecto.

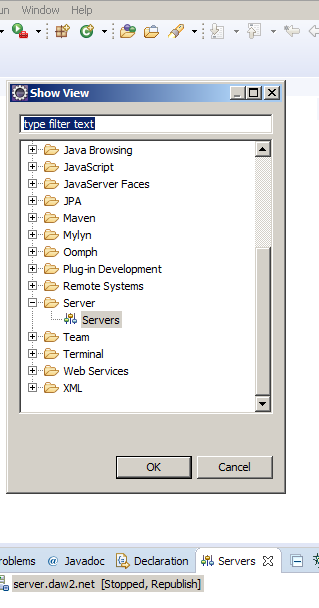
El entorno de desarrollo está dividido en diferentes partes, según la perspectiva (vista) que elijamos. Por defecto está activada la perspectiva Java. Se puede cambiar en la opción de menú: **Windows/Open perspective** por ejemplo elegimos **Resource.**

1. **Integración de Eclipse con tomcat:**

En el menú: **Window** -> **Preferences** y en el diálogo que aparece debemos seleccionar **Server** -> **Runtimes Environments** y presionar el botón "**Add**...":se selecciona el directorio de instalación de tomcat y JRE

1. Creamos un nuevo servidor, desde el botón Servers (área derecha de la pantalla), pulsamos el enlace para crearlo, se elige el servidor que tenemos instalado: **tomcat v8.0,** el hostname: localhost, nosotros especificaremos el nombre del equipo: **server.daw2.net**. Ahora podemos ver el servidor Tomcat en la vista de servidor y también un nuevo proyecto llamado Servers donde también podemos ver al servidor tomcat. **Captura la pantalla.**





También se puede hacer desde la ventana "**Servers**", (**Window -> Show View -> Servers**) y seguidamente seleccionar del menú contextual la opción **New -> Server**,

1. **Arranque y parada** de tomcat:

Desde Eclipse se puede arrancar y para el servidor tomcat, seleccionándolo y utilizando las opciones del menú contextual. Iniciarlo. Si se visualiza un error de que los puertos utilizados por tomcat están en uso por otro proceso, es porque tenemos iniciado tomcat en la máquina virtual o se ha creado anteriormente otro proceso servidor, (se puede crear una nueva instancia del servidor de forma manual, es decir, otro proceso tomcat que deberá ejecutarse en otro puerto no ocupado, por ejemplo: 8081,… ya que el 8080 estará ocupado).

Detenemos este proceso o cambiamos el puerto en el que escucha tomcat, por otro, por ejemplo el 8081, el puerto se cambia en el archivo de configuración del servidor: **server.xml.** En Eclipse en el árbol de proyectos, desplegando **Servers.**  Se visualizan los archivos de configuración de tomcat. .

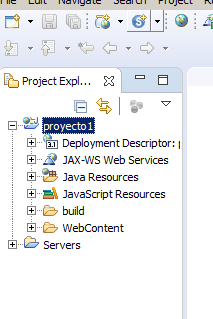
1. **Prueba de integración de Eclipse con tomcat:**

Realizamos un primer proyecto, **File, New, Dynamic Web Project**, seguimos el asistente. Introduce el nombre del proyecto, en nuestro caso: “**proyecto1”** las siguientes ventanas nos permitirán establecer:

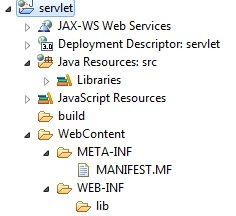
* el directorio donde estarán los fuentes (**src**) podemos dejar el de defecto (**workspace\**)
* el servidor a utilizar, en nuestro caso: **apache-tomcat v 8.0** ya ha sido seleccionado como nuestro target runtime. Si tuviéramos otro servidor instalado como JBoss y fuéramos a trabajar con él, cambiaríamos el target a JBoss.
* el directorio donde se guardarán los archivos precomiplados: **build\classes**
* el directorio por el que empezarán todas las URLs de los recursos de nuestro proyecto (**Context root)**, por defecto Eclipse utiliza el **nombre del proyecto**
* el directorio de contenidos: **WebContent** donde se guardarán los archivos **html** y **jsp**, también el descriptor de despliegue: **web.xml,**  (si activamos la casilla correspondiente).
* Activamos la casilla para que se genere el descriptor de despliegue: **web.xml**

**Eclipse**  nos habrá creado el árbol de directorios del proyecto, se visualiza en la ventana **Project Explorer.**

**Captura la pantalla.**

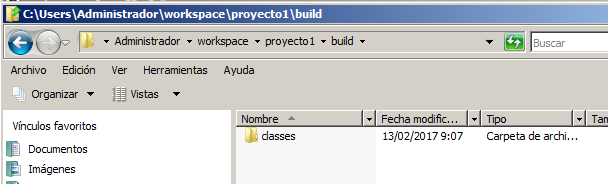


En nuestro caso el nombre del proyecto es: **proyecto1**

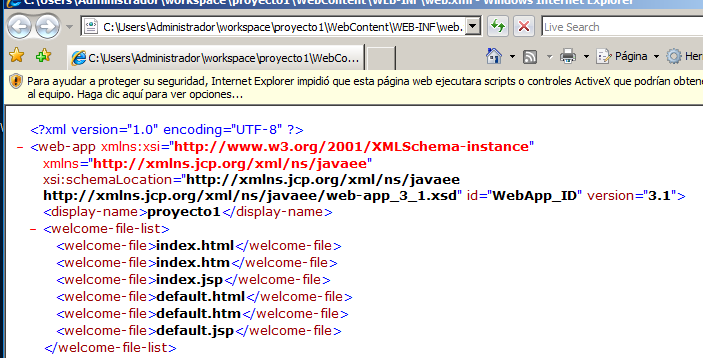
****

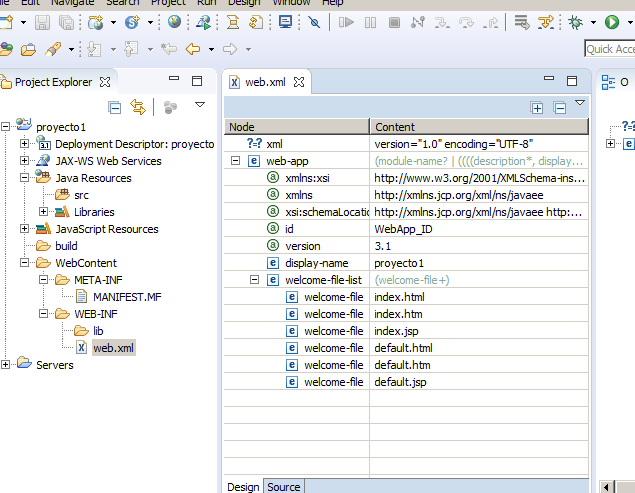
* En la carpeta **Java Resources src** se guardará el código fuente de las clases Java empaquetadas en packages. Dentro de esta carpeta, en la carpeta **Libraries**, podemos encontrar las librerías de Apache Tomcat 8, necesarias para desarrollar servlets, como la **servlet-api.jar**. (Despliega esta carpeta). Eclipse se encargará automáticamente de publicar los **\*.class** en donde corresponda.
* En la carpeta **WebContent**guardaremos los archivos Web (HTML, JavaScript, CSS, JSP, imágenes, documentos, etc), dentro de esta carpeta podemos ver la estructura estándar de J2EE para un proyecto Web con la carpeta WEB-INF, el archivo web.xml, etc.
* En la carpeta **WebContent\WEB-INF** guardaremos el descriptor de despliegue:**web.xml.**
* En la carpeta **WebContent\WEB-INF\lib**guardaremos las librerías externas que utilicemos en la aplicación Web Java.

1. Entra, con el explorador de archivos, en la carpeta donde eclipse guarda todo el espacio de trabajo, por ejemplo: Users\Administrador\**workspace**. Podemos ver lo que crea el IDE por debajo: una nueva carpeta por cada proyecto y dentro, las carpetas **src** y **WebContent** comentadas anteriormente, además de una carpeta extra **/build/classes**, donde almacena las clases Java compiladas empaquetadas en sus respectivos paquetes. **Captura la pantalla.**



1. Abre el archivo descriptor de despliegue:**web-xml** vemos que contiene únicamente las etiquetas: **<display-name>**  que visualizará el nombre del proyecto en la barra de menú y la etiqueta: **<welcom-file-list>**  que especifica la lista de páginas de index que entregará el servidor. No es necesario editar este archivo ya que el IDE se encarga de configurar el despliegue. **Captura la pantalla.**





1. Vamos a incluir una página de index: **index.html**  en nuestro proyecto**.** Sobre la carpeta **proyecto1**, botón dcho. **New, file HTML**. Utilizamos la plantilla que Eclipse nos muestra por defecto. Escribe un título y edita el cuerpo de la página. Guárdala. Observa que se ha creado dentro de la carpeta: **WebContent.\*\*\*\*\***
2. Para ejecutarla, sobre la página: **index.html** botón dcho: **Run as, Run on server** se elige el servidor: selecciona tomcat v8.0. El proyecto, en este caso, la página: **index.html** se abre en un **navegador interno** de Eclipse, fíjate en la URL: nombre del equipo, el puerto que utiliza tomcat, el nombre por el que empiezan todos los recursos del proyecto (**Context root)**. Se puede elegir otro navegador: abre el menú:**Window, Web Browser** para verlo. [http://localhost:8080/proyecto1/index.html](http://localhost:8080/proyecto1/inicio.html)
3. Vamos a incluir en nuestro proyecto los servlets creados en el Ejercicio nº1. Incluimos el servlet: **Saludo.java**. Sobre la carpeta: **proyecto1, new servlet** Implementaremos el método **doGet().**

**import java.io.IOException;**

**import java.io.PrintWriter;**

**import javax.servlet.ServletException;**

**import javax.servlet.annotation.WebServlet;**

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

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

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

/\*\*

\* Servlet implementation class Saludo

\*/

@WebServlet("/Saludo")

public class Saludo extends HttpServlet {

private static final long serialVersionUID = 1L;

public Saludo() {

super();

}

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

//Establece el tipo MIME del mensaje de respuesta en este caso HTML

**response.setContentType("text/html");**

//Crea un flujo de salida para escribir la respuesta a la request del cliente. //Pertenece a java.io

**PrintWriter salida = response.getWriter();**

//Escribe el mensaje de respuesta en una página HTML

**salida.println("<html>");**

**salida.println("<body>");**

**salida.println("<h1>Primer servlet. Genera salida en formato HTML </h1>");**

**salida.println("</body>");**

**salida.println("</html>");**

}

}

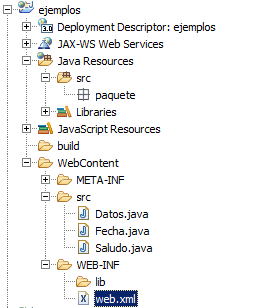
1. Ejecuta el servlet desde Eclipse. **Captura la pantalla.**



1. **Exportar la aplicación en archivo .war**: empaqueta el proyecto en un archivo **.war** que se desplegará automáticamente en cualquier contenedor java web. Botón derecho sobre “proyecto1”. **Export**, **WAR file**, elige la carpeta de destino.
2. **Ejecución de la aplicación fuera del entorno de Eclipse:** (modo producción) tenemos que **desplegar** la aplicación en el directorio: **%CATALINA\_HOME%\webapps.** Copia el **.war** de la aplicación en **webapps**. Se despliega automáticamente, visualiza el directorio desplegado: “proyecto1”. **Captura la pantalla.**



1. **Importar un proyecto:** Podemos traer a Eclipse una aplicación ya empaquetada en su archivo **.war**, con la opción **Import.** Importa la aplicación **ejemplos.war** creada en el Ejercicio\_1. Pruébala en Eclipse. **Captura la pantalla.**



1. **Creamos un nuevo proyecto:** **File, New, Dynamic Web Project**, lo llamaremos: “**proyecto2”.** En este proyecto vamos a incluir otro servlet que en este caso  **se ejecutará desde un formulario**.

En el formulario se pedirá nombre de usuario y password. El servlet comprobará si el nombre=”tomcat” y el password=”tomcat” y responderá con un mensaje: Login ok, en otro caso: Login inválido.

Los parámetros se capturan mediante el método **getParameter()** del objeto: **request**:

**String request.getParameter(String name)** el parámetro **name** es el componente del formulario que interesa capturar. (Si no se rellena algún cuadro, devuelve “” y si no existe el name que se le pasa, devuelve null).

Sobre la carpeta **proyecto2**: crea el formulario, utiliza el **index.html**  creado anteriormente. El formulario llamará al servlet “**Login”**  que crearemos ahora, con el método **GET.**

<!DOCTYPE html>

<html>

<head>

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

<title>Login de acceso</title>

</head>

<body>

<form action="**Login**" method="**get**">

<table>

<tr>

<td>Usuario</td>

<td><input **name="usuario"** /></td>

</tr>

<tr>

<td>password</td>

<td><input **name="password"** /></td>

</tr>

</table>

<input type="submit" />

</form>

</body>

</html>

1. **Crea el servlet**, botón dcho sobre la carpeta de la aplicación: **New, servlet**  le llamamos **Login** (**Class name)**, no crearemos ningún paquete, simplemente se creará en el paquete por defecto, seguimos el asistente, la **url-mapping** que por defecto configura será **/Login**, dejamos activados los métodos que vienen por defecto y que se insertarán como código en el servlet: constructores, doGet(), doPost(),…).Finalizar. Añadir el código al servlet:

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class Login

\*/

@WebServlet(description = "servlet Login básico", **urlPatterns** = { "**/Login**" })

**public class Login extends HttpServlet {**

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

**public Login() {**

**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

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

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

if (user.equals("tomcat") && pass.equals("tomcat")) {

respuesta(response, "Login ok");

} else {

respuesta(response, "invalid Login");

}

}

private void respuesta(HttpServletResponse response, String msg)

throws IOException {

**PrintWriter out = response.getWriter();**

out.println("<html>");

out.println("<body>");

out.println("<t1>" + msg + "</t1>");

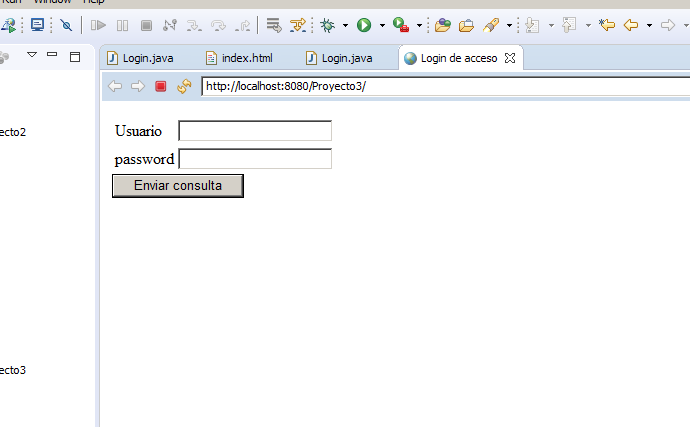
out.println("</body>");

out.println("</html>");

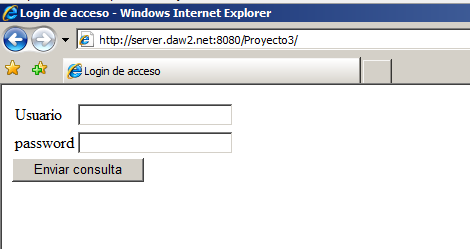
}

}

1. Probar el servlet (Ejecutar el proyecto: Run as).**Captura la pantalla.**
2. **Crear el .war.** Botón derecho sobre el proyecto. **Export**, WAR file, elige la carpeta de destino.



1. **Prueba** la aplicación ”proyecto2” desde **fuera de Eclipse:** copia el **.war** de la aplicación en **webapps**. Abre el navegador: **http://server.daw2.net:8080/proyecto2** (si es sobre el equipo real sustituir el nombre de: server.daw2.net por el de **localhost** ). **Captura la pantalla.**

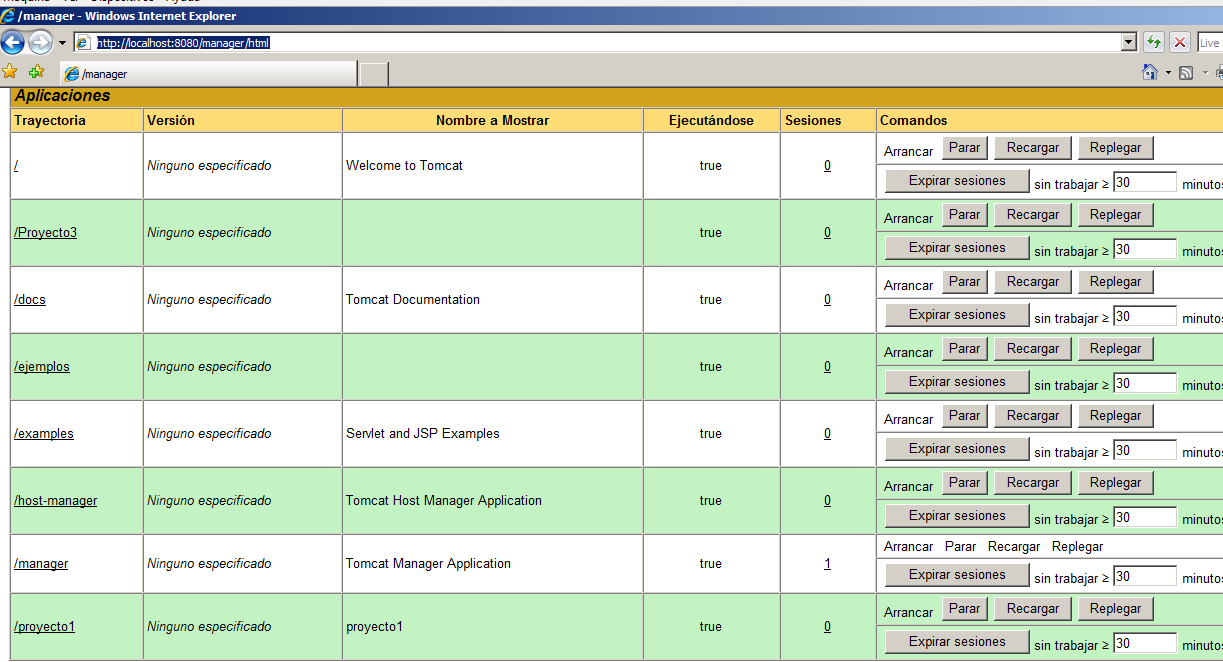


1. **Consola de administración**: tomcat dispone de la aplicación: **tomcat Web Manager** que permite administrar las aplicaciones que se están ejecutando o se van a ejecutar en tomcat, ábrela: <http://localhost:8080/manager>.

Nos pide nombre y password del usuario (tendríamos que haberlo configurado en la instalación), como no lo hicimos tenemos que editar el archivo **tomca-users.xml,** que está en el directorio:  **config** de tomcaty añadir el usuario administrador de la consola, con las siguientes líneas:

**<role rolename="manager-gui"/>**

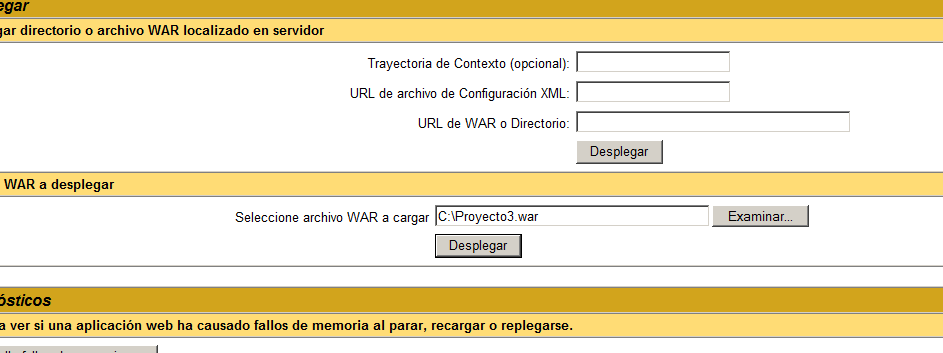
**<user username="tomcat" password="tomcat" roles="manager-gui"/>** (el usuario y su contraseña son los que queramos). Reinicia tomcat. Entra ahora en la consola de administración. **Captura la pantalla.**

****

Desde la consola manager, podemos:

* Listar las aplicaciones desplegadas
* Iniciar/parar las aplicaciones
* Desplegar/borrar las aplicaciones
* Consultar el estado del servidor
* Listar las sesiones activas y consultar sus estadísticas

1. **Despliegue desde la consola:** Comprueba que está desplegada. Antes de volver a desplegarla habrá que realizar un repliegue de la misma (borrarla)” **undeploy “**. Vuelve a desplegarla: desde la opción de despliegue, selecciona el archivo .war, (lo sube a webapps y lo descomprime automáticamente). Abre la página en un navegador. **Captura la pantalla.**



1. Modifica la aplicación desde Eclipse para utilizar el método **doPost**. Habrá que modificar en el **formulario**: <form action="**Login**" method="**post**">

y en el **servlet** añadir el método **doPost**(), que tendrá una llamada a **doGet(),** (podría haberse implementado únicamente el método doPost() con un código idéntico al de doGet():

protected void **doPost**(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

**doGet(request,response);**

}

1. Prueba la aplicación desde Eclipse. Observa que al abrirse el servlet ya no son visibles en la URL los valores de nombre y password. **Captura la pantalla.**

