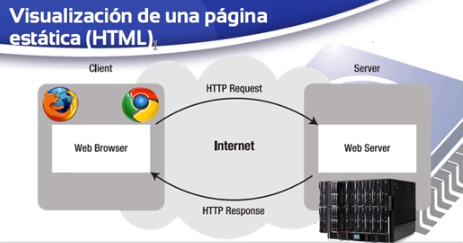
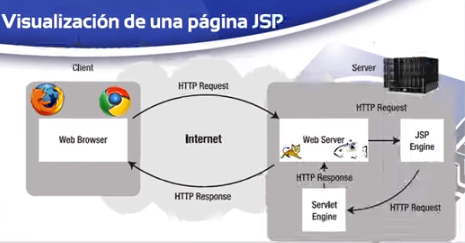
2.4.1 Desarrollo Web con JSP y Servlet - Diferencia en la ejecución de una página HTML y JSP





Los siguientes pasos explican cómo el servidor web crea la página web usando JSP:

* Al igual que con una página normal, su navegador envía una solicitud HTTP al servidor web.
* El servidor web reconoce que la solicitud HTTP es para una página JSP y **la reenvía a un motor JSP**. Esto se hace usando la URL o la página JSP que termina con **.jsp en** lugar de **.html** .
* El **motor JSP** carga la página JSP y **la convierte en un contenido de servlet**. Esta conversión es muy simple en la que todo el texto de la plantilla se convierte en declaraciones println () y todos los elementos JSP se convierten a código Java. Este código implementa el correspondiente comportamiento dinámico de la página.
* El **motor JSP** **compila el servlet en una clase ejecutable** y reenvía la solicitud original a un **motor de servlet**.
* Una parte del servidor web denominada motor de servlet carga la clase Servlet y la ejecuta. Durante la ejecución, el servlet **produce una salida en formato HTML**. La salida se transmite posteriormente al servidor web por el motor de servlets dentro de una respuesta HTTP.
* El servidor web reenvía la respuesta HTTP a su navegador en términos de contenido HTML estático.
* Finalmente, el navegador web maneja la página HTML generada dinámicamente dentro de la respuesta HTTP exactamente como si fuera una página estática.

**Diferencia entre JSP y Servlet**

**Anteriormente** a las páginas **JSP**, se utilizaron tecnologías en Java tales como **CGI** o **servlets**, pero la tecnología servlets no implementa código HTML, sino que la solicitud por parte de un cliente se realiza a código lógico de una clase Java.

Un **Servlet** podría verse como una *herramienta* que **nos permite generar HTML desde código Java.**

En una web real, especialmente si queremos que tenga una apariencia razonablemente buena, tendremos que generar cantidades de HTML mayores. Si hacemos esto desde un Servlet, será increíblemente tedioso.

Los **Servlets** son una **herramienta pésima** para crear la capa de **presentación** de la aplicación web

Las páginas **JSP** son **código HTML que tiene código Java empotrado**, pero su mayor característica es que son principalmente código HTML, por lo que simplemente tendríamos que empotrar pequeñas cantidades de código Java para crear sus partes dinámicas

Otras tecnologías que compiten con Java Server Pages son ASP y PHP.

## ¿QUE ES JSP?

Una página JSP permite embeber o incluir código Java utilizando etiquetas <% %>, todo lo que vaya dentro de estas etiquetas la JVM de Java lo reconoce y lo ejecuta como código Java.

Las páginas JSP contienen sintaxis ***HTML***, para que una página pueda incluir el código Java tiene que tener la extensión **.jsp**y claro está las etiquetas <% %>.

Ejemplo **hora.jsp**

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

<html>

<body>

<h1>La hora es: <%= new Date() %></h1>

</body>

</html>

En contenedor de servlets como [Tomcat](http://tomcat.apache.org/) traduce este archivo JSP y lo convierte en un Servlet.

Un servlet es una clase Java que implementa la interfaz Servlet ( heredando de HttpServlet por ejemplo ) y que generalmente escriben en un flujo de datos de una petición HTTP ( como la que un browser hace a un servidor web )

## ¿QUE ES UN SERVLET?

Los Servlets son tecnología que se ejecuta del lado del servidor y procesan todas las **peticiones que hace un cliente**,

Ejemplo **Hora.java**

package hora;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.util.Date;

public class Hora extends HttpServlet {

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

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<body>");

out.printf("<h1>La hora es %s </h1>", new Date());

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

}

}

Estos dos ejemplos escriben aproximadamente lo mismo, se puede ver la diferencia entre uno y otro. Mientras que el servlet es una clase Java "normal" el JSP es una plantilla de propósito específico ( hecho para escribir HTML )

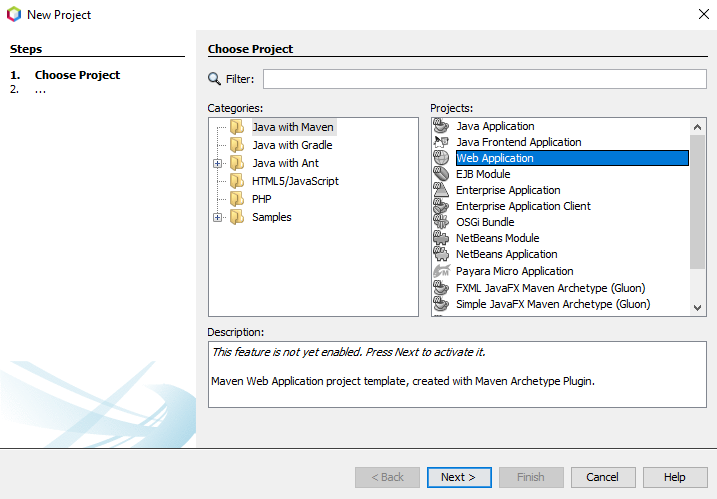
PROGRAMACIÓN WEB CON SERVLETS

1. Creación de una aplicación Web

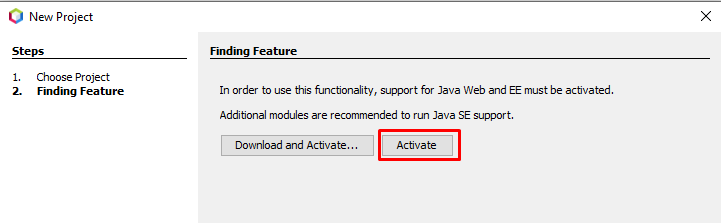
El ejemplo que vamos a realizar consiste en un formulario html y un servlet que captura los datos y crea una página de respuesta. Realizaremos la aplicación mediante NetBeans y haremos el despliegue en el servidor **GlassFish**.

1.1 Creación del proyecto

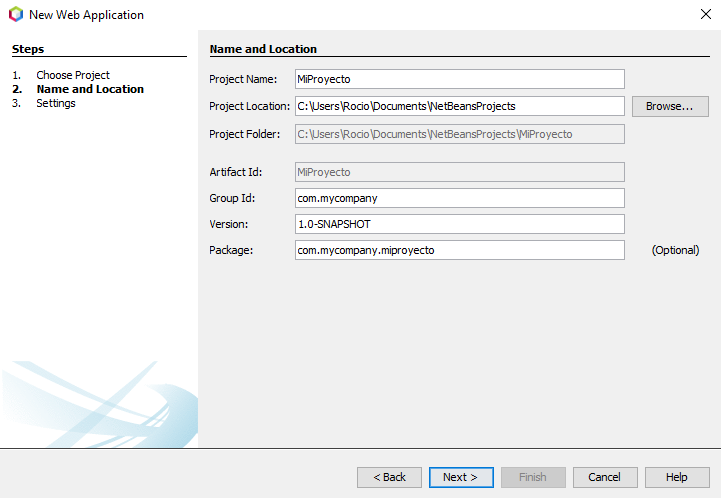
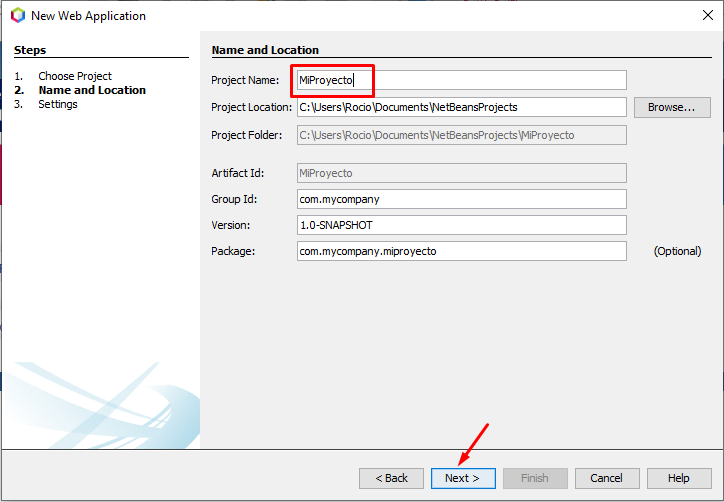
En **NetBeans** nos vamos a New Project y seleccionamos proyecto Web:



Como veis(esta sin activar) la primera vez me va a salir que debo activar Java Web



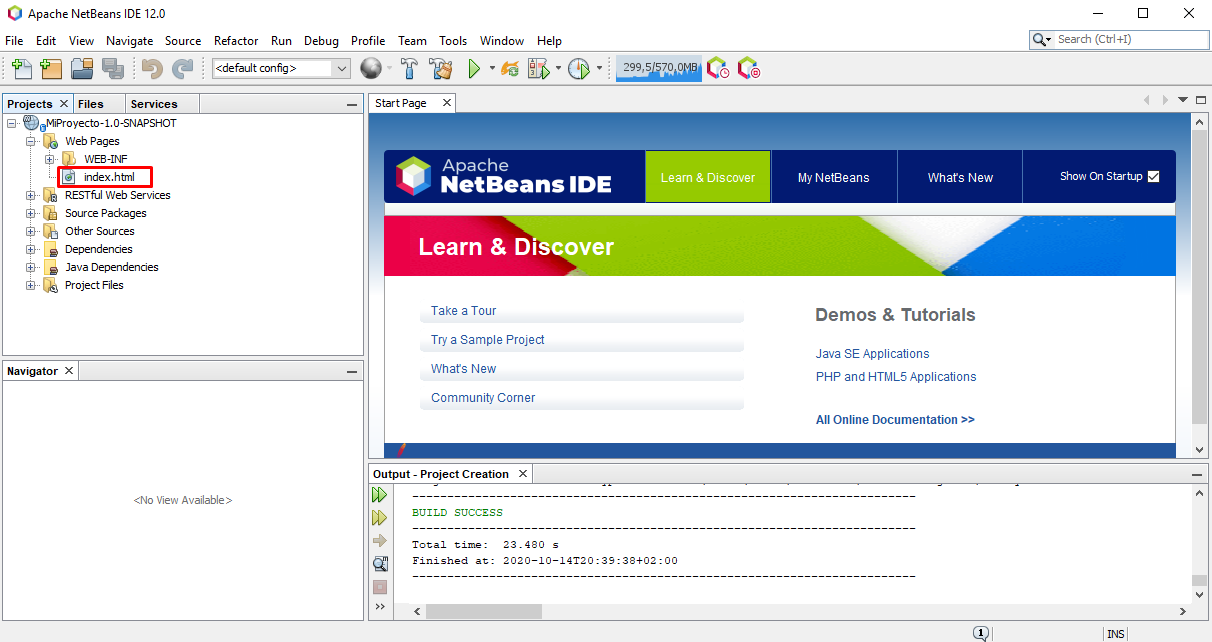
Una vez activado ya nos deja crear el proyecto y darle un nombre en mi caso MiProyecto



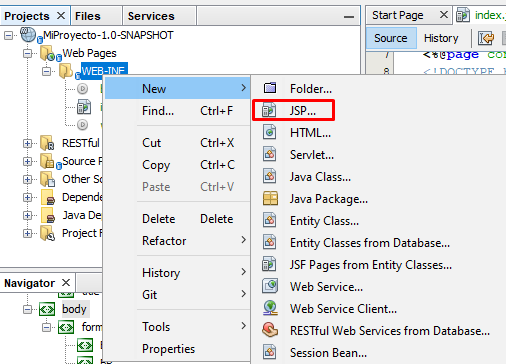
Si no tenemos ningún servidor activado no pasa nada, vamos a trabajar con el que hemos instalado en el capítulo anterior Tomcat 9..

En este caso vamos a trabajar con GlassFish Server.

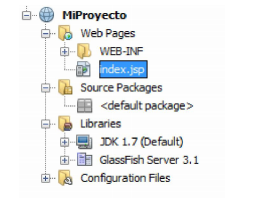
Finalmente nos aparecerá el proyecto y toda su estructura.



Al principio te creara un index.hmtl, lo **borraremos** y crearemos un nuevo jsp al que llamaremos **index.jsp**



El resultado de la creación del proyecto es un directorio con la estructura estándar de los proyectos web:



1.2 Desarrollo del formulario

Como podemos apreciar al crear el proyecto se genera la página “**index.jsp**” que reutilizaremos para codificar nuestro formulario.

* Código final:

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<title>Mi Primer Servlet</title>

</head>

<body>

<form action="**MiPrimerServlet**" method="POST">

<br><br>Introduzca un texto en el cuadro y pulse "Enviar Consulta"

<br><br><input type="text" name="TEXTO">

<br><br><input type="submit"><input type="reset">

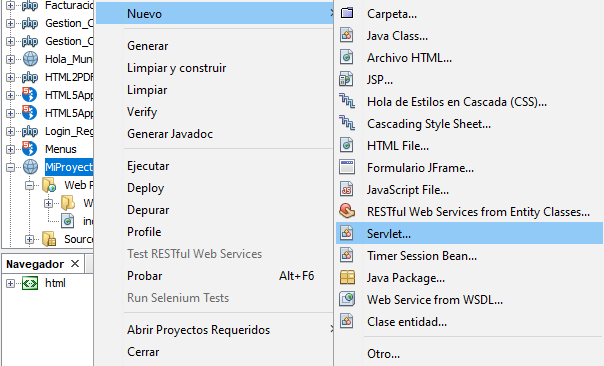
</form>

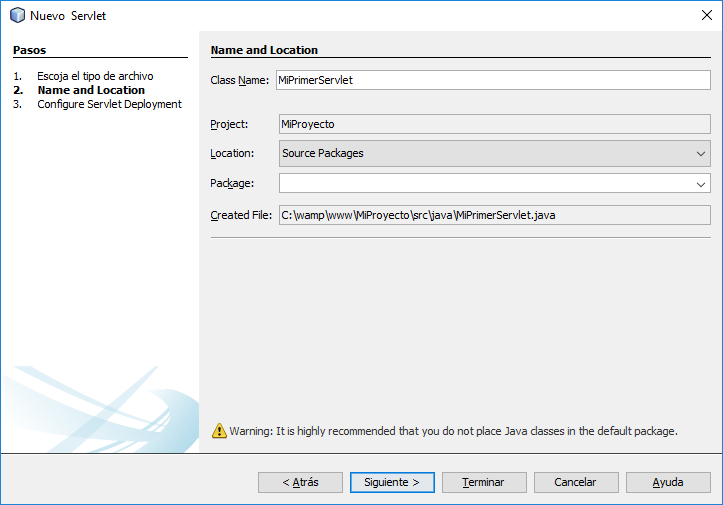
</body>

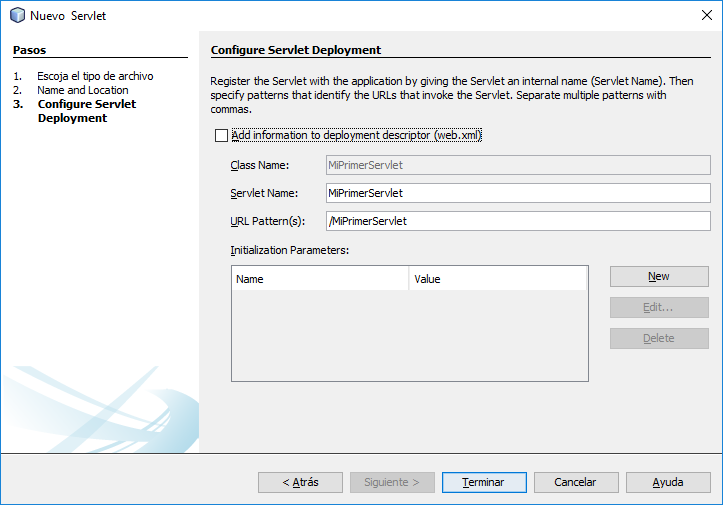
</html>

1.3 Desarrollo del servlet

Una vez realizado el formulario, crearemos un servlet para tratar la información.







Como podemos apreciar en la última pantalla ya no es necesario incluir la información del servlet en el fichero web.xml como se hacía antiguamente, esto se ha sustituido por una anotación en el propio código como veremos a continuación.

* Código final:

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;

@WebServlet(name = "MiPrimerServlet", urlPatterns = {"/MiPrimerServlet"})

public class MiPrimerServlet extends HttpServlet {

/\*\*

\* Processes requests for both HTTP <code>GET</code>

\* and <code>POST</code> methods.

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

protected void processRequest(HttpServletRequest request, HttpServletResponse

response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String txt = request.getParameter("TEXTO");

try {

out.println("<html>");

out.println("<head>");

out.println("<title>Servlet MiPrimerServlet</title>");

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

out.println("<body>");

out.println("<p>Usted ha escrito : " + txt + "</p>");

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

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

} finally {

out.close();

}

}

/\*\*

\* Handles the HTTP <code>GET</code> method.

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

/\*\*

\* Handles the HTTP <code>POST</code> method.

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

/\*\*

\* Returns a short description of the servlet.

\* @return a String containing servlet description

\*/

@Override

public String getServletInfo() {

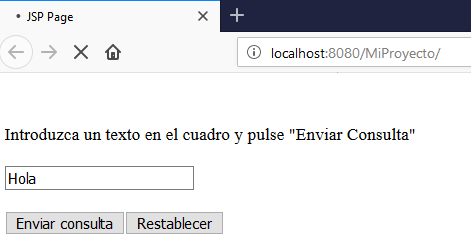
return "Short description";

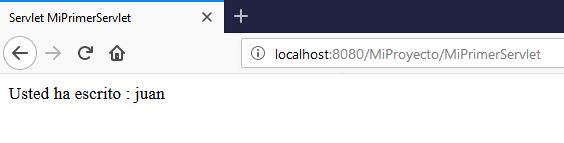
}

}

1.4 Desplegando la aplicación y ejecutándola

Para desplegar y ejecutar la aplicación en el servidor tan solo tenemos que pulsar sobre el botón derecho en **MiProyecto** y seleccionar “**Deploy**” (**desplegar**) y después ejecutar la aplicación o directamente ejecutarla ya que previamente se realizará el despliegue.





### [Empaquetado de Aplicaciones JEE - Archivos WAR](http://jmaw.blogspot.com/2012/09/empaquetado-de-aplicaciones-jee_28.html)

Los archivos **WAR**, son un tipo especial de JAR utilizado para **distribuir** los artefactos o contenido de las **aplicaciones Web en tecnología JEE**: páginas Web HTML o JSP, clases Java, servlets Java, archivos XML, librerías de etiquetas (tag libraries) y otros recursos.

Al igual que los JAR pueden estar firmados digitalmente por temas de seguridad.

Los archivos WAR incluyen carpetas y archivos especiales. Se incluye la carpeta META-INF con el manifiesto como en los JAR, pero además está la carpeta WEB-INF específica

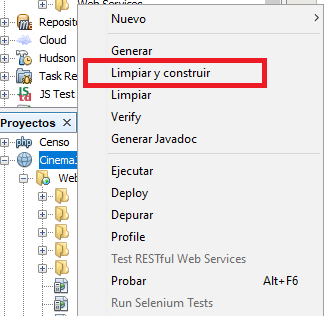
de las aplicaciones Web.

**WEB-INF**  
Hasta este punto el servidor Tomcat no difiere mucho de cualquier otro. Es decir, un directorio que hace de raíz de la aplicación donde se guardan todas las páginas o archivos accesibles a través de url. La principal diferencia es que en Tomcat existe una carpeta especial y cuyo contenido no es referenciable mediante url. Dicha carpeta se denomina WEB-INF y dentro de ella podemos encontrar:

* Directorio classes: que contiene todos los ficheros .class que utiliza nuestra aplicación. Por ejemplo, los servlets.
* Directorio lib: que contiene las librerías (.jar) necesarias. Por ejemplo, la de mysql o la de apache Tiles (para el sistema de plantillas).
* Directorio src: que contiene los ficheros fuente (.java) de nuestra aplicación. Serán compilados para generar los .class.
* Fichero web.xml. Muy importante, ya que todos los parámetros de configuración específicos para nuestra aplicación se encuentran en él.

En este directorio se coloca el archivo web.xml, donde se establece la configuración de la aplicación web.

Para generar un **.war** utilizando el IDE de Netbeans , una vez tengamos la aplicación web terminada, pulsamos con el botón derecho del ratón sobre la aplicación, seleccionaremos la opción Limpiar y construir



Dentro de la carpeta de tu proyecto verás una nueva carpeta llamada "target".Dentro la carpeta verás tu archivo WAR creado.

Práctica Obligatoria : Despliegue de aplicaciones con Apache Tomcat

Deberéis desplegar la aplicación  que os he pasado, la he sacado de :

https://github.com/wzalazar/cine

para comprobar el correcto funcionamiento del despliegue de dicha aplicación, yo podré acceder desde mi puesto a vuestro servidor de aplicaciones y poder utilizar dicha aplicación, antes de que yo la ejecute, probar que vuestro compañero de al lado pueda acceder y poder ejecutar correctamente dicha aplicación.

Además, me deberéis subir un pantallazo de la aplicación en la cual deberéis **cambiar el compre del cinema en vez de UTN pondréis vuestro nombre**.

