## Modulo 2  Introducción a Java Web con JSP

**Introducción al mundo Web con Java**

Cuando navegamos por internet, interactuamos con muchas páginas y aplicaciones WEB.

Cuando se tratan de **páginas web** estamos hablando de aquellas que por lo general tienen **contenido estático** desarrollado en**HTML y CSS**.

Por otro lado, cuando hablamos de **aplicaciones web**, hacemos referencia a aquellas páginas que tienen **dinamismo** y ya incluyen también algún**lenguaje de programación** como tal, permitiendo **cambiar** según lo que necesitemos, como por ejemplo las tiendas online o las redes sociales.

**Java EE (Enterprise Edition)**

**Java EE** (Java Platform, Enterprise Edition), es una plataforma de programación desarrollada por Oracle (anteriormente Sun Microsystems) para crear **aplicaciones empresariales** y **servicios web complejos y robustos**.

**Java EE** se enfoca principalmente en el **desarrollo de aplicaciones empresariales y web**, abordando aspectos como la lógica de negocio, la seguridad, la gestión de datos y más.

**Java EE** se diseñó para crear aplicaciones que se ejecuten en servidores, proporcionando un conjunto de herramientas, bibliotecas/librerías y especificaciones que permiten a los desarrolladores construir aplicaciones que pueden manejar tareas empresariales críticas y operar en entornos de alto rendimiento y escalabilidad.

**¿Qué es JSP (Java Server Pages)?**

**Java Server Pages (JSP)** es una tecnología que permite crear aplicaciones web dinámicas basadas en **HTML** y **XML**, entre otros tipos de documentos.

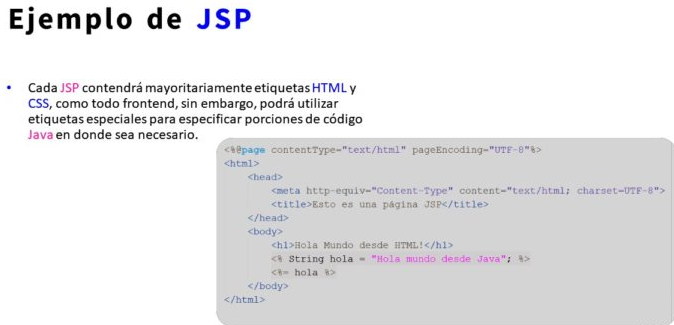
**JSP** tiene la particularidad de que requiere de un **servidor web** que sea compatible con **contenedores servlet** como por ejemplo: **Apache Tomcat o Glassfish**.

Si bien **JSP** no es una tecnología nueva, tiene la ventaja de la portabilidad de Java y permite comprender los fundamentos de la programación web.

**Principales Usos de JSP:**

**JSP** como tecnología para el desarrollo de aplicaciones web tiene diferentes usos, entre los cuales se destacan:

1. **Aplicaciones de Empresas**: Muchas empresas utilizan Java EE y JSP para construir aplicaciones internas que les ayudan a manejar sus operaciones, como llevar un registro de sus empleados o gestionar inventarios/stock.
2. **Tiendas Online**: Cuando compramos en línea, interactuamos con tiendas virtuales que en muchas situaciones están construidas con Java Web y JSP. Estas tecnologías permiten mostrar productos, agregarlos al carrito de compras y realizar pagos de manera segura.
3. **Homebanking o sistemas bancarios**: Muchas enteidades bancarias o financieras utilizan para sus aplicaciones de homebanking a Java Web  JSP.  ¿Por qué se suele usar JSP en este tipo de sistemas? Porque estas tecnologías permiten manejar la información de forma segura.



**¿Qué es Apache Tomcat?**

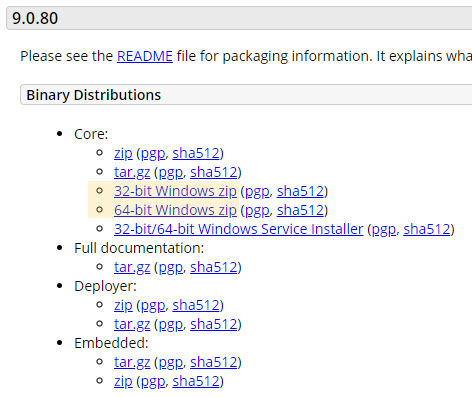
**Apache Tomcat** es un servidor web y contenedor de **servlets** de código abierto desarrollado por la **Apache Software Foundation**. Su función principal es proporcionar un entorno en el que las aplicaciones web Java, especialmente aquellas basadas en tecnologías como **JavaServer Pages (JSP)** y **Servlets**, puedan ejecutarse de manera eficiente.

Tomcat es especialmente conocido por su compatibilidad con **JavaServer Pages (JSP)**. Como ya sabemos, JSP es una tecnología de desarrollo web que permite a los programadores crear páginas web dinámicas utilizando código Java incrustado dentro de etiquetas HTML.

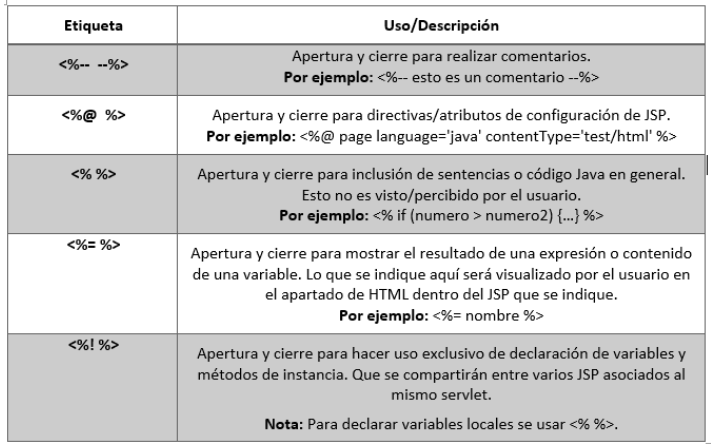
Por otro lado, Tomcat también permite el despliegue de aplicaciones web en forma de archivos **WAR (Web Application Archive)**. Éstos, contienen todos los recursos y clases necesarios para que una aplicación web funcione correctamente. Tomcat toma estos archivos, los despliega y los pone a disposición de los usuarios finales.

**¿Cómo instalar Apache Tomcat?**

1. En primer lugar debemos dirigirnos a la [**web oficial de Apache**](https://tomcat.apache.org/download-90.cgi). En caso de no tener un link directo, podés encontrar la web de forma sencilla colocando «Descargar Apache Tomcat 9» en Google.
2. Una vez en la sección de descargas, no s dirigimos al apartado de «Binary Distributions» (distribuciones binarias) y elegimos la que corresponda a nuestro ssitema operativo. En caso de ser Windows, seleccionaremos la versión zip de 32 o 64 bits, dependiendo de nuestras configuraciones.



3. Descargamos el archivo .zip, y lo descomprimimos en nuestra PC. Importante guardar la carpeta en un lugar que tengamos acceso siempre. Evitar carpeta de descargas o lugares similares donde podamos eliminar accidentalmente.4. Solo queda configurar el servidor en nuestro IDE junto con la creación de nuestro primer JSP.



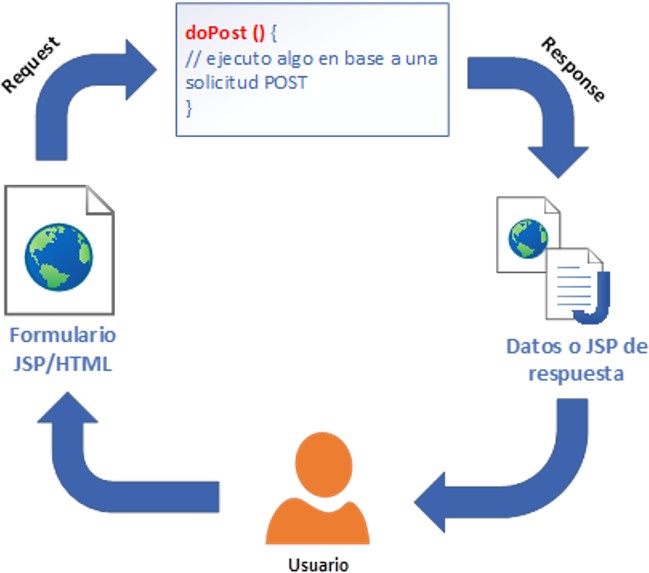
La principal ventaja que tiene JSP frente a otros tipos de implementaciones, es el hecho de que Java es un lenguaje que va más allá del mundo Web. Esto hace que JSP tenga las características de portabilidad (ejecución en diferentes sistemas operativos) que posee Java.

Por otra parte, para cumplir correctamente con el modelo de capas y el traspaso de información desde un cliente a un servidor, JSP se vale de una ayuda muy importante, que son los Servlets.

Servlets

La definición más común de un servlet es que se trata de una clase Java que se utiliza para poder ampliar las capacidades de un determinado servidor. Su principal característica es la de ser un punto intermedio entre una página JSP y el servidor web (donde generalmente se encuentra el servicio o la lógica de negocio de una aplicación).

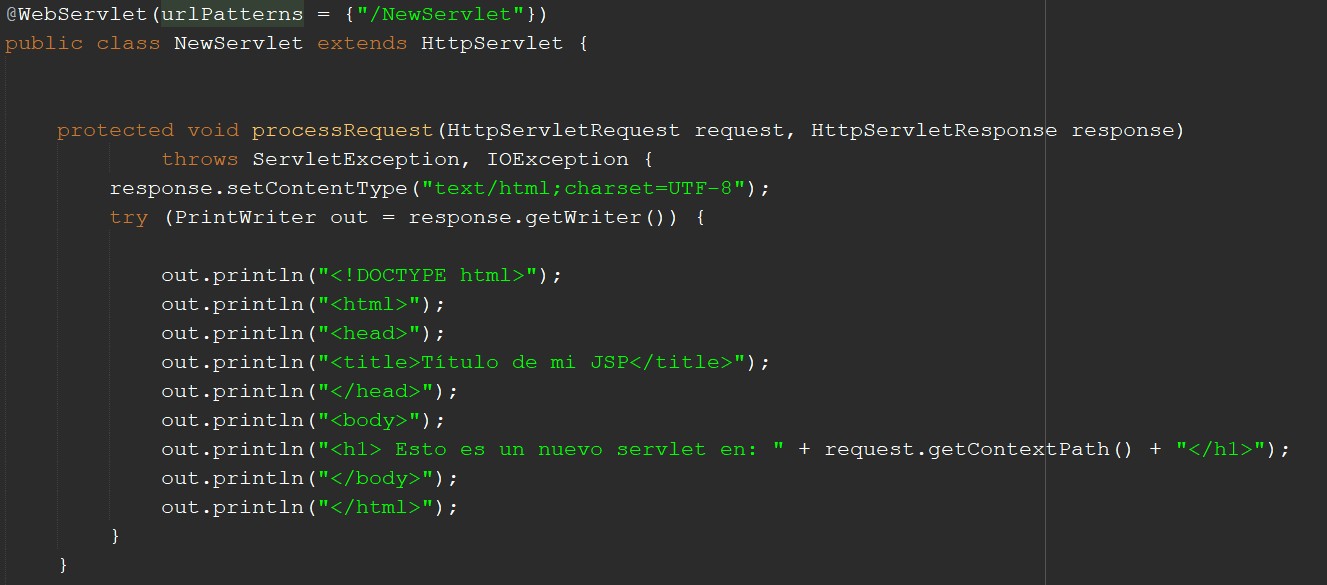
Un servlet se encarga de recibir peticiones o request desde un cliente, tratarlas y analizar si es necesario realizar alguna solicitud en particular o brindar una determinada respuesta o response. Para poder tratar cada una de las peticiones, utiliza una serie de métodos donde dependiendo del verbo por el cual se reciba la petición (GET, POST, PUT, DELETE, etc) se ejecutará la especificación del que corresponda. Un ejemplo con el método POST puede visualizarse a continuación:



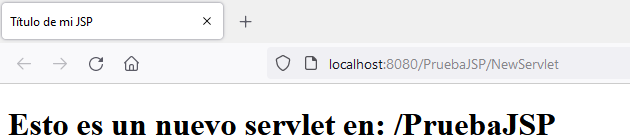
**¿JSP como servlet o servlet como JSP?**

Aunque JSP y servlets parecen a primera vista tecnologías distintas, en realidad el servidor web traduce internamente el JSP a un servlet, lo compila y finalmente lo ejecuta cada vez que el cliente solicita la página JSP. Por ello, en principio, JSPs y servlets ofrecen la misma funcionalidad, aunque sus características los hacen apropiados para distintos tipos de tareas.

Los JSP son mejores para generar páginas con gran parte de contenido estático. Un servlet que realice la misma función debe incluir gran cantidad de sentencias del tipo out.println() para producir el HTML. Por el contrario, los servlets son mejores en tareas que generen poca salida, datos binarios o páginas con gran parte de contenido variable.

Un ejemplo de un servlet transformándose en JSP:

El ejemplo a nivel código, se vería en el navegador tal y como se muestra a continuación:



En proyectos más complejos, lo recomendable es combinar ambas tecnologías: los servlets para el procesamiento de información y los JSP para presentar los datos al cliente. Esta última opción es la que veremos en los ejemplos que se desarrollarán.

**Ciclo de vida de un Servlet**

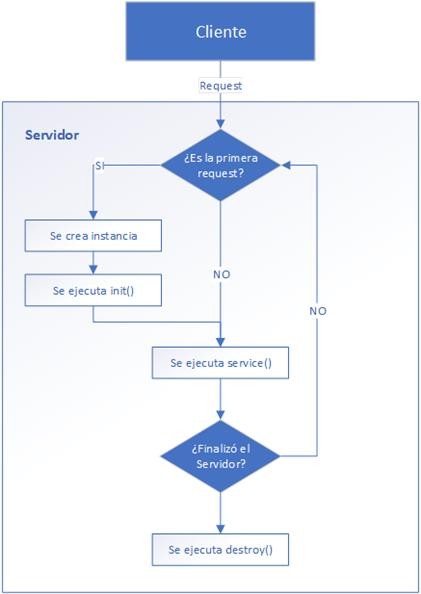
Todo servlet tiene un ciclo de vida dentro de una aplicación web. Cada etapa del ciclo de vida dependerá estrechamente de las operaciones que esté haciendo el servidor web donde se encuentre alojado y se puede simplificar en 3 grandes pasos o momentos:

* **Inicio del servidor**: Al comenzar la ejecución de un servidor se producen dos acciones en un servlet:
* Se crea una instancia del Servlet
* Se inicia el Servlet mediante el método **init ()**
* Llegada de una request o solicitud:

o Se hace llamado al método **service()**, el cual implementa un nuevo thread (hilo) para atender a dicha solicitud.

* Cierre o apagado del servidor:
* Se hace llamado al método **destroy()**, el cual se encarga de destruir el servlet.

Un resumen de este ciclo de vida puede visualizarse en el siguiente diagrama de flujo:



**Métodos de Servlets**

Los servlets tienen diferentes métodos que pueden ser utilizados dependiendo del tipo de solicitud que reciban por parte del cliente, sin embargo, los dos más utilizados son el método doGet(), para recibir peticiones mediante GET y doPost(), para recibir peticiones mediante POST.

**doGet():** Es el método encargado de recibir las solicitudes que provienen mediante GET. Generalmente recibe los parámetros desde la URL de la petición y su principal función es la de solicitar datos del servidor y devolverlos al cliente.

**doPost():** Es el método encargado de recibir las solicitudes que provienen mediante POST. Los parámetros, objetos o datos pueden provenir en el header o body de una solicitud, a partir del envío (submit) de un formulario HTML desde JSP. Su principal función es la de obtener datos desde el cliente para generar cambios en el servidor.