# Java y Aplicaciones Avanzadas en Internet
# Cursada 2025

**Profesores: Claudia Queiruga  y Jorge Rosso**
**JTP: Diego Bellante**
**Ayudantes: Andrés Vazzano, Francisco Blanco y Sebastián Villena**

# ¿Qué aprenderemos? ¿Cómo nos organizamos?

## Objetivos de aprendizaje

Las y los estudiantes podrán:

- **Construir aplicaciones web** que combinen **tecnologías JAVA server-side** para el backend y **SPA (Single Page Application)** para el frontend.
- **Abordar** un **proyecto de software sobre una problemática real usando dichas tecnologías**, a partir de una situación-problema acercada por la Secretaría de Extensión.

## Metodología de trabajo

A lo largo de la cursada se trabajará:

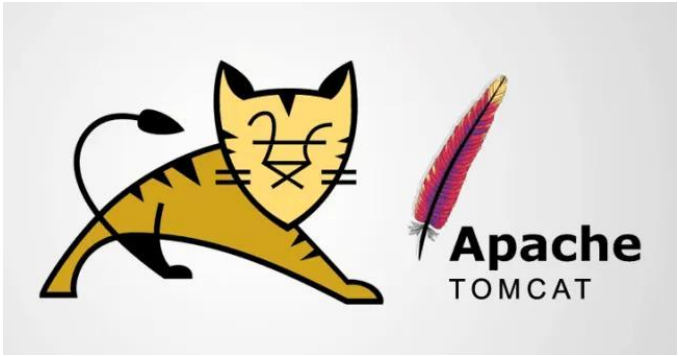
- **Contenidos teóricos** sobre desarrollo web con tecnologías JAVA y SPA (Angular).
- En el **desarrollo de un proyecto:** con 6 entregables en fechas pautadas durante las clases prácticas. Las entregas se realizarán en tareas en el aula virtual
- En **equipos de 2 estudiantes** para el desarrollo del proyecto. Cada **equipo** tendrá asignado un docente que tutoreará su trabajo.

## Evaluación

Promoción directa.

# ¿Qué herramientas usaremos?





*Herramientas para integración continuo: devops*



# JEE-Jakarta EE
# Servlets





# Tecnologías Java

Es una combinación de un **lenguaje de programación** y una **plataforma**.

- **El Lenguaje de Programación JAVA**

**Plataforma**

- **Una Máquina Virtual**

  Es el fundamento de la plataforma Java y sus características fundamentales son: **a)** está implementada para múltiples SO y hardware, favoreciendo la **compatibilidad binaria de código JAVA**. Las aplicaciones funcionan en forma consistente sobre distintas implementaciones de la máquina virtual. **b)** provee un control estricto del código binario, permitiendo una **ejecución segura** de código no-confiable.

- **Una Interface de Programación estándar y amplia (API)**

  Es una colección de componentes de software listas para usar. Provee un amplio soporte para desarrollar programas JAVA, desde librerías para escribir interfaces de usuario gráficas, soporte para criptografía, acceso a BD, conectividad, facilidades de internacionalización, etc.

Actualmente **Java está presente en todos lados**: la plataforma JAVA está disponible para una amplia gama de computadoras, desde dispositivos móviles como celulares y tablets, hasta computadoras personales y servidores de alta prestaciones.

# Los sabores de JAVA

## Java Standard Edition (JSE) - JAVA para aplicaciones de escritorio

Está diseñada para aplicaciones de escritorio. Se ejecuta sobre todos los SO. Provee herramientas de compilación y ejecución, una máquina virtual y API básicas. Esta API provee la funcionalidad principal del lenguaje: define todos los tipos básicos y objetos JAVA hasta clases de alto nivel para conectividad, seguridad, acceso a BD, desarrollo de GUI, parsing XML, etc

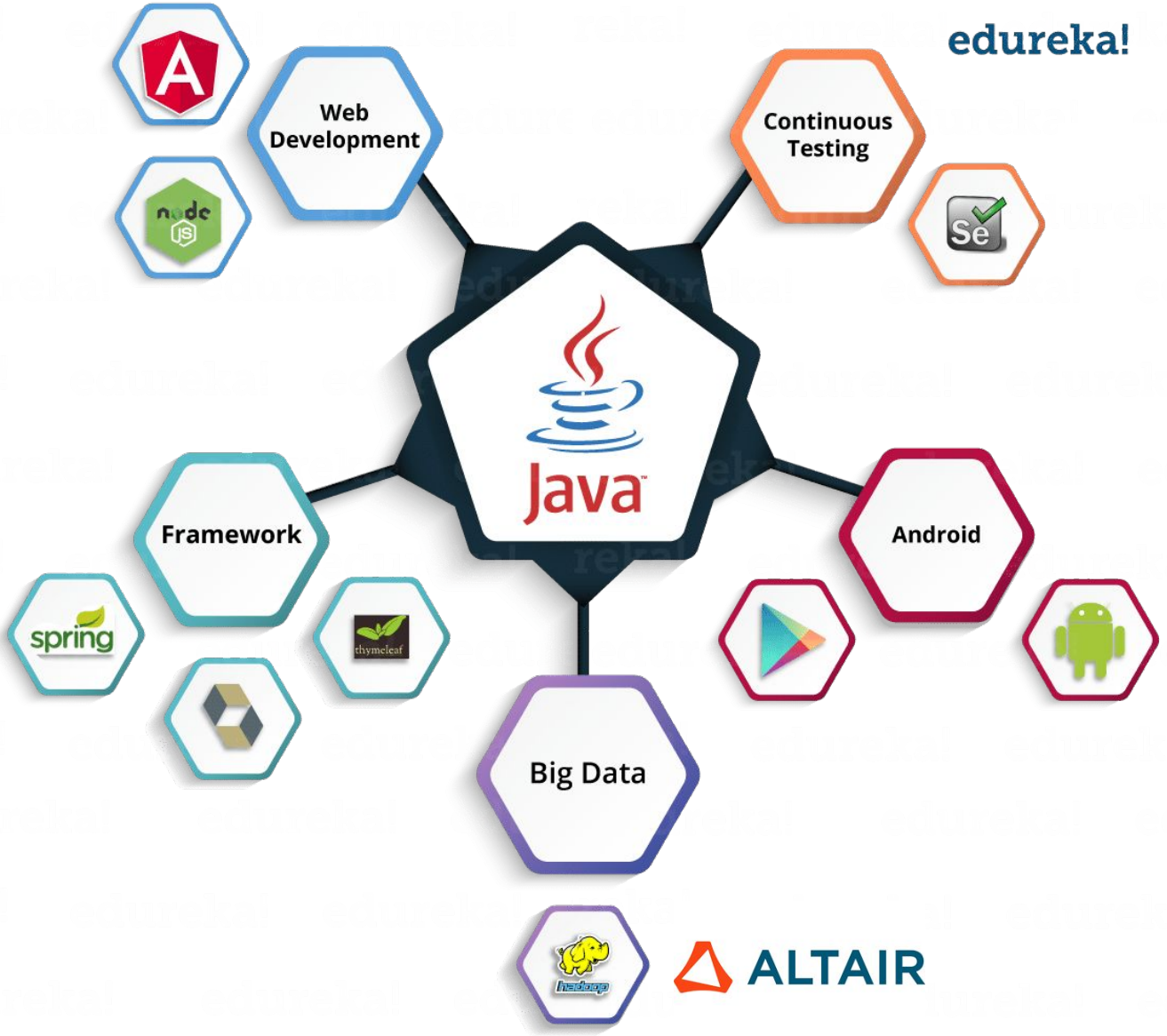
## Java Enterprise Edition (JEE) - JAVA del lado del servidor

Es una plataforma multiusuario, distribuida, para desarrollo de aplicaciones empresariales sobre la Web. Está basada en JSE y agrega APIs para realizar computación escalable, confiable y segura del lado del servidor.

## Java Micro Edition (JME) - JAVA para aplicaciones móviles

Es una versión simplificada de JSE para dispositivos móviles. La falta de acceso al hardware del dispositivo y la ejecución en un entorno controlado (sandbox) da como resultado aplicaciones móviles que no aprovechan las ventajas propias del mundo móvil. Esto hizo que no fuera adoptada como plataforma de desarrollo de aplicaciones móviles. **Android es la plataforma de desarrollo y ejecución de aplicaciones móviles nativas escritas en JAVA. Actualmente está siendo reemplazado por Kotlin**

# ¿Dónde se usa JAVA?



# Comunidad de desarrollo de JEE

Actualmente el desarrollo de JEE está liderado por la comunidad global **Eclipse Foundation**.

Es una **comunidad abierta y colaborativa formada por organizaciones e individuos**, es un **ecosistema de innovación abierta**, que lidera el desarrollo de **código fuente abierto**.

## ¿Qué es el Jakarta EE?

- Es un **proyecto** de la comunidad **Eclipse Foundation** que lidera **el desarrollo de especificaciones de tecnologías JAVA backend.**
- Se trata de un ecosistema del que forman parte empresas líderes de la industria de software como IBM, Oracle, redHat, etc.
- A partir de la versión 9 de JEE, su desarrollo comenzó a liderarlo la comunidad Jakarta, antes lo lideraba el **JCP.** Esto impactó en el nombre, pasó a llamarse Jakarta EE en lugar de JEE. La versión actual es **Jakarta EE 10**.

## ¿Para qué sirven las especificaciones?

Contar con una especificación transforma a todas las tecnologías JAVA en **estándares**. Cada fabricante de software desarrolla su implementación respetando la especificación. Se garantiza **compatibilidad** y **portabilidad**. Punto clave: **independencia del fabricante**.

# Aplicaciones y Plataformas Empresariales

- Una **aplicación empresarial** es una aplicación que implementa la lógica de negocios de una organización y cumple con determinados requerimientos no funcionales, por ejemplo: ser **distribuída**, de **alta disponibilidad**, **tolerante a fallas**, proveer **acceso seguro y transaccional**, proveer soporte para **concurrencia** y **acceso multiusuario**, ser **escalable**, **interoperable**, etc.
- Una **plataforma empresarial** es una infraestructura de software que provee todos los **servicios necesarios para ejecutar una aplicación empresarial**.
- **Jakarta EE (antes JEE) es la plataforma empresarial de JAVA** usada para desarrollar, desplegar y gerenciar aplicaciones empresariales. La **arquitectura** de las **aplicaciones empresariales JAVA** está basada en **componentes** de software estándares, modulares y distribuidos. Jakarta EE ofrece a las componentes, implícitamente, de un conjunto de servicios que son específicos de las aplicaciones empresariales.

# Jakarta EE: la Plataforma Empresarial Java

**Jakarta EE es un conjunto de especificaciones** que definen el desarrollo, despliegue y gerenciamiento de aplicaciones JAVA distribuídas, centradas en servidor. **Jakarta EE NO es un producto.**

La comunidad de **Eclipse Foundation** es la encargada de **definir las especificaciones**, que incluyen:

- Documentos de especificaciones y API: definición y descripción de paquetes, interfaces, clases JAVA.
- Kit de compatibilidad tecnológica (TCK): ofrece un conjunto de test de compatibilidad usados para probar el código implementado. Usado por los implementadores de servidores.
- Implementación de referencia: Eclipse GlassFish

Garantiza **compatibilidad entre diferentes fabricantes** y **portabilidad de aplicaciones. La implementación de las APIs es responsabilidad del fabricante.** Promueve “Write Once, Run Anywhere” del lado del servidor.

La versión actual es Jakarta EE 10: **https://jakarta.ee/specifications/platform/10/**

# Arquitectura de aplicaciones multitiered Java

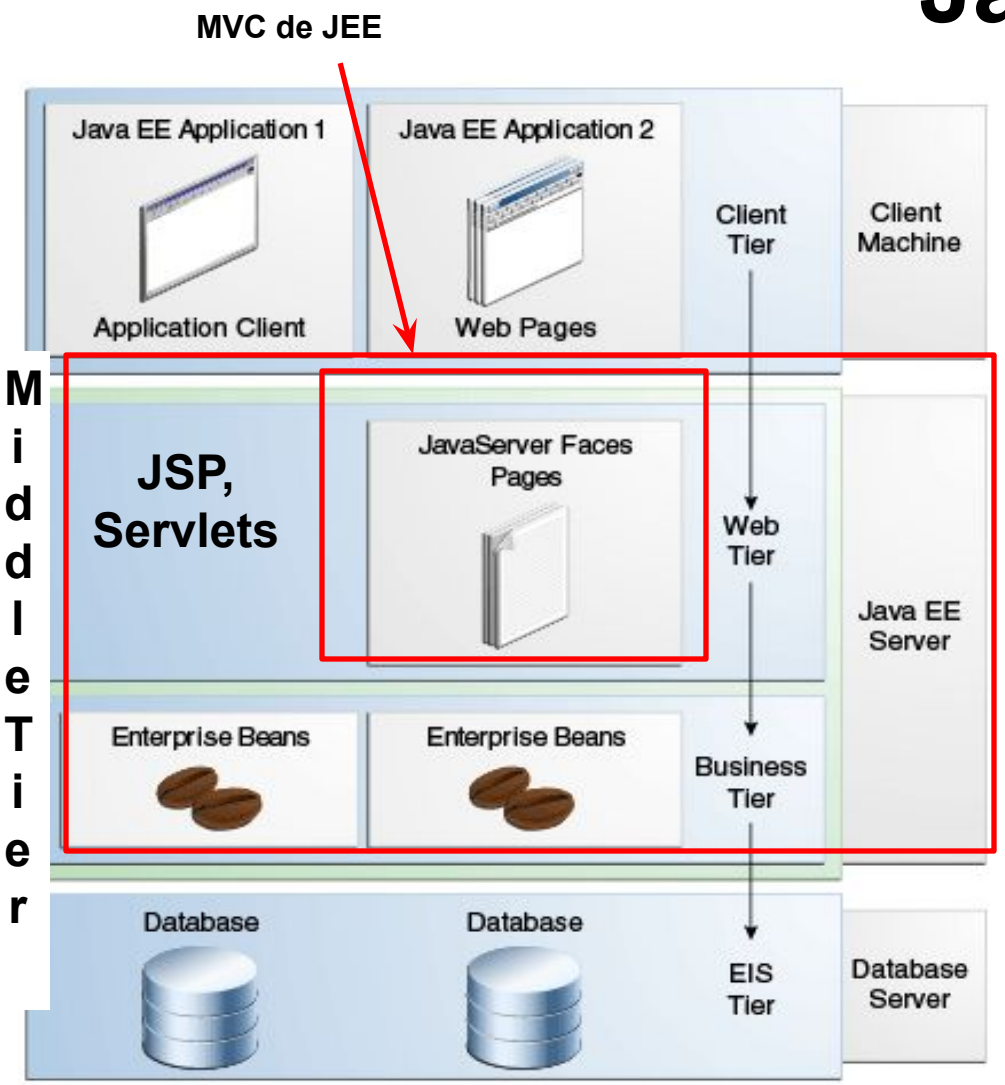



Figura extraída del The Java EE Tutorial Release 7 (Oracle): https://docs.oracle.com/javaee/7/tutorial/overview003.htm

**Layer:** es una **capa lógica**, se refiere a la organización del código.

**Tier:** es una **capa física**, es el lugar dónde se ejecuta el código.

**Los tiers son el lugar dónde se despliegan y ejecutan los layers**

Jakarta EE (o JEE) ofrece un framework para el desarrollo de aplicaciones empresariales que promueve la **separación de la funcionalidad de la aplicación en tiers.** Típicamente tienen 3 tiers:

**Client Tier:** es una aplicación cliente que hace peticiones al middle tier, se ejecuta en una computadora cliente.

**Middle Tier:** maneja las peticiones de los clientes, procesa datos de la aplicación, los guarda en la BD, devuelve la respuesta al cliente. Se ejecuta en un servidor de aplicaciones JAVA (ej. Tomcat)

**Data Tier o EIS Tier:** motor de BD, sistema legados, etc. alojado en servidores diferentes del servidor JEE.

**El desarrollo de las aplicaciones Jakarta EE se enfoca en el middle tier**

# Componentes y Contenedores

Las **aplicaciones Java server-side** están formadas por **componentes** JAVA que se **ejecutan en tiers específicos en el entorno de contenedores.**

**Una componente es una unidad de software que cumple ciertas restricciones y se ensambla en una aplicación empresarial; se comunica con otras componentes y se ejecuta en un contenedor JAVA. Son creadas por los desarrolladores e implementan la lógica de negocios.**

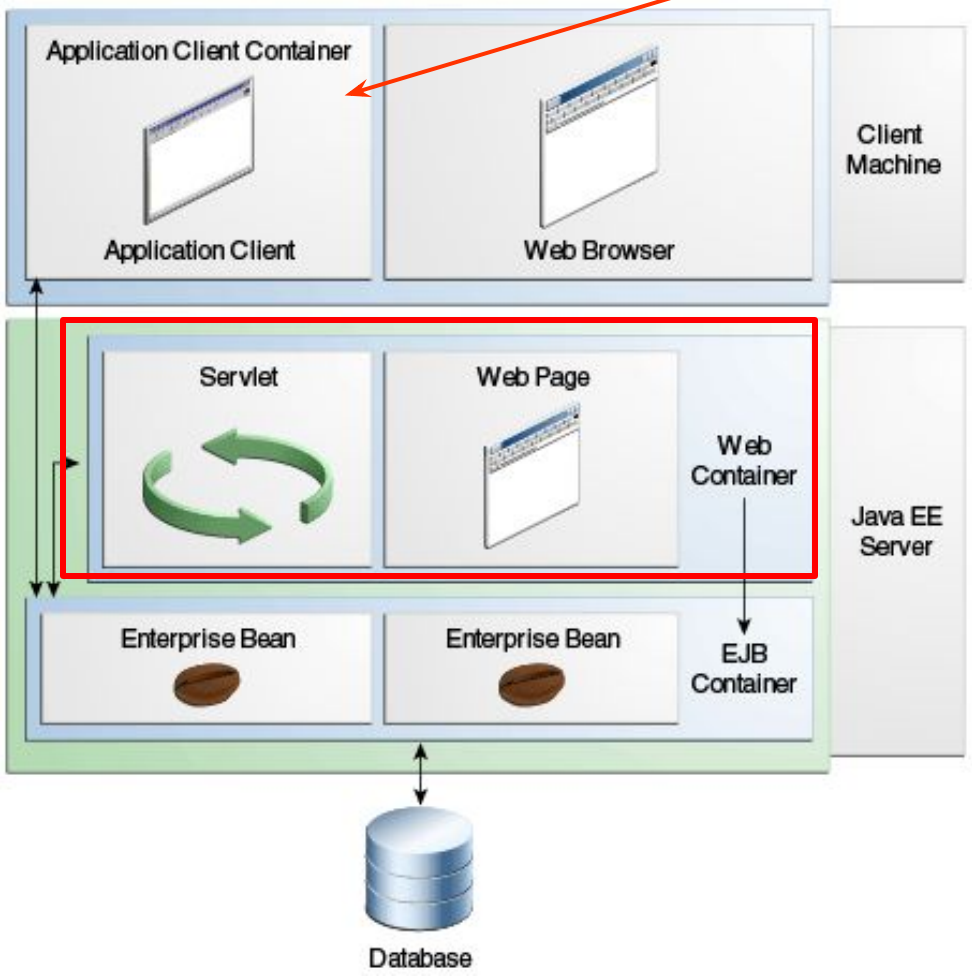
La especificación define las siguientes componentes:

- **Clientes web** y **aplicaciones de escritorio** JAVA que se ejecutan en una computadora cliente.
- **Servlets**, páginas web construidas con JSF (JavaServer Faces) y JSP (JavaServer): son componentes web que se ejecutan en un servidor JEE.
- **Enterprise beans**: son componentes que se ejecutan en un servidor JEE.

Las **componentes se escriben en JAVA** y se compilan de la misma manera que cualquier programa JAVA. La diferencia entre componentes Jakarta EE y clases JAVA estándares es que **las componentes se ensamblan dentro de una aplicación empresarial**, se verifica que estén bien formadas y, que sean compatibles con la especificación. Se despliegan para producción en la computadora (servidor) y **son gerenciadas por un contenedor.**

**Un contenedor es un entorno de ejecución seguro y controlado para la ejecución de componentes; ofrece servicios de infraestructura a las componentes (gestión de recursos, seguridad, administración de transacciones, etc). Es el responsable de gerenciar el ciclo de vida de las componentes.**

# Tipos de Contenedores JEE™




**Contenedor de la aplicación cliente**: gerencia la ejecución de la aplicación cliente JAVA de escritorio. La aplicación cliente y su contenedor se ejecutan en la computadora cliente.

**Contenedor WEB:** gerencia la ejecución de las componentes del tier web (servlets, JSF, JSP, listeners, filtros, etc).

**El contenedor WEB se ejecuta en un servidor Jakarta EE.**

Figura extraída del The Java EE Tutorial Release 7 (Oracle): https://docs.oracle.com/javaee/7/tutorial

# Perfiles Jakarta EE

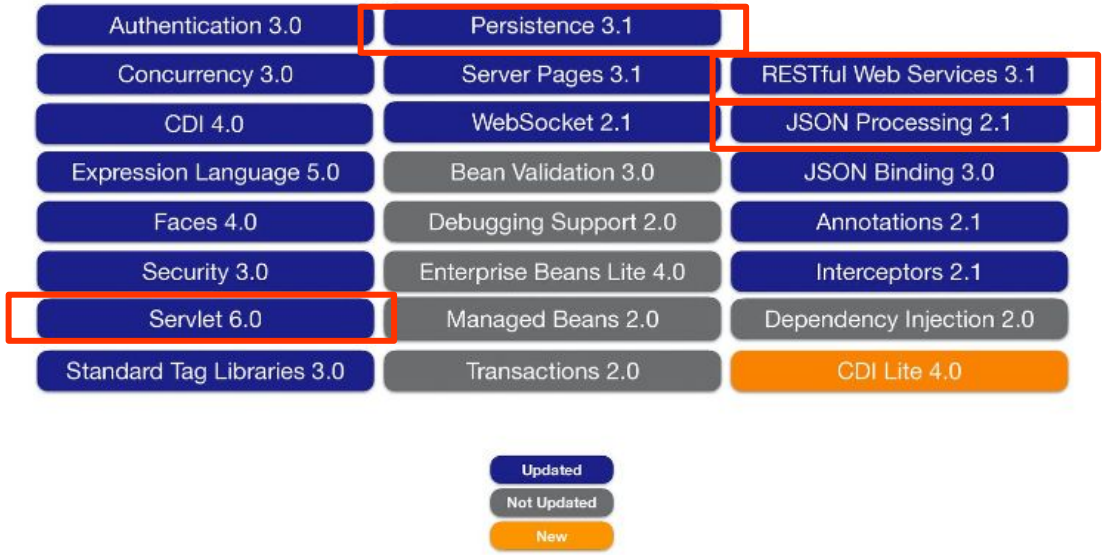
## ¿Qué son los perfiles Jakarta EE?

- Un **perfil** es una **configuración de la plataforma** adecuada para una **clase particular de aplicaciones**. Contiene un subconjunto de las tecnologías ofrecidas por la plataforma, dejando de lado tecnologías que la plataforma soporta pero que no son útiles en un ámbito concreto.
- Un **perfil** puede definir ciertas **tecnologías como opcionales** y en este caso los productos que implementen el perfil pueden incluir o no la tecnología en cuestión.
- Un **producto puede implementar dos o más perfiles de Jakarta EE**, o la plataforma completa.

El perfil con el que trabajaremos es con el **perfil web**:

https://jakarta.ee/specifications/webprofile/10/

# El perfil web Jakarta EE




Fuente: Web Profile Jakarta EE 10 (Ivar Grimstad).

**Tecnologías y componentes**:

**Servlets:** extienden las capacidades de un server HTTP que aloja aplicaciones.
**JSF:** es el framework MVC estándar propuesto por JEE, construido sobre la tecnología de Servlets y JSP. **No lo vamos a usar**
**JSP**: son documentos de texto que se ejecutan como servlets y que ofrecen una aproximación más natural a la generación de páginas web. **No lo vamos a usar**

**Web Profile es una especificación de Jakarta EE que define un subconjunto de tecnologías enfocadas en aplicaciones web**

**APIs:**

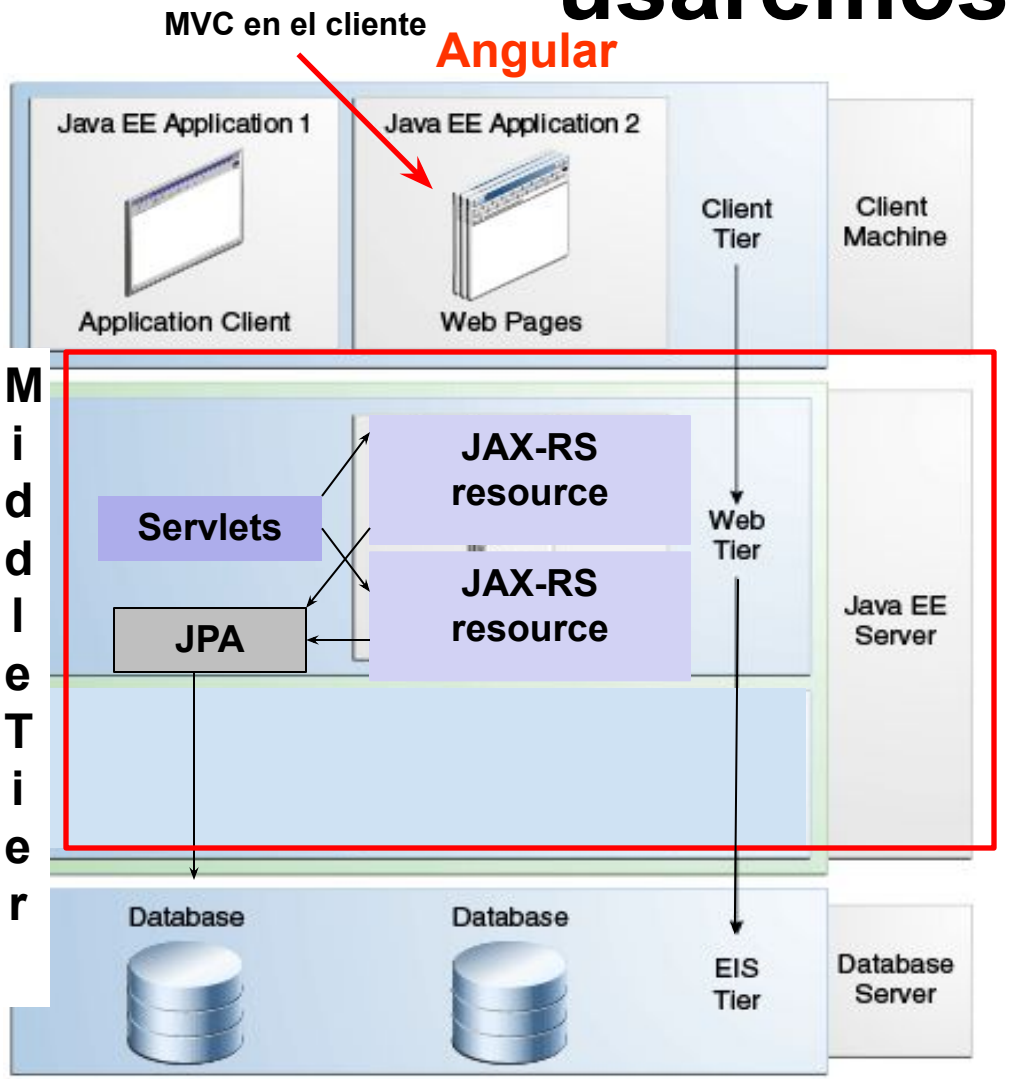
**Java API para RESTful Web Services (JAX-RS)**: permite desarrollar servicios web basados en RESTful.

**Java Persistence API (JPA)**: ofrece un mapeo objeto-relacional para manejo de datos relacionales en aplicaciones JAVA.

**Java Database Connectivity (JDBC)**: incluida en JSE, ofrece acceso a BD relacionales desde aplicaciones JAVA.

**JSON-P**: JSON es un formato de intercambio de texto derivado de JavaScript usado para conectar aplicaciones. Permite parsear, transformar y consultar datos JSON .

# Arquitectura de aplicaciones multitiered que usaremos - Backend




En esta arquitectura, el MVC está del lado del navegador web: consume datos que generan las componentes del "middle tier" (servicios Rest).

**Angular** es el framework SPA del lado del cliente que usaremos para consumir los datos generados por los recursos o servicios JAX-RS.

**Esta arquitectura es la que usaremos para el desarrollo del proyecto de la cátedra.**

Figura extraída del The Java EE Tutorial Release 7 (Oracle):
https://docs.oracle.com/javaee/7/tutorial

# Servidores Jakarta EE

**Servidores certificados que cumplen la especificación Jakarta EE**

| | Código Fuente Abierto | Propietarios |
|---|---|---|
| **Contenedor web (solamente)** | **Tomcat**<br>Jetty | |
| **Contenedor J2EE completo** | GlassFish<br>JBoss<br>Apache Geronimo<br>JOnAS | IBM WebSphere Application Server<br>Oracle WebLogic Server |



# Servlets

- Un **servlet** es un **componente web** escrito en Java gerenciado por un **Contenedor Web**. Genera contenido dinámico y realiza procesamiento **middleware**.
- En forma idéntica a otras componentes Java estándares, los **servlets** son clases Java independientes de la plataforma, se compilan a código de bytes (*bytecodes*), se cargan dinámicamente y se ejecutan en un servidor web.
- Un **servlet** interactúa con el cliente usando el **paradigma pedido-respuesta**, usado comúnmente en servidores web: un cliente envía un mensaje de pedido al servidor y el servidor devuelve un mensaje de respuesta al cliente.
- Lo destacado de los servlets es el uso de la plataforma Java y en la interacción con el Contenedor Web:
  - La **Plataforma Java** ofrece al programador de servlets de una API robusta basada en POO, neutralidad de la plataforma, capacidad de *garbage collector* y todas las características de seguridad de la JVM.
  - El **Contenedor Web** es la plataforma de ejecución que provee **servicios a los servlets** tales **como conectividad con la red**, **entrega de peticiones**, **producción de respuestas**, **concurrencia**, **seguridad**, **gerenciamiento del ciclo de vida del servlet**. Además del **acceso a múltiples APIs**
- La versión actual de servlets es la 6.0 y forma parte de Jakarta EE 10.

# Contenedor Web

El **Contenedor Web** forma parte del servidor Jakarta EE y ofrece servicios de infraestructura a los servlets.

El **Contenedor Web** está construido sobre la plataforma JSE™, **implementa la API de servlets** y todos los servicios requeridos para procesar pedidos HTTP.

El Contenedor Web es responsable de:

- la **conectividad** con la red
- **capturar los requerimientos HTTP** (MIME-type), traducirlos a objetos que el servlet entienda, entregarle dichos objetos al servlet quien los usa para producir las respuestas y **generar una respuesta HTTP** en un formato correcto (MIME-type)
- **gerenciar** el ciclo de vida del servlet
- **manejar** errores
- **proveer seguridad**

El **Contenedor Web** interactúa con los servlets invocando **métodos de gerenciamiento o métodos callback**. Estos métodos definen la interface entre el Contenedor y los servlets.

Cada fabricante tiene su propia implementación de la API de servlets.

Nosotros usaremos TOMCAT 10 que implementa Jakarta EE 9.1 (Servlet 5.0).

# Servlets

## Ciclo de Vida

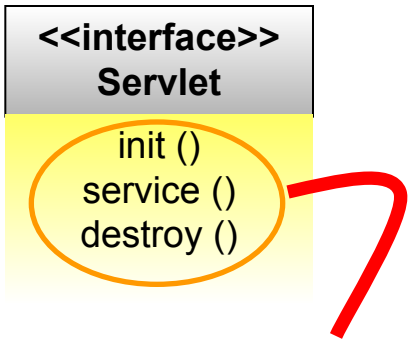
El **Contenedor Web** gerencia el ciclo de vida de cada servlet invocando a **tres métodos** definidos en la interface **jakarta.servlet.Servlet**: init(), service() y destroy().

El **Contenedor Web es responsable de la carga e instanciación de los servlets**: puede ocurrir en el arranque o cuando una aplicación se actualiza o se recarga, o se posterga hasta que el servlet es requerido por primera vez.

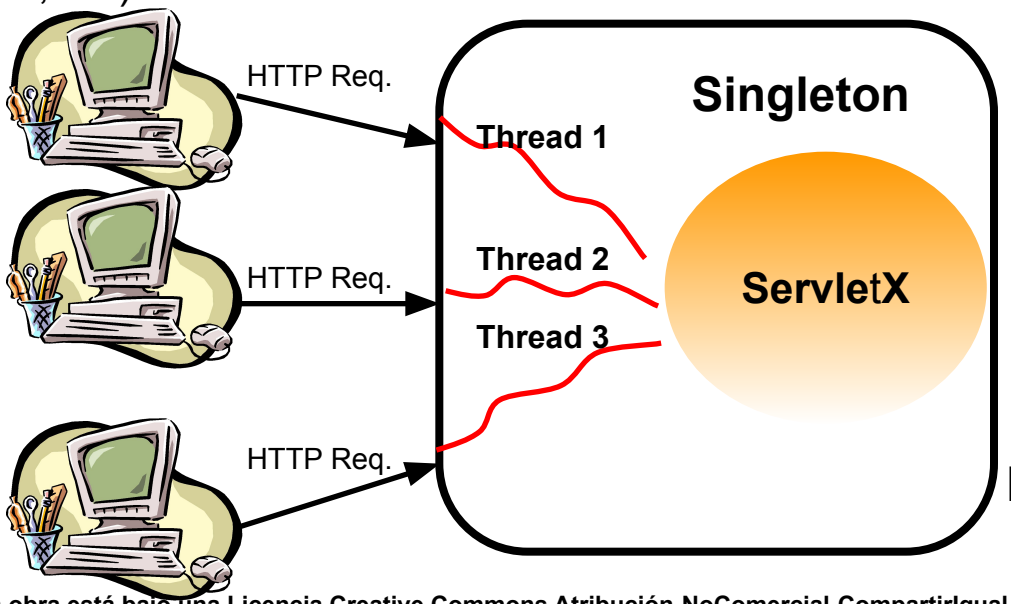
El **Contenedor Web** crea **una única instancia de cada servlet declarado** en la aplicación web.

El **Contenedor Web** maneja los **requerimientos concurrentes** a un mismo servlet, ejecutando el método **service()** concurrentemente en múltiples threads Java.

El programador de servlets debe considerar los efectos laterales que tiene el procesamiento concurrente y tomar las previsiones necesarias sobre los recursos compartidos (acceso a v. de instancia, v. de clase, recursos externos como archivos, etc).




**Las 3 etapas del ciclo de vida de un servlet**



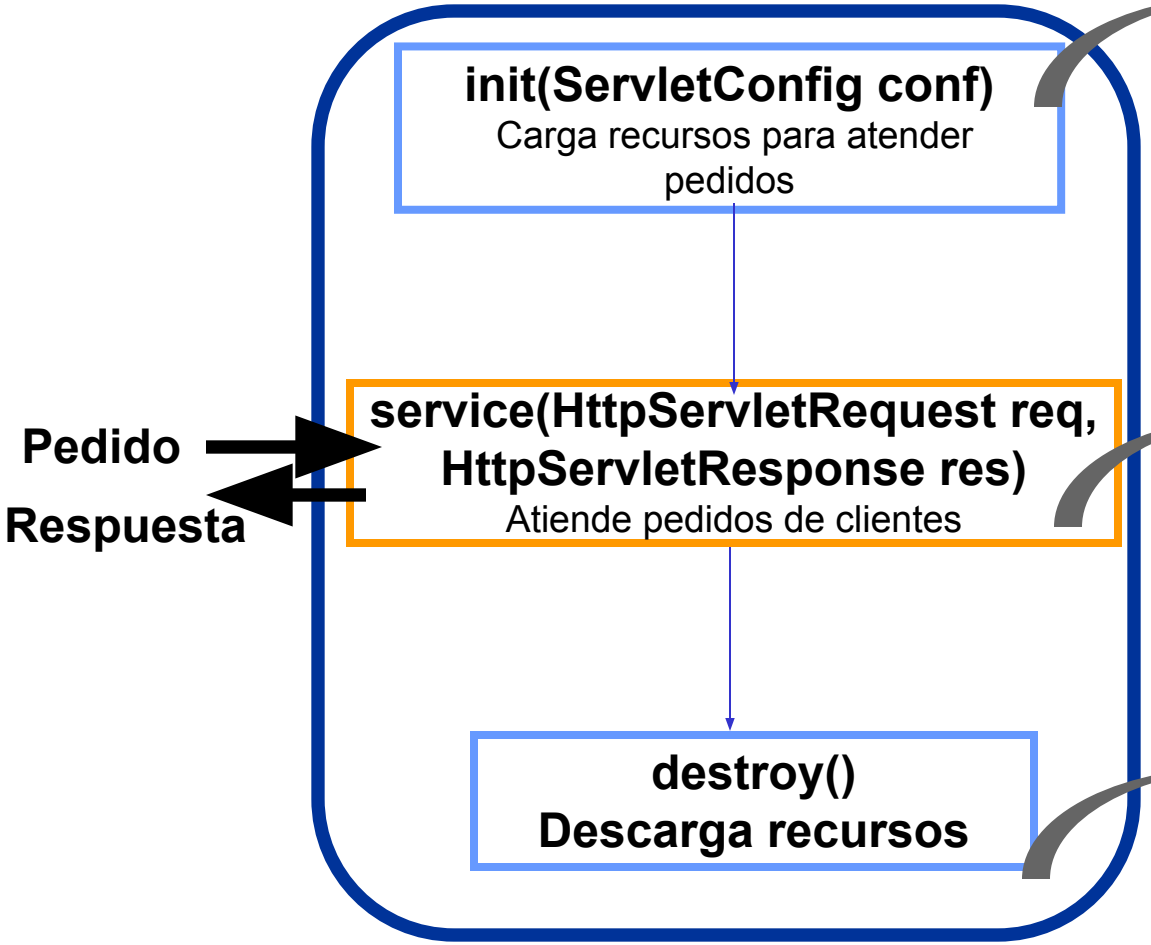

Los servlets son **objetos multithreaded**

**No es Thread-Safe**

# Servlets
## Ciclo de Vida

**Etapas del ciclo de vida de un servlet**

- **Carga** de la clase que implementa el servlet e **instanciación**
- **Inicialización**
- **Atención de pedidos**
- **Finalización de atención de pedidos**

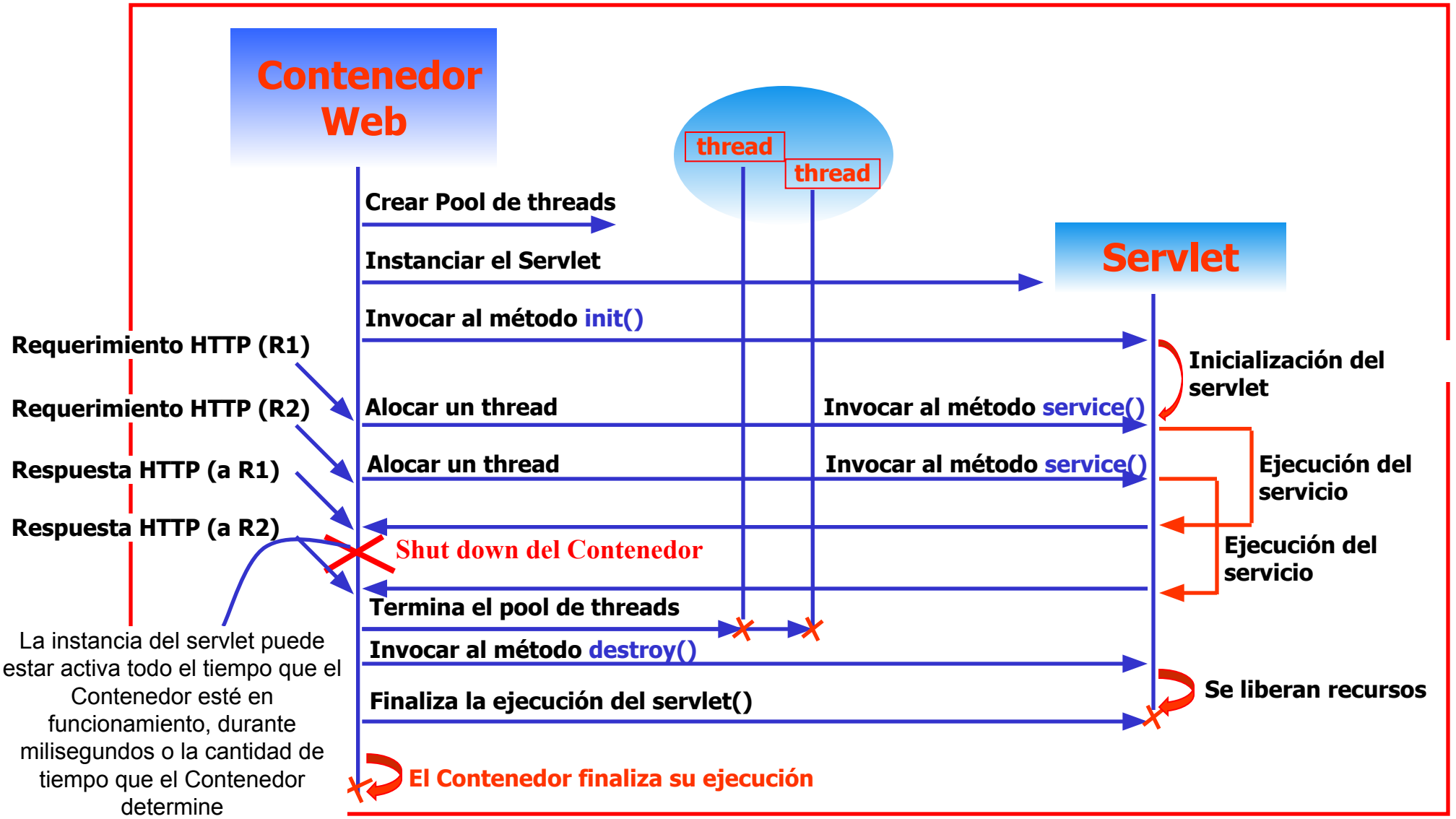



- El Contenedor crea un objeto **ServletConfig**, que es pasado como parámetro al método **init** y de esta manera el servlet puede acceder a parámetros de inicialización (de la forma nombre-valor).
- Si el método **init() no termina exitosamente** (dispara una excepción), el **servlet no es puesto en servicio** y podría ser liberado por el Contenedor

- Atiende requerimientos entrantes de clientes. El Contenedor crea dos objetos que pasa como parámetro al servlet: **HttpServletRequest** y **HttpServletResponse**. Estos objetos representan el requerimiento y la respuesta del cliente respectivamente

- Lo ejecuta el Contenedor antes de remover al servlet

# Servlets
# Ciclo de Vida



# La interface de Programación de Servlets

Las clases e interfaces para implementar servlets están agrupadas en dos paquetes:

- **jakarta.servlet**: contiene la interface básica de servlets, llamada **Servlet**.
  - La inteface **jakarta.servlet.Servlet** es la abstracción central de la API de servlets.
- **jakarta.servlet.http**: contiene la clase **HttpServlet** que implementa la interface **Servlet** y una serie de clases e interfaces específicas para atender requerimientos HTTP.
  - La clase **jakarta.servlet.http.HttpServlet** es la más usada para implementar servlets HTTP.

# La interface Servlet

Directa ó indirectamente todos los servlets HTTP implementan la interface **jakarta.servlet.Servlet**

**public interface Servlet**

- public void init (ServletConfig config) throws ServletException
- public void service(ServletRequest req, ServletResponse res) throws ServletException, UnAvailableException
- public void destroy()
- public ServletConfig getServletConfig()
- public String getServletInfo()

getInitParameter(clave): permite obtener parámetros de inicialización del servlet
getServletContext(): permite obtener el contexto de la aplicación (objeto ServletContext)

Representa el requerimiento del cliente

Representa la respuesta que se le enviará al cliente

Generalmente para implementar servlets se extienden las clases jakarta.servlet.GenericServlet o jakarta.servlet.http.HttpServlet

# Las clases GenericServlet y HttpServlet

**public abstract class GenericServlet implements Servlet, ServletConfig, Serializable**

La clase **GenericServlet** provee una implementación básica de la interface **jakarta.servlet.Servlet**.

Está contenida en el paquete **jakarta.servlet**. Es una **clase abstracta**. Esto se debe a que el método **service() es abstracto**, por lo tanto cualquier clase que extienda esta clase debe implementar el método service().

**public abstract class HttpServlet extends GenericServlet**

La clase **HttpServlet** provee una implementación específica para HTTP de la interface **jakarta.servlet.Servlet**. Agrega métodos adicionales que son invocados automáticamente por el método service() para ayudar al procesamiento de requerimientos HTTP.

Es la clase que implementan la mayoría de los servlets. Está contenida en el paquete **jakarta.servlet.http**.

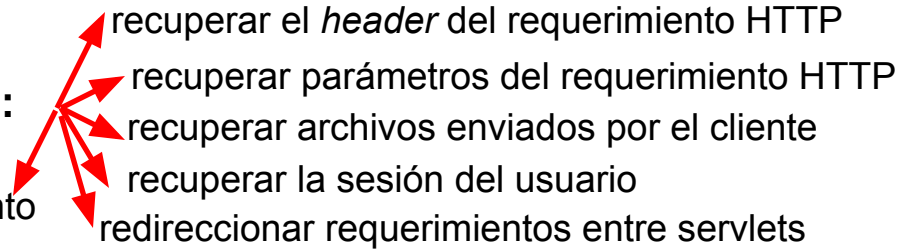
- **protected void service(HttpServletRequest req, HttpServletResponse res)**
  **throws ServletException, IOException**
- **public void service(ServletRequest req, ServletResponse res)**
  **throws ServletException, IOException**

# La interface HttpServletRequest

**public interface HttpServletRequest extends jakarta.servlet.ServletRequest**

El requerimiento HTTP de un cliente está representado por un objeto **HttpServletRequest**.

**El objeto HttpServletRequest se usa para:**




**Métodos para recuperar parámetros del requerimiento del cliente:**

**public String getParameter(String key)**

Devuelve el valor del parámetro del requerimiento con la clave dada. Si hay múltiples valores para ese parámetro, devuelve el primero.

**public String[] getParameterValues(String key)**

Devuelve todos los valores del parámetro del requerimiento con la clave dada, en el caso de un CHECKBOX o Combo Box devuelve los valores de todos los ítems seleccionados.

**public Enumeration getParameterNames()**

Devuelve una lista con los nombres de todos los parámetros del requerimiento.

# La Interface HttpServletResponse

**public interface HttpServletResponse extends jakarta.servlet.ServletResponse**

El objeto HttpServletResponse representa la respuesta que se le enviará al cliente. Por defecto produce una respuesta HTTP vacía.

Para generar una respuesta *customizada* es necesario usar los métodos ***getWriter()*** o ***getOutPutStream()*** que permiten obtener un *stream* de salida donde se escribe el contenido.

**public void setContentType(String type)**

Antes de devolver la respuesta se debe invocar a este método para setear el tipo MIME de la respuesta HTTP. Ej: text/html, image/JPG

**public PrintWriter getWriter() throws IOException**

El objeto PrintWriter que devuelve es usado por el servlet para escribir la respuesta como texto.

**public ServletOutPutStream getOutPutStream() throws IOException**

El objeto ServletOutPutStream (subclase de java.io.OutPutStream) es usado para enviar al cliente datos binarios (imágenes).

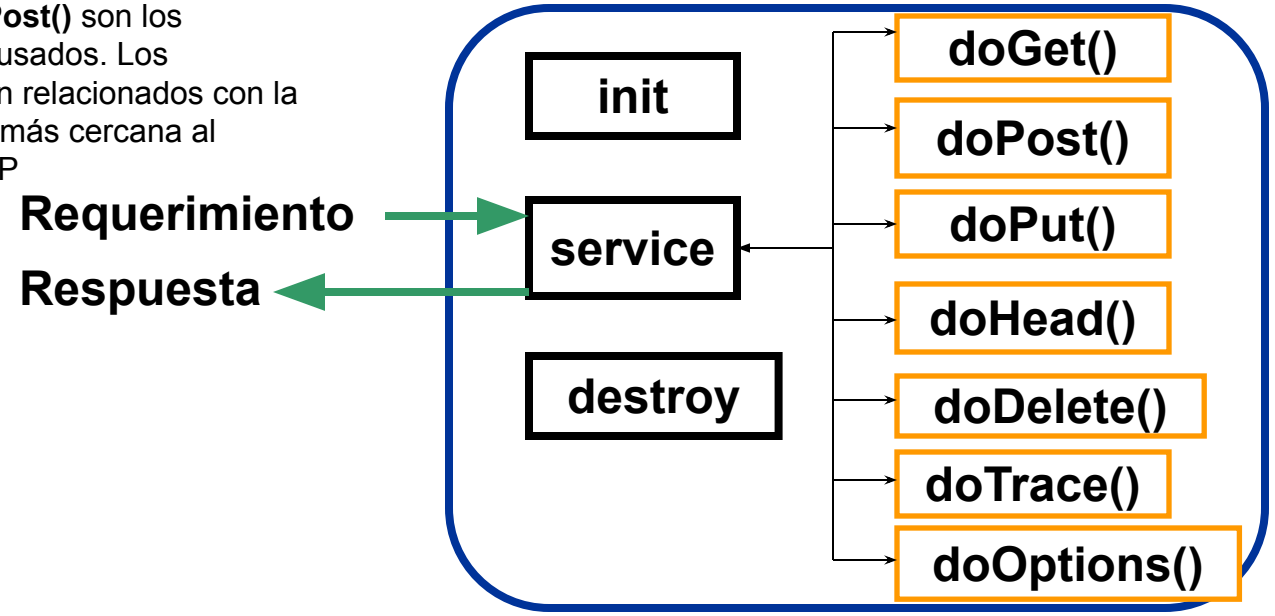
**public void setHeader(String name, String value)**

# La implementación del método service() de HttpServlet

El método **service()** del **objeto HttpServlet** invoca a uno de los siguientes 7 métodos “helpers”:

- doGet()
- doPost()
- doPut()
- doHead()
- doOptions()
- doDelete()
- doTrace()

**doGet()** y **doPost()** son los métodos más usados. Los restantes están relacionados con la programación más cercana al protocolo HTTP




Método HTTP con que el cliente hace el pedido

El método **service()** determina el método “helper” a invocar analizando la línea de inicio del requerimiento HTTP. Ejemplo: **GET /servletValida HTTP/3**

# Servlet “Hola Mundo”

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;

import java.io.*;

public class HolaMundo extends HttpServlet
{
      public void doGet (HttpServletRequest req, HttpServletResponse res) throws
                              ServletException, IOException
      {
            res.setContentType(“text/html”);
            PrintWriter out=res.getWriter();
            out.println(“<HTML>”);
            out.println(“<HEAD>”);
            out.println(“<TITLE> Hola Mundo </TITLE>”);
            out.println(“</HEAD>”);
            out.println(“<BODY>”);
            out.println(“<CENTER> <H1> Hola Mundo! </H1> </CENTER>”);
            out.println(“</BODY>”);
            out.println(“</HTML>”);

            out.close();
      }
}
```

Paquetes que incluyen clases e interfaces Java que proveen el soporte para servlets

**HttpServlet** es una clase que implementa la interface **jakarta.servlet.Servlet**

Se abre un **OutputSream** que contendrá la respuesta para el cliente web (browser)

Codificación de la salida

Envío de la respuesta al browser

# Servlet “Hola Mundo”

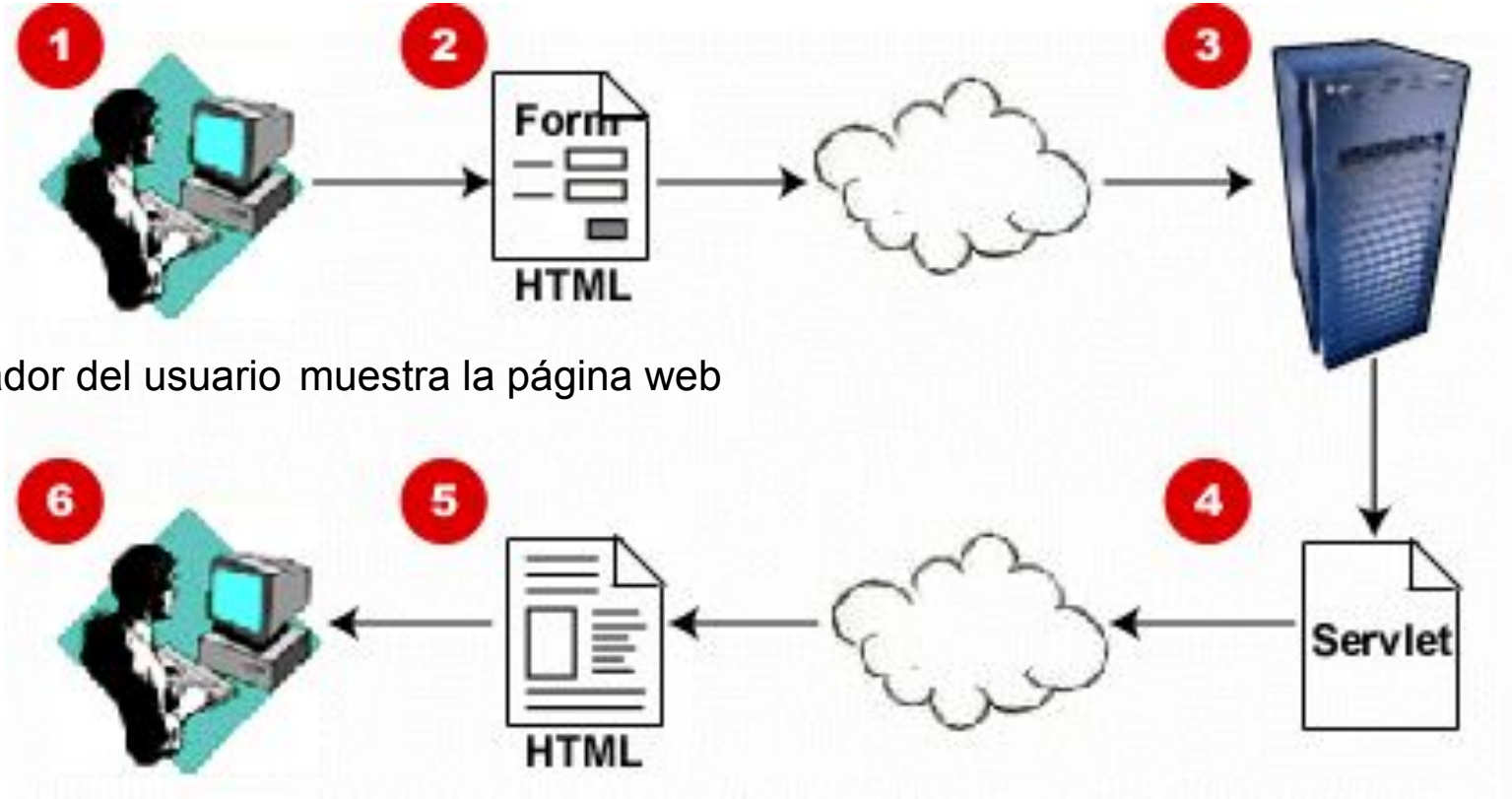
Página HTML obtenida por el cliente web como resultado de la ejecución del servlet:

| Página HTML de la respuesta | Código del servlet HolaMundo |
|---|---|
| <HTML><br><HEAD><br><TITLE> Hola Mundo </TITLE><br></HEAD><br><BODY><br><CENTER> <H1> Hola Mundo! </H1> </CENTER><br></BODY><br></HTML> | out.println(“<HTML>”);<br>out.println(“<HEAD>”);<br>out.println(“<TITLE> Hola Mundo </TITLE>”);<br>out.println(“</HEAD>”);<br>out.println(“<BODY>”);<br>out.println(“<CENTER> <H1> Hola Mundo! </H1> </CENTER>”);<br>out.println(“</BODY>”);<br>out.println(“</HTML>”); |

# Ejemplo

El usuario solicita información completando un formulario HTML que contiene una referencia a un servlet y presiona el botón “submit”

El servidor Jakarta EE localiza al servlet requerido



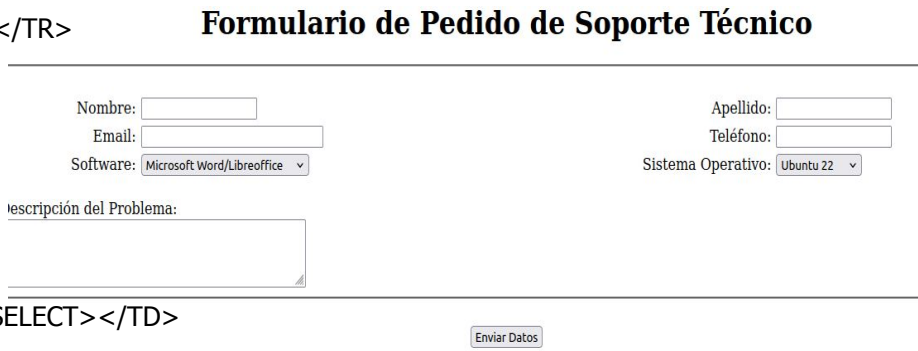

El navegador del usuario muestra la página web

El servlet con la información recibida procesa el pedido del usuario y construye una página web de respuesta con información específica, que se la envía al usuario

# Ejemplo: petición al servlet /soportetecnico

```
<!DOCTYPE html>
<HTML lang="es">
<HEAD>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Departamento de Informática</title>
</HEAD>
<BODY>
        <FORM ACTION="http://localhost:8080/compras/soportetecnico" METHOD="POST">
<CENTER>
<H1>Formulario de Pedido de Soporte Técnico </H1>
<HR>
<BR>
<TABLE ALIGN="center" WIDTH="100%" CELLSPACING="2" CELLPADDING="2">
<TR>   <TD ALIGN="right">Nombre: </TD>
        <TD><INPUT TYPE="Text"  NAME="txtNombre" ALIGN="LEFT" SIZE="15"></TD>
        <TD ALIGN="right">Apellido: </TD>
        <TD><INPUT TYPE="Text"  NAME="txtApellido" ALIGN="LEFT" SIZE="15"></TD> </TR>
<TR><TD ALIGN="right">Email: </TD>
        <TD><INPUT TYPE="Text"  NAME="txtMail" ALIGN="LEFT" SIZE="25"></TD>
        <TD ALIGN="right">Teléfono: </TD>
        <TD><INPUT TYPE="Text"  NAME="txtTel" ALIGN="LEFT" SIZE="15"></TD></TR>
<TR><TD ALIGN="right">Software: </TD>
        <TD><SELECT NAME="soft" SIZE="1">
                        <OPTION VALUE="Word">Microsoft Word/Libreoffice</OPTION>
                        <OPTION VALUE="Excel">Microsoft Excel/Libreoffice </OPTION>
                        <OPTION VALUE="Access">Microsoft Access/Libreoffice </OPTION></SELECT></TD>
        <TD ALIGN="right">Sistema Operativo: </TD>
        <TD><SELECT NAME="os"  SIZE="1">
                        <OPTION VALUE="95">Ubuntu 22 </OPTION>
                        <OPTION VALUE="98"> Windows 10 </OPTION>
                        <OPTION VALUE="NT"> Windows 11 </OPTION></SELECT></TD></TR>
</TABLE>
</CENTER>
<BR>Descripción del Problema:
<BR><TEXTAREA NAME="txtProblema"  COLS="50" ROWS="4"></TEXTAREA>
<HR>
<BR>
<CENTER>
        <INPUT TYPE="Submit" NAME="enviar"  VALUE="Enviar Datos">
</CENTER></FORM></BODY></HTML>
```

Servlet



# Ejemplo: servlet /soportetecnico

```
package misservlets;

import jakarta.servlet.*;
import jakarta.servlet.http.*;

import java.io.*;
import java.util.*;
import sun.net.smtp.SmtpClient;

public class SoporteTecnico extends HttpServlet
{
private String message, msgFrom, msgTo, msgSubject;

public void doPost(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException
{
        res. setContentType("text/html");
        PrintWriter out=res.getWriter();

        getParameters(req);

        if (! sendMail())
        {
          res.sendError(res.SC_INTERNAL_SERVER_ERROR,
           "Error de acceso al servidor de correo");
          return;
         }

        out.close();

 }
```

Se abre un OutputSream para armar la respuesta al browser

Invoca sucesivamente al getParameter() sobre el objeto req y recupera todos los datos del formulario en la variable **message**

Reporta un error al browser

Envía la respuesta al browser

```
public void getParameters (HttpServletRequest req) throws
                ServletException, IOException
{       StringBuffer tempStringBuffer =new StringBuffer(1024);
        msgSubject=" Requerimiento de soporte técnico";
        msgTo="claudiaq@info.unlp.edu.ar";
        msgFrom=req.getParameter("txtMail");
        tempStringBuffer.append("From:");
        tempStringBuffer.append(req.getParameter("txtNombre"));
        tempStringBuffer.append(" ");
        tempStringBuffer.append(req.getParameter("txtApellido"));
        tempStringBuffer.append("\n");
        message=tempStringBuffer.toString();

}
public boolean sendMail ()
{       PrintStream salida;
        SmtClient send;
        try {
        send=new SmtClient("info.unlp.edu.ar");
        send.from(msgFrom);
        send.to(msgTo);
        salida =send.startMessage();
        salida.println("From: "+msgFrom);
        salida.println(To: "+msgTo);
        salida.println("Subject: "+msgSubject);
        salida.println("\n----------------------\n");
        salida.println(message);
        ...................
        salida.flush();
        salida.close();
        send.closeServer();
        } catch (IOException e) {
          log ("Error al enviar el mail", e);
          return false;
        }
        return true;
}
}
```

# El módulo web

Los **recursos web** están formados por las **componentes web** y **archivos estáticos** (imágenes, archivos html, páginas de estilo) que pueden referenciarse por una url.

Un **módulo web** es la **unidad desplegable de recursos web**. A su vez puede contener otros archivos como clases utilitarias server-side y client-side.

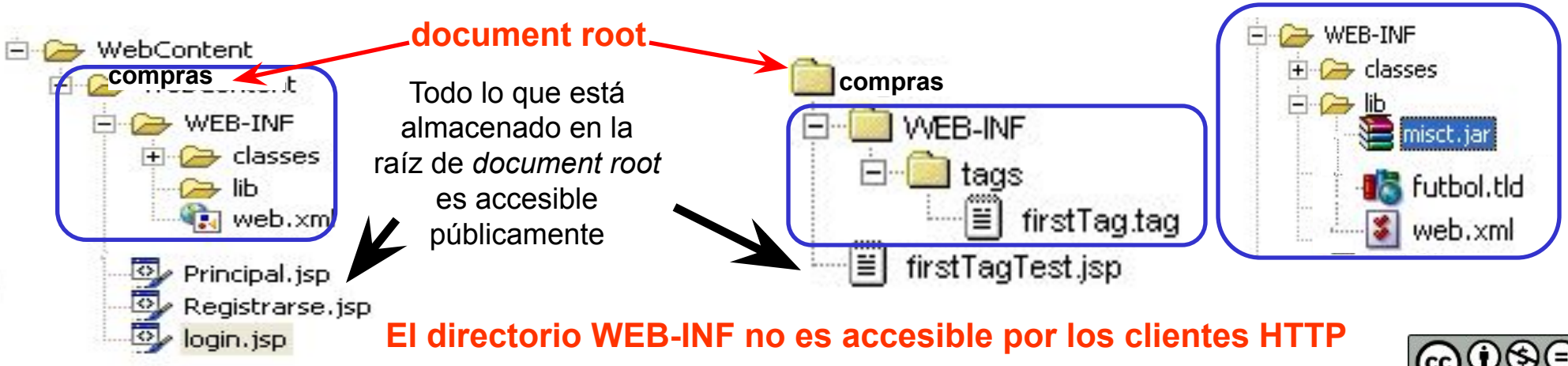
Para la especificación de servlets **un módulo web** se corresponde con **una aplicación web**.

Un **módulo web** tiene una estructura de directorios específica:

*document root*: es el directorio de más alto nivel, donde se almacenan las **JSPs**, **clases** y **archivos client-side** y **recursos estáticos (imágenes, archivos html)**

El *document root* tiene un subdirectorio llamado **/WEB-INF/** que contiene archivos y directorios:

- **web.xml**: es el archivo descriptor de la aplicación web
- **TLDs:** archivos descriptores de librerías de tags, son archivos XML
- **classes**: es un directorio que contiene clases server-side como los servlets, filtros, listeners, clases utilitaris y javabeans
- **lib**: es un directorio que contiene archivos .JAR de librerías usadas por las clases server-side.
- **tags**: es un directorio que contiene archivos que implementan librerías de tags




**El directorio WEB-INF no es accesible por los clientes HTTP**

# El archivo web.xml

- El archivo descriptor de la aplicación web, **web.xml**, define TODO lo que el contenedor web necesita conocer sobre la aplicación web.
- Es estándar y se ubica en la carpeta **/WEB-INF/web.xml.**
- Los IDEs proveen editores visuales y ayudas durante el desarrollo de la aplicación web que permiten crear, actualizar y editar en forma simple y consistente el **web.xml**.
- La especificación de Servlets incluye un Document Type Descriptor (DTD) para el **web.xml** que define su gramática. Por ej. los elementos descriptores <filter>, <servlet> y <servlet-mapping> deben ser ingresados en el orden establecido por el DTD. En general los contenedores fuerzan estas reglas cuando procesan el **web.xml.**
- Al ser declarativa la información contenida en el archivo **web.xml** es posible modificarla sin necesidad de modificar el código fuente de las componentes.
- En ejecución, el contenedor web lee el archivo **web.xml** y actúa en consecuencia.

```
<web-app>
    <display-name>compras</display-name>
     <context-param>
       <param-name>jdbcDriver</param-name>
       <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
     </context-param>
     <context-param>
       <param-name>jdbcURL</param-name>
       <param-value>jdbc:odbc:usuariosDB</param-value>
     </context-param>
     <filter>
       <filter-name>holafiltro</filter-name>
       <filter-class>misfiltros.HolaFiltro</listener-class>
     </filter>
    <filter-mapping>
       <filter-name>holafiltro</filter-name>
       <url-pattern>/HolaFiltro </url-pattern>
    </filter-mapping>
     <listener>
       <listener-class>practica8.InicializaConDB</listener-class>
     </listener>
     <servlet>
       <servlet-name>chequearlogin</servlet-name>
       <servlet-class>practica8.ChequearLogin</servlet-class>
     </servlet>
     <servlet>
       <servlet-name>chango</servlet-name>
       <servlet-class>practica8.Chango</servlet-class>
     </servlet>
     <servlet-mapping>
       <servlet-name>chequearlogin</servlet-name>
       <url-pattern>/chequearlogin</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
       <servlet-name>chango</servlet-name>
       <url-pattern>/chango</url-pattern>
      </servlet-mapping>
  </web-app>
```

**Descripción general de la aplicación web**

**Parámetros de inicialización de la aplicación (GLOBALES)**

**Servlets Filtros**

**Servlets Listeners**

***Declaración de los Servlets de la aplicación y de sus URLs***

**A partir JEE 5, se incorporan ANOTACIONES para añadir metainformación al código fuente que será usada en ejecución por el contenedor web. Se reduce el web.xml**

**web.xml**

# Servlets en Acción

- El archivo **.class del servlet** se ubica en la carpeta **/WEB-INF/classes** de la aplicación web junto con otras clases JAVA.

- Para que un cliente pueda acceder a un servlet debe declararse una URL o un conjunto de URL's asociadas al servlet.

- A partir JEE 5 se incorporan **ANOTACIONES** para añadir metainformación al código fuente que será usada en ejecución por el contenedor web.

- Antes de JEE 5 se usaban descriptores en el **web.xml** para declarar servlets y mapeos de servlets.

- Las **ANOTACIONES** reemplazan en parte a los descriptores de despliegue en el web.xml

# Servlets en Acción
## Descriptores en el web.xml

- Para que un cliente pueda acceder a un servlet debe declararse una URL o un conjunto de URL's asociadas al servlet en el archivo **web.xml**.
- El archivo **web.xml** usa los elementos **<servlet>** y **<servlet-mapping>** para declarar los servlets que serán cargados por el contenedor web y para mapearlos a una URL o conjunto de URL's respectivamente.
- En el archivo **web.xml** pueden aparecer múltiples tags **<servlet> y <servlet-mapping>**, los que permiten definir los servlets y los mapeos necesarios.

```
<servlet>
        <servlet-name>soportetecnico </servlet-name>
        <servlet-class>misservlets.SoporteTecnico </servlet-class>
</servlet>
```

Se declara un servlet, asignándole un **nombre único** y una **clase Java** que lo implementa

Se mapea un servlet con una URL

```
<servlet-mapping>
        <servlet-name>soportetecnico </servlet-name>
        <url-pattern>/soportetecnico </url-pattern>
</servlet-mapping>
```

URL de la aplicación web

**URL completa del servlet:** **http:// localhost:8080/compras** **/soportetecnico**

# Configurar Servlets en el web.xml

- Es posible proveer a los servlets de información de configuración inicial que consiste de un string o conjunto de strings que se incluyen en el **web.xml**.
- Para definir parámetros de configuración inicial se usan los sub-elementos **<init-param>**, **<param-name>** y **<param-value>** en el **web.xml**.
- Este tipo de configuración permite especificar parámetros iniciales para un servlet fuera del código compilado y cambiarlos sin necesidad de recompilar el servlet.
- Cada servlet tiene asociado un objeto **ServletConfig** creado por el contenedor (implementa la interface jakarta.servlet.ServletConfig) que contiene los parámetros de inicialización.

```
<servlet>
   <servlet-name>holamundo</servlet-name>
   <servlet-class>misservlets.HolaMundo</servlet-class>
    <init-param>
    <param-name>saludo</param-name>
    <param-value>Hola!!!</param-value>
    </init-param>
</servlet>
```

Los parámetros de configuración de un servlet se especifican dentro del elemento <servlet>

La configuración de servlets mediante parámetros de inicialización strings es un método simple, efectivo y limitado.

Para servlets complejos, es posible crear un archivo xml de configuración que acompañe al web.xml.

```
public void init(SevletConfig arg0){
arg0.getInitParameter("saludo");
}
```

```
public void doGet(HttpServletRequest req, HttpServletResponse res ) throws...{
  this.getServletConfig().getInitParameter("saludo");
}
```

# Anotar servlets

```
package misservlets;
@WebServlet("/soportetecnico")
public class SoporteTecnico extends HttpServlet {
    // código del servlet
}
```

El servlet **SoporteTecnico** es mapeado a la url ***/soportetecnico***

```
@WebServlet(
    name = “soporte",
    description = “Un ejemplo de servlet anotado",
    urlPatterns = {"/soportetecnico"}
)
public class SoporteTecnico extends HttpServlet {
    // código del servlet
}
```

Aquí se agrega información a la declaración del servlet

```
@WebServlet (urlPatterns =
{"/sendFile", "/uploadFile"})
public class UploadServlet extends HttpServlet {
    // código del servlet
}
```

El servlet **UploadServlet** puede ser accedido por 2 URLs: **/sendFile** y **/uploadFile**.

# Anotar servlets

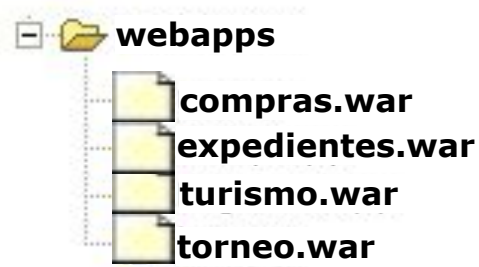
```
@WebServlet(
        urlPatterns = "/imageUpload",
        initParams =
        {
            @WebInitParam(name = "saveDir", value = "D:/FileUpload"),
            @WebInitParam(name = "allowedTypes", value = "jpg,jpeg,gif,png")
        }
)
public class ImageUploadServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException {
        String saveDir = getInitParameter("saveDir");
        String fileTypes = getInitParameter("allowedTypes");

        PrintWriter writer = response.getWriter();

        writer.println("saveDir = " + saveDir);
        writer.println("fileTypes = " + fileTypes);
    }
}
```

El servlet **ImageUploadServlet** es mapeado a la URL ***/imageUpload*** y además especifica 2 parámetros de inicialización: ***saveDir*** y ***allowedTypes***.

El método **doGet()** recupera los valores de esos parámetros y los imprime en la pantalla del browser.

# El archivo .war

- Las aplicaciones web JAVA se empaquetan en un archivo ***Web ARchive* (WAR)**. El archivo WAR es ideal para distribuir e instalar una aplicación. El formato "desempaquetado" es útil en la etapa de desarrollo.
- Un WAR tiene una estructura de directorios específica. El directorio raíz de un WAR es el *document root* de la aplicación web.
- El archivo WAR es un archivo JAR que contiene un módulo web: páginas HTML, archivos de imágenes, JSPs, clases y archivos client side, páginas de estilo, código JavaScript, el directorio WEB-INF y sus subdirectorios (classes, lib, tag, etc), el archivo web.xml.
- Los archivos WAR están definidos oficialmente en la especificación de Servlets a partir de la versión 2.2. Es un **estándar**. Todos los contenedores que implementan la especificación de la API de Servlets 2.2 y superiores deben soportar archivos WAR.
- La compresión usada en los archivos WAR es ZIP, de la misma manera que los archivos JAR.
- Se puede crear el archivo WAR mediante la línea de comando con el comando *jar* del JSDK. Los IDEs ofrecen funcionalidades para construir el WAR en forma automática.
- En el servidor Tomcat el archivo WAR de la aplicación web se debe copiar en el directorio *webapps:*




**Cuando Tomcat arranca, automáticamente expande a partir de webapps el contenido de cada uno de los archivos .war en su forma "desempaquetada". Si usamos esta técnica para hacer el "deployment" de nuestra aplicación y necesitamos actualizarla, debemos reemplazar el .WAR y ELIMINAR la estructura de directorios expandida y luego re-iniciar Tomcat.**