

Taller de Tecnologías de Producción de Software

Cursada 2025

Docentes

Laura Fava, Jorge Rosso, Lourdes Valli, Sebastián Villena,
Álvaro Aberastain Marzorati

CLASE 1 - Contenido

- Aplicaciones web. Definiciones. Arquitectura.
- Tecnologías Java
- Arquitectura de una aplicación
 - Capas lógicas (layers)
 - Capas físicas (tiers)
- Breve síntesis de HTTP y HTML
- Servidores J2EE
- Introducción a Servlets
- Parámetros de inicialización de Servlets
- Configuración mediante XML y mediante anotaciones

Aplicaciones web

Qué son y cómo trabajan?

Son programas interactivos contruidos con tecnologías web -HTML, CSS, JS- que manipulan datos que se encuentran almacenados en algún servidor. Las aplicaciones web pueden ser usadas simultáneamente por varios usuarios a través de internet.

Cualquier aplicación web típica, está compuesta por dos códigos/partes diferentes que se ejecutan en distintos lugares:

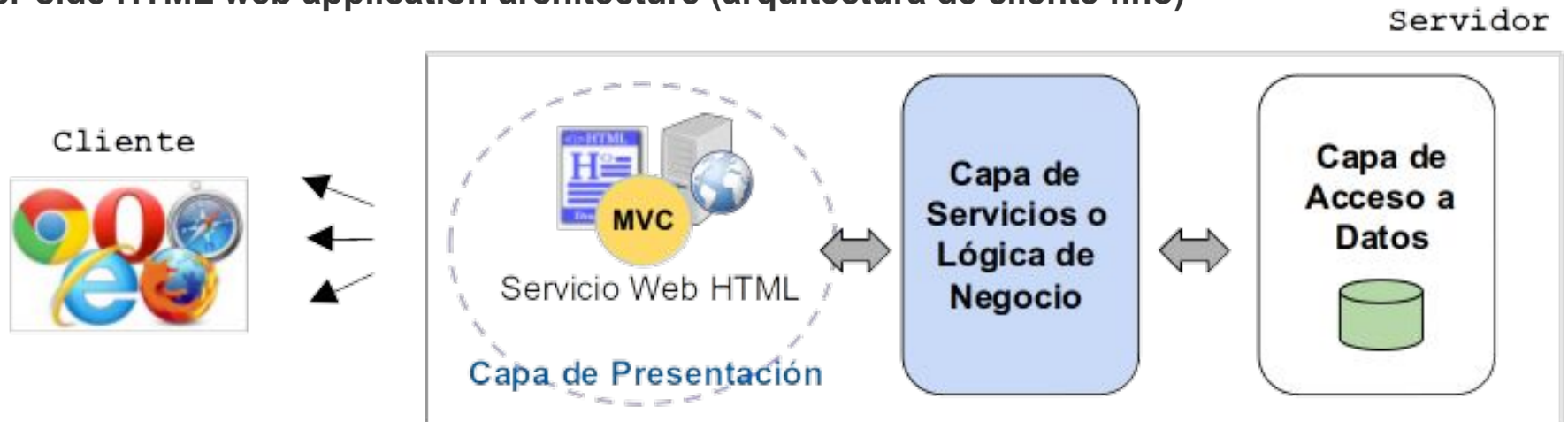
- **Frontend (client-side code):** Es el código que ejecuta en el navegador de la máquina del cliente y permite hacer peticiones y recibir/ver respuestas a esas peticiones.
- **Backend (server-side code):** Es el código que está en el o los servidores, los cuales procesan y responden a los requerimientos desde los clientes.



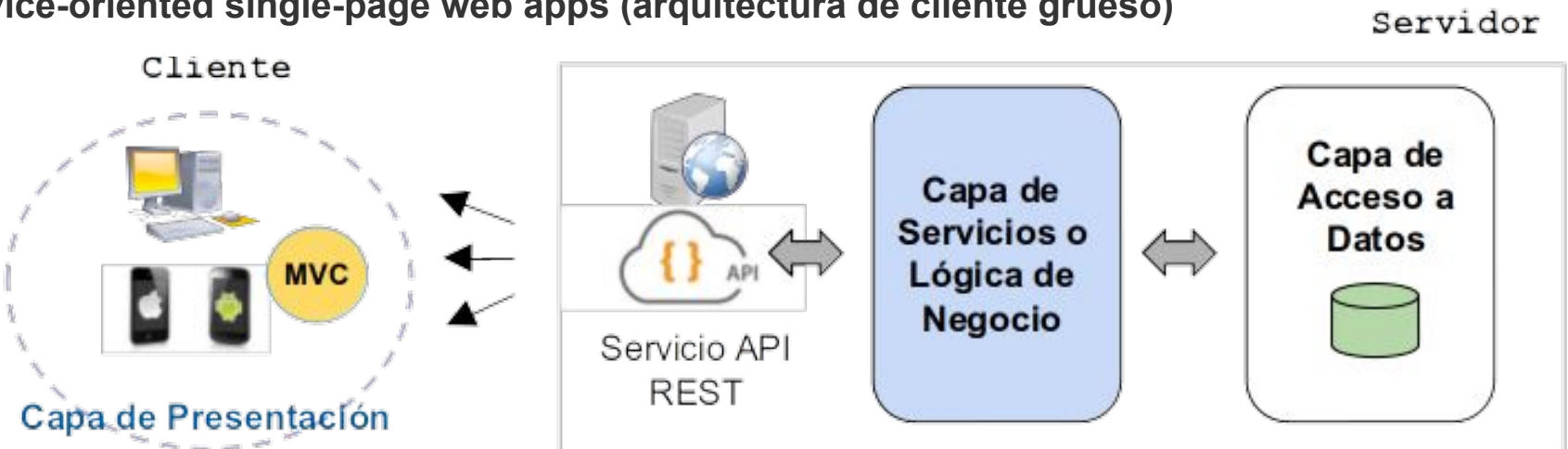
Aplicaciones web

Tipos de arquitecturas

Server-side HTML web application architecture (arquitectura de cliente fino)



Service-oriented single-page web apps (arquitectura de cliente grueso)



Arquitectura de una aplicación

Layers (capas lógicas) y Tiers (capas físicas)

Layers

Son formas de organizar el código. Una aplicación, típicamente incluye como mínimo las capas de Presentación, Negocio y Datos.

¿Por qué usar layers?

- Para lograr beneficios de una organización lógica y agrupar por funcionalidad => reusabilidad.
- Reduce los costos de mantenimiento de la aplicación.
- Mejor diseño.

Tiers

Se refiere a donde corre el código. Las *tiers* son los lugares donde ejecutan las *layers*. Cada recurso ejecutando en un proceso es considerado un *tier*.

¿Cuál es el beneficio de hacer el despliegue (deploy) y la ejecución de las capas lógicas en las capas físicas?

Para lograr escalabilidad, tolerancia a fallas y seguridad.

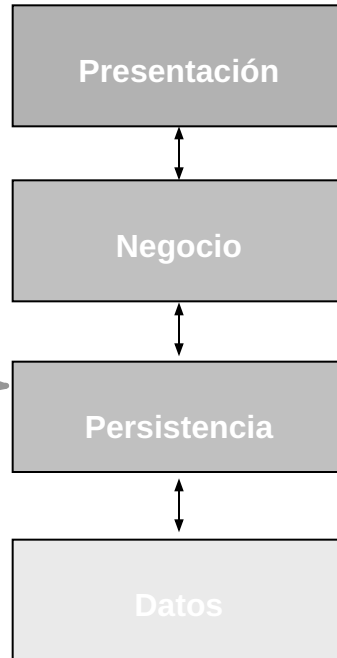
Arquitectura de una aplicación

Layers (capas lógicas)

La mayoría de las aplicaciones pueden ser organizadas para que soporten un conjunto de capas lógicas o layers. Una arquitectura de alto nivel para una aplicación empresarial, usa estas capas:

Arquitectura
Layered

Cada lógica (**layer**) provee un conjunto de servicios a la/s capa/s adyacentes.



Provee la lógica necesaria para la interfaz de usuario. Está destinada a formatear y desplegar información.

Implementa la lógica necesaria de la aplicación. En esta capa se implementan las reglas de negocio o requerimientos del sistema.

Esta capa está formada por un conjunto de clases y componentes responsables de almacenar los datos y recuperarlos desde una fuente de datos.

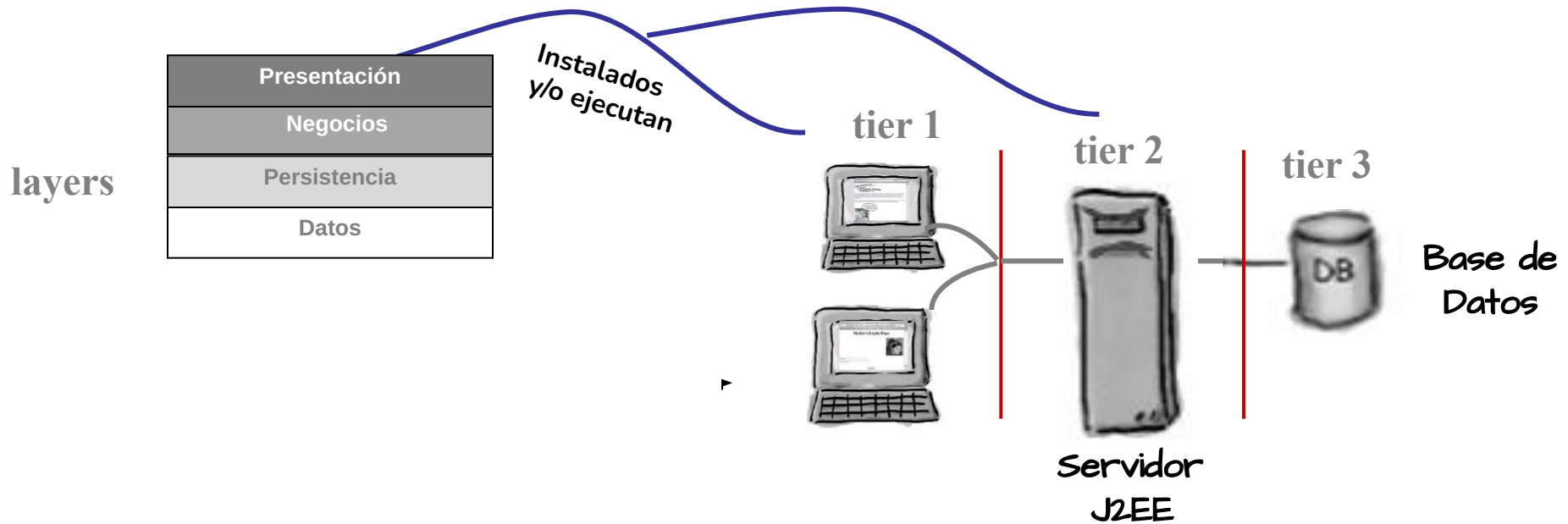
Representa los datos subyacentes. Es la representación persistente de los datos del sistema.

Con una arquitectura *layered*, los cambios en una de las capas no deberían afectar o afectar muy poco al resto de las capas.

Arquitectura de una aplicación

Tiers (capas físicas)

Una plataforma *empresarial* es por naturaleza, una plataforma distribuida. Las tareas están distribuidas en distintas computadoras. Cada computadora implementa/ejecuta uno o más **layers**.



Una aplicación JEE se “despliega” comúnmente en una máquina cliente, un servidor JEE y un servidor de datos.

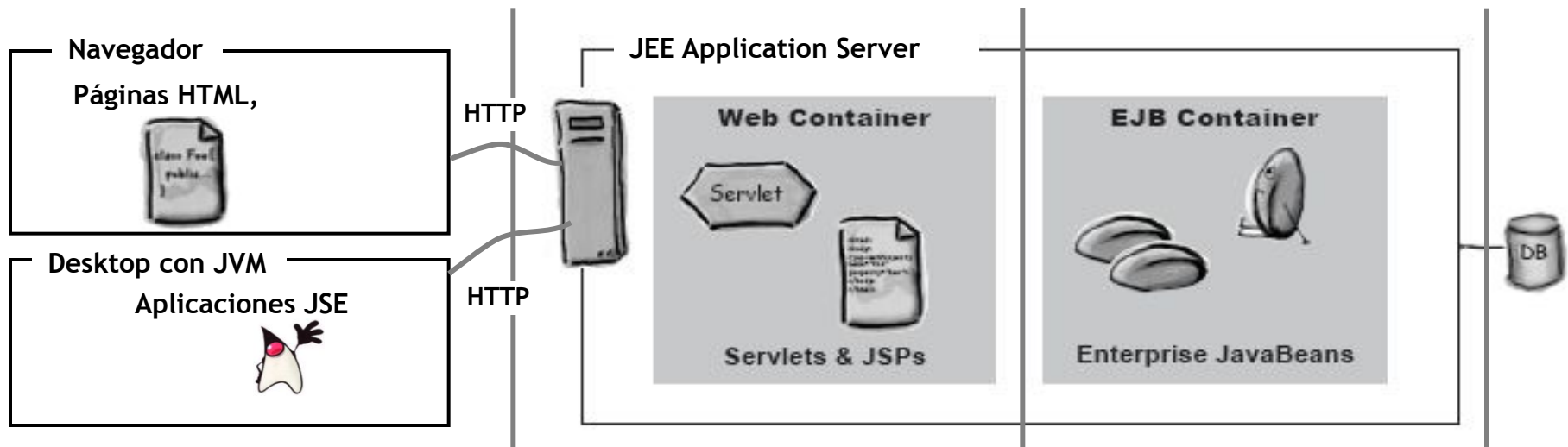
Arquitectura de JEE (Jakarta EE Platform)

Componentes

La plataforma JEE usa un modelo de aplicación “multitiered” distribuida. La lógica de la aplicación está dividida en componentes de acuerdo a su función, y las componentes que forman parte de la aplicación java EE son instaladas en varias máquinas dependiendo de la “tier” en el ambiente “multitiered” java EE a la que pertenece.

La especificación Java EE define las siguientes componentes:

- **Componentes Cliente:** Aplicaciones JSE, páginas HTML.
- **Componentes Web:** Servlets, Filtros, Llisteners , JSPs y JavaServer Faces
- **Componentes Empresariales:** Enterprise JavaBeans



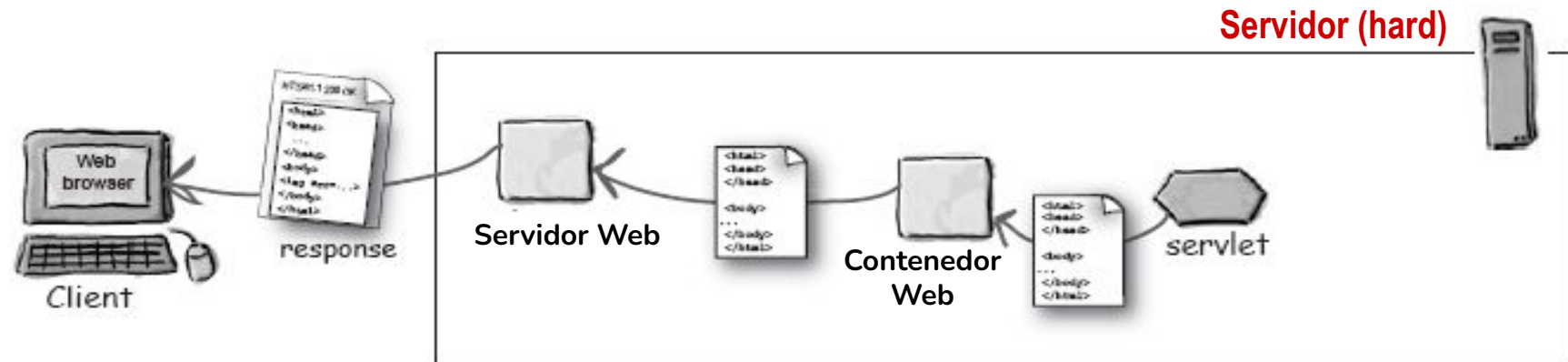
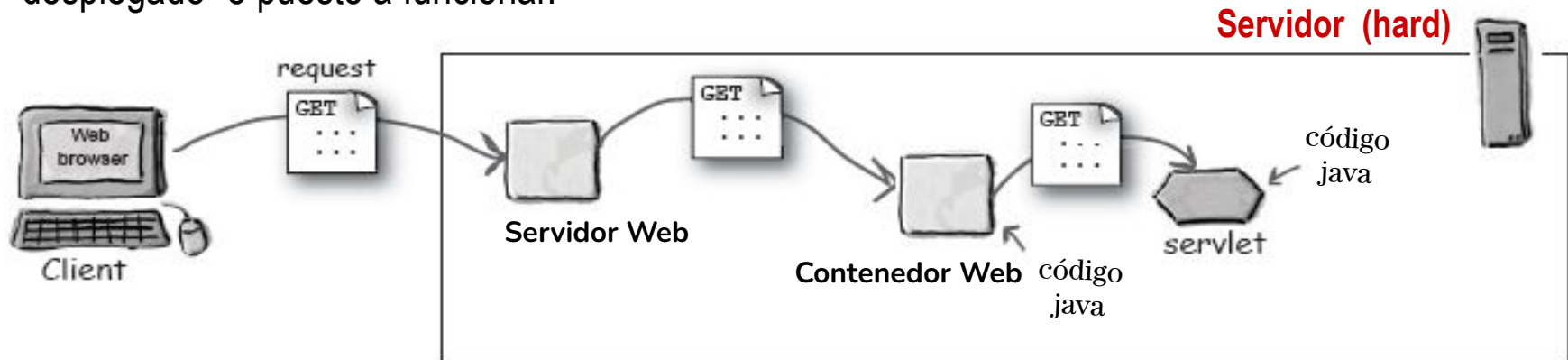
Una aplicación JEE está basada en componentes, donde una componente JEE es una unidad funcional de software, que debe correr dentro de un contenedor y que puede comunicarse con el resto de las componentes.

Arquitectura JEE

Contenedores

Las componentes web JEE no tienen un método `main()` como las aplicaciones de escritorio. Ellas están bajo el control de otra aplicación java, llamada Contenedor.



Cuando el servidor HTTP recibe un requerimiento para una componente web (supongamos un Servlet), el servidor no maneja el requerimiento solo, sino que lo hace con ayuda del contenedor web en donde el servlet fue “desplegado” o puesto a funcionar.



Arquitectura JEE

Contenedores y Servidores JEE

Servidores JEE certificados según la especificación Jakarta EE:

Contenedor JEE	Tomcat (Apache) Jetty (Eclipse Foundation)  Apache Tomcat
Servidores de aplicaciones JEE	TomEE (Apache) GlassFish (Eclipse Foundation) Red Hat JBoss EAP (Enterprise Application Platform) WebSphere Application Server (IBM)  Apache TomEE



Arquitectura JEE

Contenedores y Servidores JEE

Tomcat o TomEE?

Para las organizaciones que están considerando servidores web/aplicaciones de código abierto, Tomcat y TomEE suelen estar entre las primeras en la lista. Pero existen algunas diferencias clave entre Tomcat y TomEE que los equipos deben tener en cuenta antes de decidir.

La diferencia principal entre Tomcat y TomEE es que Tomcat soporta servlets y JSP, mientras que TomEE cuenta además con un conjunto más amplio de tecnologías basadas en JSR, como EJB y CDI.

Apache Tomcat tiene un tiempo de inicio y despliegue muy rápido. La versión actual de Apache Tomcat es la 10.0.x, la cual implementa las especificaciones Servlet 5.0 y JSP 3.0. La licencia de Apache Tomcat está gestionada por la Fundación Eclipse bajo las licencias Apache 2.0.

Apache TomEE aprovecha todas las características de Apache Tomcat (JNDI, seguridad, etc.), incluyendo Enterprise Java Beans (EJB), Contextos e Inyección de Dependencias (CDI), Java Persistence API (JPA), Java Transaction API (JTA), Java Server Faces (JSF), Java Message Service (JMS), Servicios Web y una implementación de Sistema de Gestión de Bases de Datos Relacionales (RDBMS) con Java Database Connectivity (JDBC). La licencia de TomEE también se gestiona bajo la licencia Apache 2.0.

En resumen:

- TomEE = ya viene con soporte para JPA y otras APIs Java EE → no es necesario añadir JPA.
- Tomcat = sólo Servlets y JSP → si se necesita JPA hay que agregarlo en el POM.

Cientes y servidores

conocen HTTP y HTML

¿Qué es el protocolo HTTP?

La mayoría de las conversaciones que mantienen los navegadores con los servidores lo hacen usando el protocolo HTTP -HyperText Transfer Protocol-, el cual permite conversaciones del tipo requerimiento - respuesta. El cliente envía un requerimiento HTTP y el servidor responde con una respuesta HTTP.

Los elementos clave de un **request**:

- El **método HTTP**
- El recurso a acceder (**URL**)
- Los **parámetros** del formulario



¿Qué es HTML?

Cuando un servidor responde a un requerimiento, en general envía al navegador un conjunto de instrucciones escritas en HTML -HyperText Markup Language-. El HTML le dice al navegador **cómo** mostrar el contenido al usuario.

Los elementos clave de un **response**:

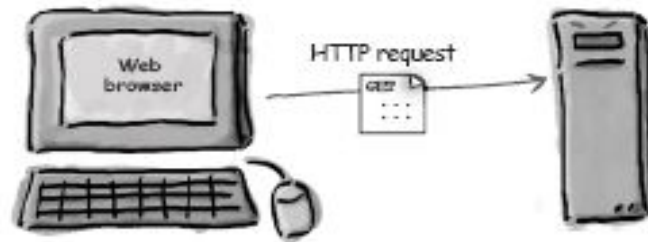
- El código de estado.
- El Content-type (text-picture-**HTML**, etc.)
- El contenido

En términos generales los códigos de estados 5xx son errores de servidor, los 4xx son errores de clientes, los 3xx son redirecciones y los 2xx ok

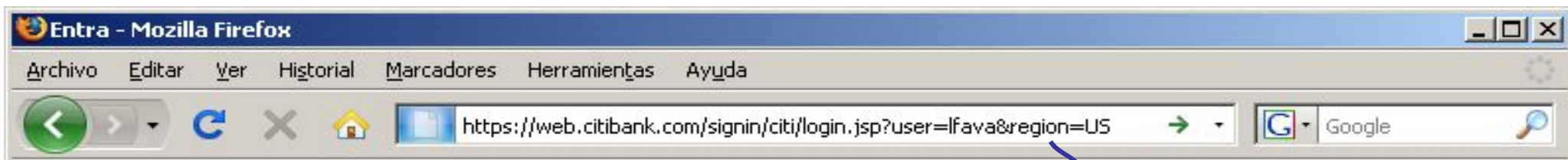
HTTP Request

¿Qué tiene un requerimiento HTTP?

Un requerimiento HTTP tiene un nombre de método (no son métodos java). HTTP tiene varios métodos pero básicamente usaremos GET/POST.



Los métodos más comunes para hacer una petición a un determinado recurso son GET/POST. Ambos pueden enviar datos en el pedido: con el GET los datos aparecen en la URL y tiene limitaciones –cada servidor soporta una cantidad limitada de caracteres-, con POST no hay límite porque los datos viajan en el cuerpo del HTTP.



Cuando se tipea la URL en la barra del navegador (al igual que al presionar un link) se genera un método GET

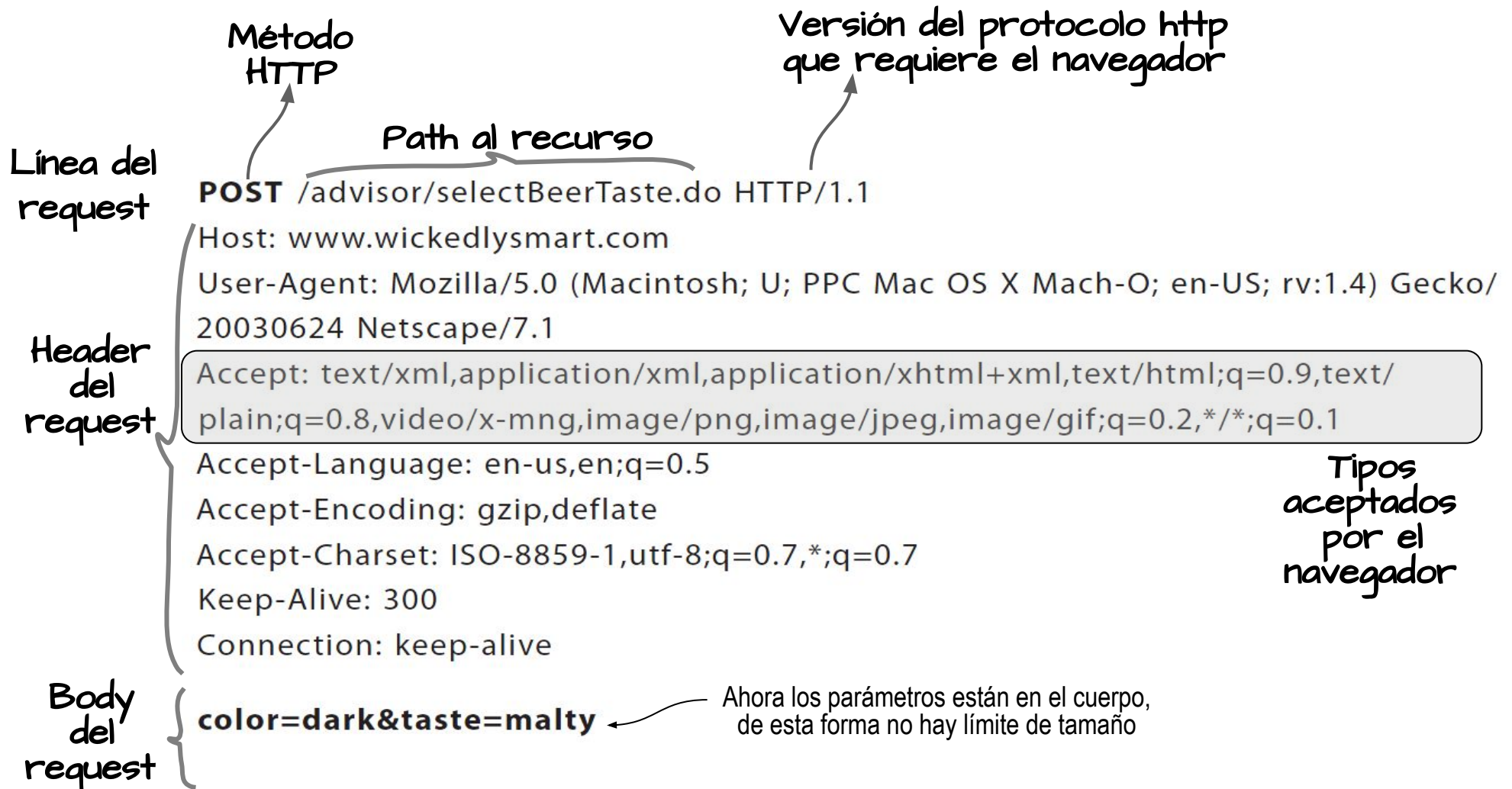
HTTP Request

Anatomía de un requerimiento HTTP GET



HTTP Request

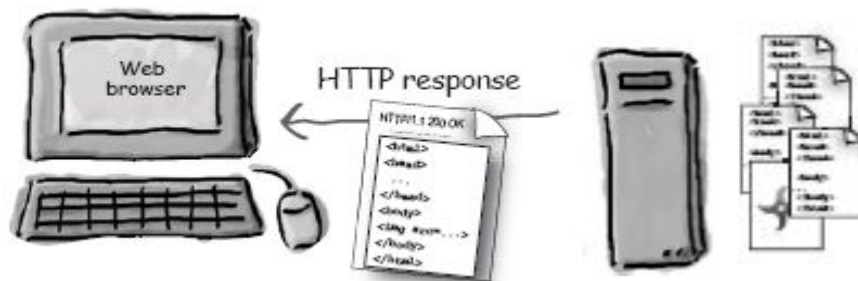
Anatomía de un requerimiento HTTP POST



HTTP Response

¿qué tiene una respuesta HTTP?

Una respuesta HTTP *puede* contener HTML. HTTP agrega un **header** con información arriba del contenido en la respuesta. Un navegador usa el **header** para procesar la respuesta y poder mostrarla.



HTTP header

HTTP header info

```
<HTML>
<HEAD>
<TITLE> Ejemplo de HTML </TITLE>
</HEAD>
<!-- cuerpo -->
<BODY>
```

Comentario HTML

HTTP body

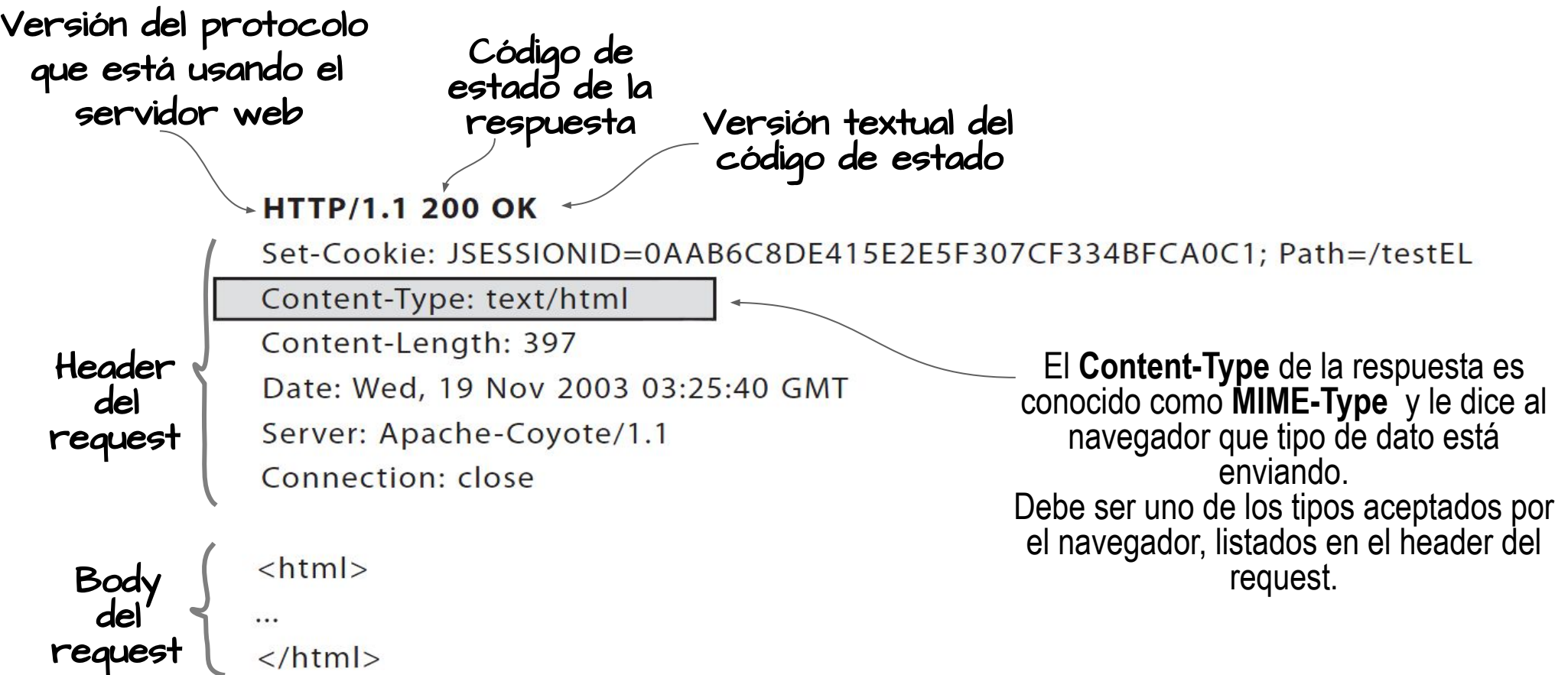
```
<P>
  Esto es un ejemplo de un documento HTML.
</P>
<FORM ACTION="chequearLogin" METHOD="POST">

  .
  .
  .
</FORM>
</BODY>
</HTML>
```

Estructura de un documento HTML.
Es parte de la respuesta HTTP

HTTP Response

Anatomía de un response HTTP

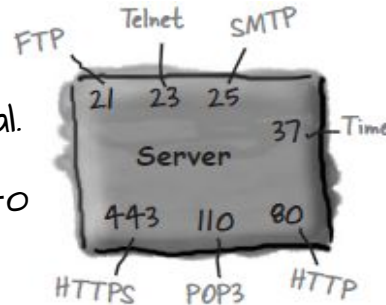


URL: Uniform Resource Locators

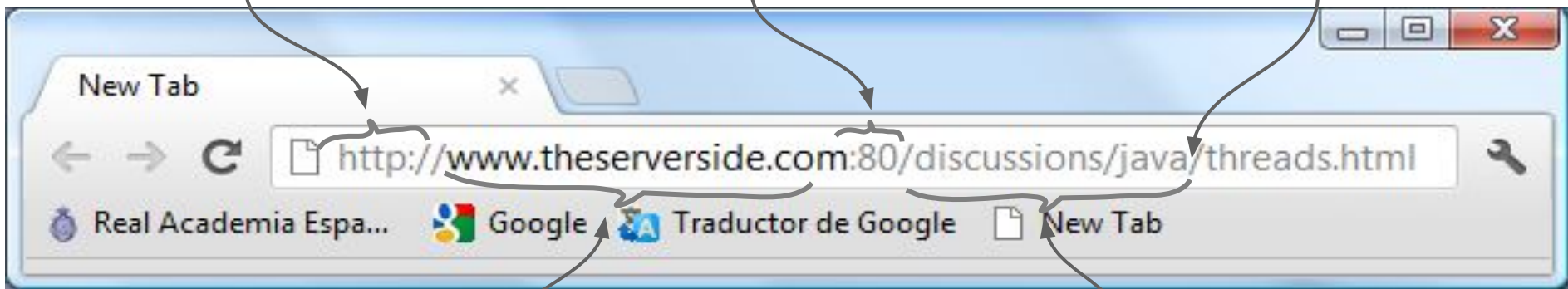
Cada recurso en la web tiene su propia y única dirección en el formato URL.

Protocolo : le indica al servidor que protocolo de comunicación será utilizado (http/https)

Port (puerto) : esta parte de la url es opcional. Cada servidor soporta muchos puertos. El puerto por defecto para servidores web, es 80.



Recurso : el nombre del recurso que se está pidiendo. Podría ser un HTML, una imagen, un servlet, etc. Si no se especifica nada, el servidor buscará un index.html



Servidor : nombre único del servidor al que se hace la petición. Este nombre mapea con una única dirección IP y se podría especificar la dirección IP en vez del nombre.

Path: camino de la ubicación del recurso buscado en el servidor.

Query String: si es un requerimiento GET la información extra (parámetros) son agregados al final de esta URL, comenzando con "?" y cada uno de los parámetros (nombre=valor) separado por "&".

HTML (HyperText Markup Language)

Evolución

HTML -“HyperText Markup Language”- especifica el formato de las páginas web, separando el contenido de las páginas, de su formato de presentación. Fue creado en los laboratorios CERN por Tim Berners-Lee.

Define un conjunto de símbolos (etiquetas o tags) que especifican la estructura lógica de un documento y de todos sus componentes.

Es independiente de la plataforma y su código es interpretado por los clientes web.

En el año 2004 comenzó a desarrollarse HTML5, actualmente el último estándar.

Los navegadores actuales soportan muchas de las características de HTML 5, los mejores posicionados para las versiones desktop son Google Chrome 36, Opera 22, Mozilla Firefox 30, para dispositivos móviles Tizen 2.2, BlackBerry 10.2, Chrome 35, etc.

La lista completa de navegadores y el soporte que brindan para HTML 5 puede consultarse en:
[**http://html5test.com/**](http://html5test.com/)

Una página web tiene una **estructura** dada por los tags HTML y una **visualización** determina por las hojas de estilo.

HTML (HyperText Markup Language)

Estructura de un documento HTML

HTML es un lenguaje que utiliza etiquetas (tags) y texto para marcar y definir la estructura de un documento HTML. Los tags consisten de un nombre encerrado entre <> para la apertura y </> para el cierre.

Por ejemplo:

```
<H1> un texto </H1>
```

A los tags con el contenido en el medio se los denomina elemento.
En este caso podemos decir elemento <H1>

Los tags pueden contener atributos. Los atributos permiten especificar información adicional a un elemento. Por ejemplo:

```
<a href="top10.html"> Great Movies </a>
```

nombre del
atributo

Valor del atributo. Siempre
entre comillas dobles

Hay algunos tags que excepcionalmente no tiene tag de cierre

```

```

HTML (HyperText Markup Language)

Estructura de un documento HTML/Formularios

HTML define diferentes componentes, tales como encabezados, títulos, cuerpo, **formulario**, tablas, hipervínculos, imágenes, etc. Un **formulario** es un conjunto de campos que permiten la interacción entre el usuario y el servidor HTTP. Se define con los tags <FORM> ... </FORM> y entre ellos se encuentran las definiciones de los campos del formulario, que viajarán como parámetros hacia el servidor.

Especifica la URI (Universal Resource Identifier) que atenderá el requerimiento

Especifica la forma en que se transferirán los datos, en general GET o POST

```
<html>
<body>
<FORM ACTION="chequearLogin" METHOD="POST">
  Nombre:<input type="text" name=nombre><br>
  Contraseña:<input type="text" name=contra><br>
    <input type="submit" name=b1 value="Enviar">
</FORM>
</body>
</html>
```

Cuando el navegador “envía” un formulario, crea un parámetro en el requerimiento para cada campo en el formulario (FORM)

Login.html

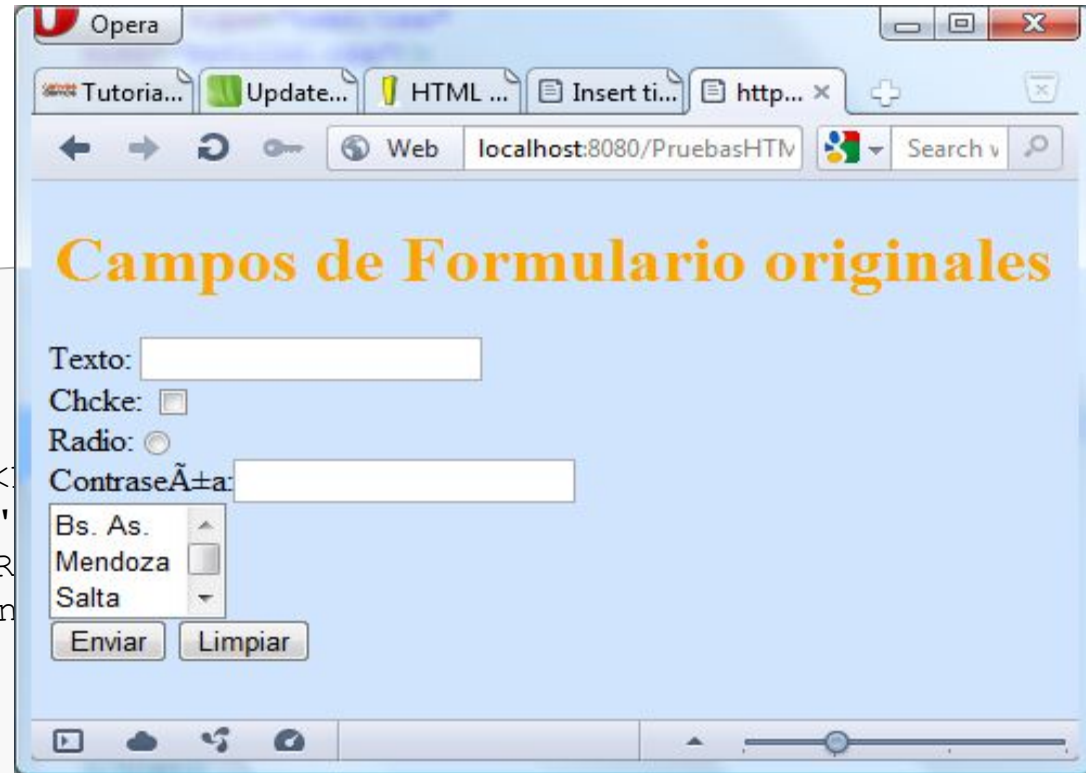
HTML (HyperText Markup Language)

Campos de formulario originales

Los campos "input" originales que están disponibles desde las primeras versiones del HTML son:

- **text** : campos de búsqueda
- **radio**: campos cuyo valor es una URL
- **checkbox** : direcciones de correo
- **password**: fechas
- **reset**: números
- **select**: un color hexadecimal

```
<!DOCTYPE html>
<html><body>
. . .
<h1>Campos de Formulario originales</h1>
Texto: <INPUT type="text" name="nombre">
Chcke: <INPUT type="checkbox" name="comp">
Radio:<INPUT type="radio" name="sexo"><BR>
Contraseña:<INPUT type="password" name="n
<SELECT size= "3" name="provSel">
  <OPTION value="BA"> Bs. As. </OPTION>
  <OPTION value="MZ"> Mendoza </OPTION>
  <OPTION value="CB"> Salta</OPTION>
  <OPTION value="MZ"> Misiones </OPTION>
</SELECT><BR>
<INPUT type="submit" value="Enviar">
<INPUT type="reset" value="Limpiar">
</body></html>
```



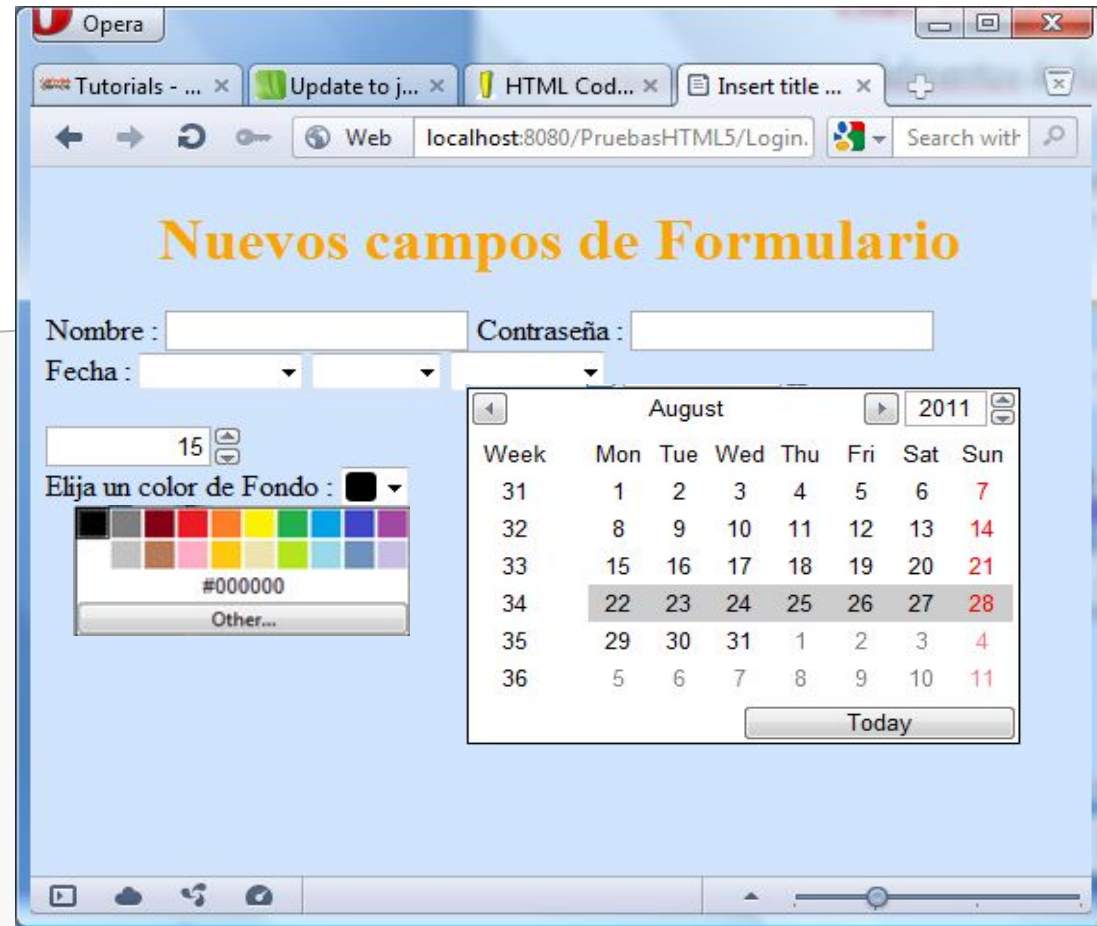
HTML (HyperText Markup Language)

Campos de Formulario del nuevo HTML 5

Algunos nuevos tipos de `<input>` son:

- **tel**: números telefónicos
- **search**: campos de búsqueda
- **url**: campos cuyo valor es una URL
- **email**: direcciones de correo
- **date**, **month**, **week**: fechas
- **number**: números
- **color**: colores en hexadecimal

```
<!DOCTYPE html>
<html><body>
. . .
<h1>Nuevos campos de Formulario</h1>
Fecha:
  <input type="date" name="dia"/>
  <input type="month" name="mes"/>
  <input type="week" name="semana"/>
<br>
  <input type="number" min="0" max="15"
        step="2" Value="15">
Elija un color de Fondo :
  <input type="color" name="colores"/>
</body>
</html>
```



Ejemplos en: http://www.w3schools.com/html/html_forms.asp

HTML (HyperText Markup Language)

Otras componentes de HTML5

También se incorporaron dos elementos para **contenido multimediales** :

- **<audio>**: Permite reproducir **audio** en una página web sin necesidad de plugins externos. Los formatos soportados comúnmente: Ogg (.ogg) → formato abierto y libre. MP3 (.mp3) → ampliamente soportado en navegadores y WAV (.wav) → sin compresión, mayor tamaño.
- **<video>**: Permite insertar el video en una página web. Los formatos soportados comúnmente: MP4 (.mp4) → muy usado, códec H.264. WebM (.webm) → formato abierto y Ogg/Theora (.ogv) → menos común pero soportado por navegadores.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
```

Ejemplo del Tag audio

```
<audio src="elegirCuento.wav" controls></audio>
```


Ejemplo del Tag video

```
<video src="html5.mp4" controls width="180"
height="260"></video>
```

```
</body>
```

```
</html>
```

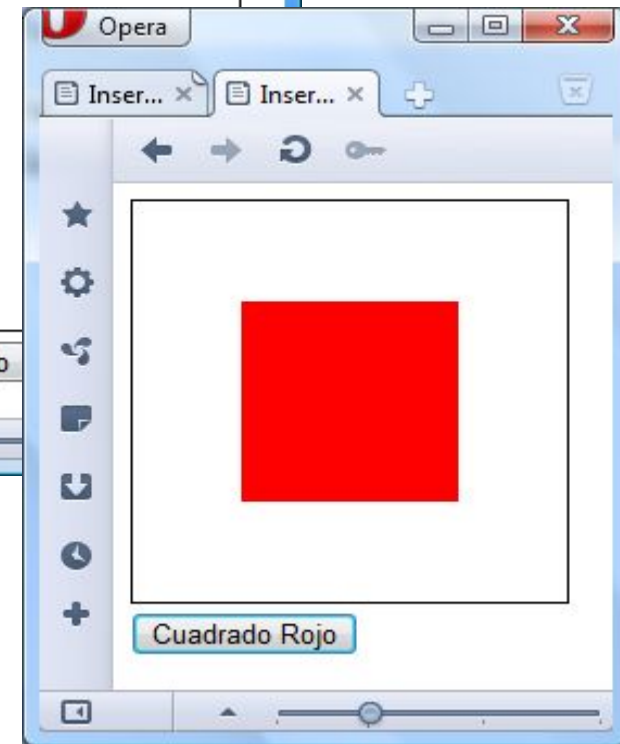
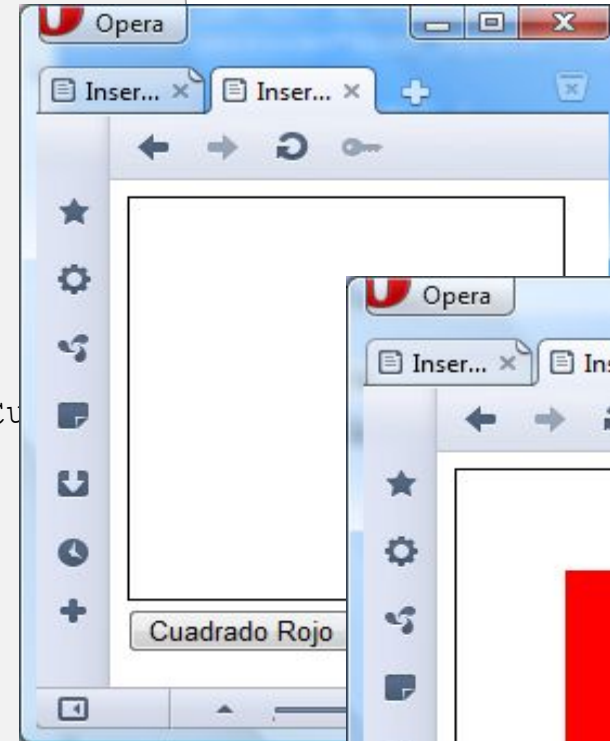


HTML (HyperText Markup Language)

Otras componentes de HTML5

Una de las incorporaciones más interesantes en HTML5 es la componente **Canvas**. Un canvas es un espacio en blanco y javascript, cumpliendo la función del lápiz y el pincel para dibujar y animar un gráfico.

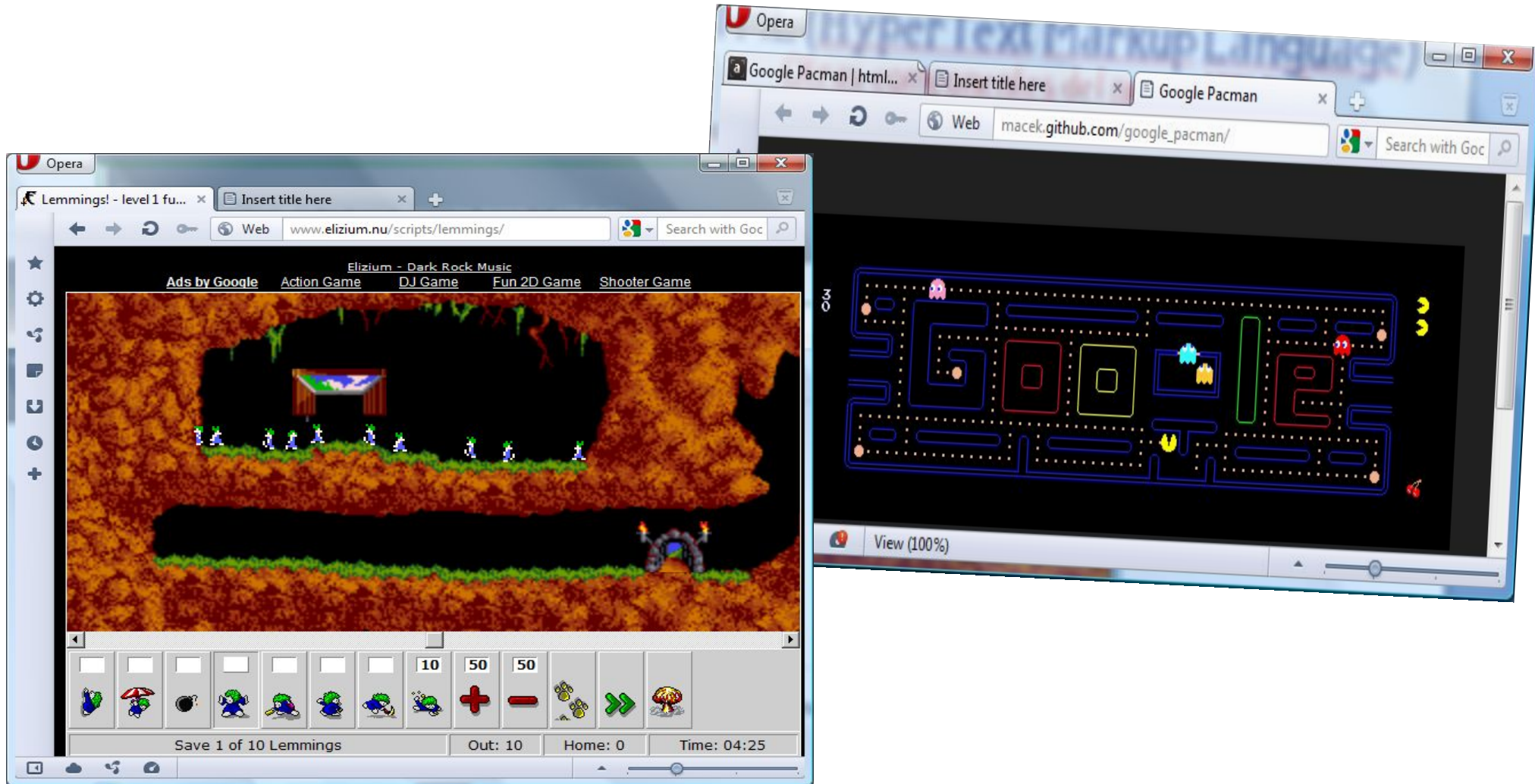
```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<canvas id="c1" width="200" height="200"
style="border:solid 1px #000000;"></canvas>
<button onclick="draw_square();return true;">Cu
Rojo</button>
<script>
function draw_square() {
  var c1 = document.getElementById("c1");
  var c1_context = c1.getContext("2d");
  c1_context.fillStyle = "#f00";
  c1_context.fillRect(50, 50, 100, 100);
}
</script>
</body>
</html>
```



HTML (HyperText Markup Language)

Otras componentes de HTML5

Ejemplos de canvas con programación más avanzada.



HTML (HyperText Markup Language)

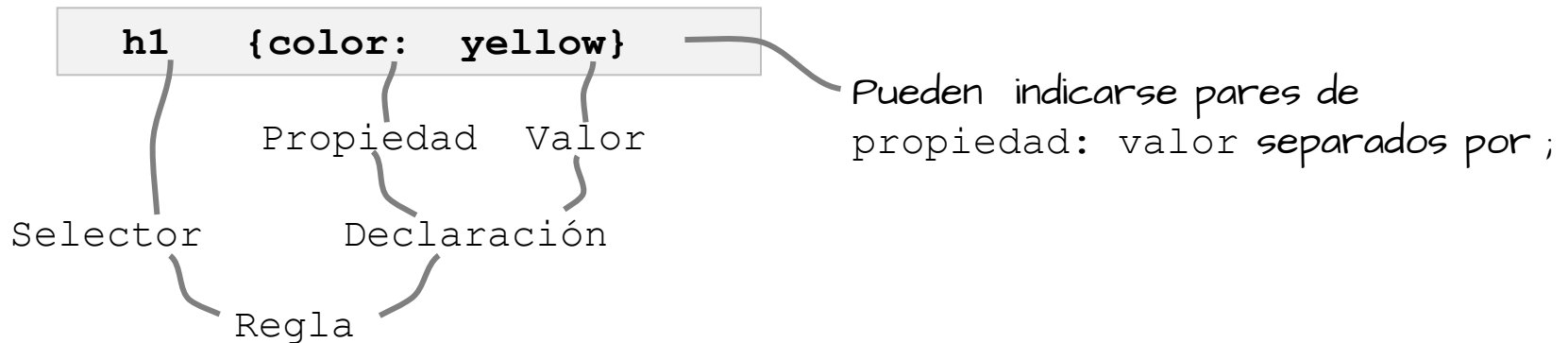
Cascading Style Sheets(CSS)

Las hojas de estilo describen o determinan la visualización de los elementos de una página web. El estándar más utilizado para la confección de los archivos de estilos es el de estilo en cascada o CSS (Cascading Style Sheets). El objetivo de trabajar con hojas de estilos es mantener el mismo aspecto en todas las páginas pertenecientes a un sitio o aplicación web.

Una hoja de estilo consiste de un **conjunto de reglas**, que definen un estilo para cada elemento o grupos de elementos HTML.

Una regla de estilo tiene dos partes:

- Un selector, que identifica el elemento o grupo al que el estilo se aplicará.
- Una declaración de propiedades que se aplicarán al selector.



HTML (HyperText Markup Language)

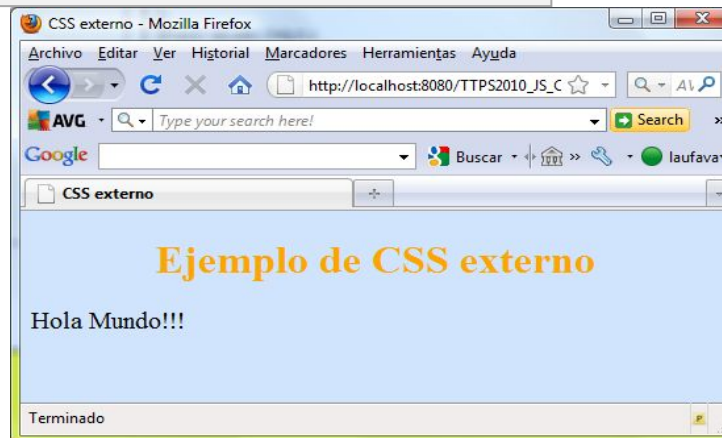
Cascading Style Sheets(CSS)

La definición de la visualización puede estar embebida en la página web o puede ser importada desde una fuente externa.

```
<html><head>                                     pruebaCSS.html
<style type="text/css">
body { background-color:#d0e4fe;}
h1{ color:orange;
    text-align:center;}
p{ font-family:"Times New Roman";
   font-size:20px;}
</style>
</head><body>
<h1>Ejemplo de CSS interno</h1>
<p>Hola Mundo!!!</p>
</body>
</html>
```

```
<html><head>                                     pruebaCSS.html
<link rel="stylesheet" type="text/css"
href="unEstilo.css"/>
<title>CSS externo</title>
</head>
<body>
<h1>Ejemplo de CSS externo</h1>
<p>Hola Mundo!!!</p>
</body></html>
```

```
html{height: 100%;}  unEstilo.css
body{
background-color:#d0e4fe;}
h1{ color:orange;
    text-align:center;}
p{font-family:"Times New Roman";
   font-size:20px;}
```



HTML (HyperText Markup Language)

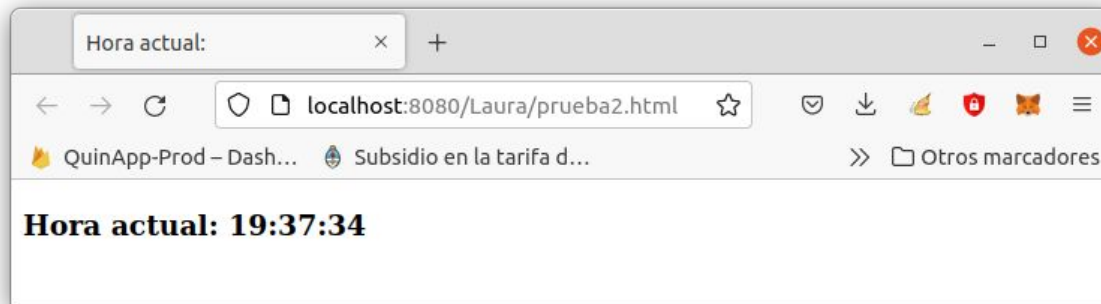
JavaScript (JS)

JavaScript es el lenguaje de programación de la web. Al igual que los CSS, los códigos JS pueden estar embebidos en la página web o pueden ser importados desde una fuente externa.

```
<!DOCTYPE html>                                     pruebaJS.html
<html lang="en-US">
<head>
<title>Hora actual: </title>
</head>
<body>
<script>
var d = new Date();
document.body.innerHTML = "<h3>Hora actual: " +
d.getHours() + ":" + d.getMinutes() + ":" +
d.getSeconds()+"</h3>";
</script>
</body>
</html>
```

```
<!DOCTYPE html>                                     pruebaJS.html
<html lang="en-US">
<head>
<title>Hora actual: </title>
</head>
<body>
<script src="myscript.js"></script>
</body>
</html>
```

```
myscript.js
var d = new Date();
document.body.innerHTML = "<h3>Hora actual: " +
+d.getHours() + ":" + d.getMinutes() + ":" +
d.getSeconds()+"</h3>";
```



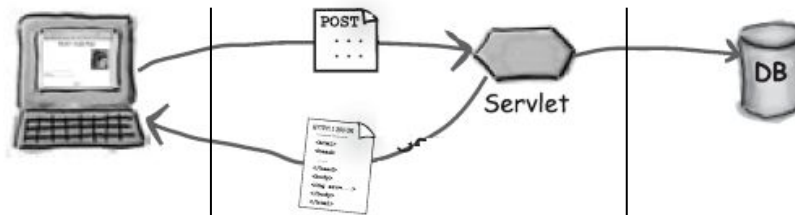
Servlets

Características generales

Servlets

Características generales

- Un servlet es una componente web escrita en Java que es gerenciada por un Contenedor Web. Procesa requerimientos y construye respuestas dinámicamente. Son ideales para realizar procesamiento en la capa del medio (middleware). Es la tecnología básica para la construcción de aplicaciones web JAVA.



- Los servlets son clases Java independientes de la plataforma, se compilan a código de bytes (bytecodes), se cargan dinámicamente y se ejecutan en un Contenedor web.
- Los servlets hacen uso de la Plataforma Java y de servicios provistos por el Contenedor Web
- La Plataforma Java provee una API robusta basada en POO que se puede usar para construir servlets.
- El Contenedor Web, evita que el programador se ocupe de la conectividad con la red, de capturar los pedidos, de producir las respuestas, seguridad, etc. Gerencia el ciclo de vida del servlet.
- La última versión de **Servlet es la 6.0**, liberada con la plataforma **JEE 10** (<https://jakarta.ee>).

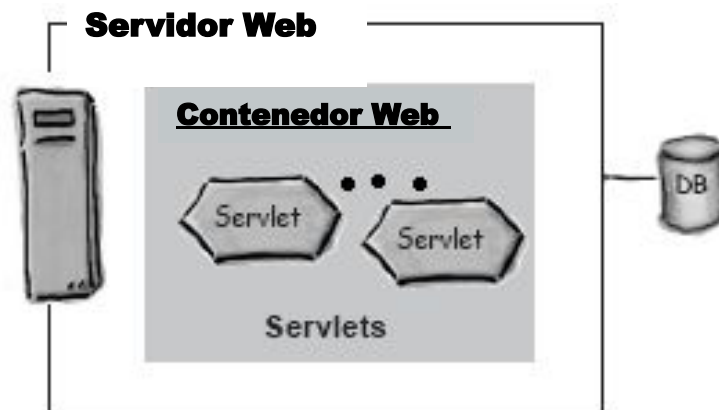
Servlets

El Contenedor Web

El Contenedor Web es responsable de:

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

Un Contenedor Web es una extensión del servidor web que provee soporte para servlets. Es parte del servidor web o del servidor de aplicaciones.



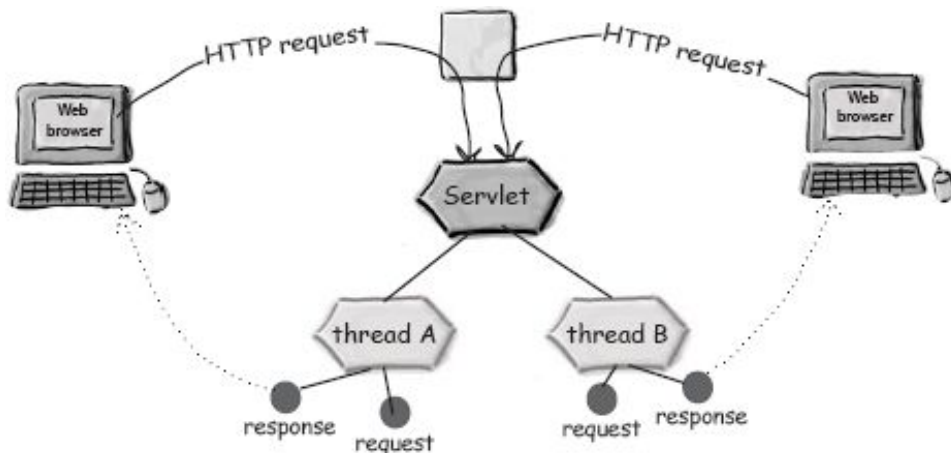
El Contenedor Web interactúa con los servlets invocando métodos de gerenciamiento o métodos callback. Estos métodos definen la interface entre el Contenedor y los servlets(API de servlets).

El Contenedor Web está construido sobre la plataforma estándar de Java (JSE™), implementa la API de servlets y todos los servicios requeridos para procesar pedidos HTTP. Cada contenedor web tiene su propia implementación de la API de servlets.

Contenedor Web

Los servlet son controlados por el Contenedor

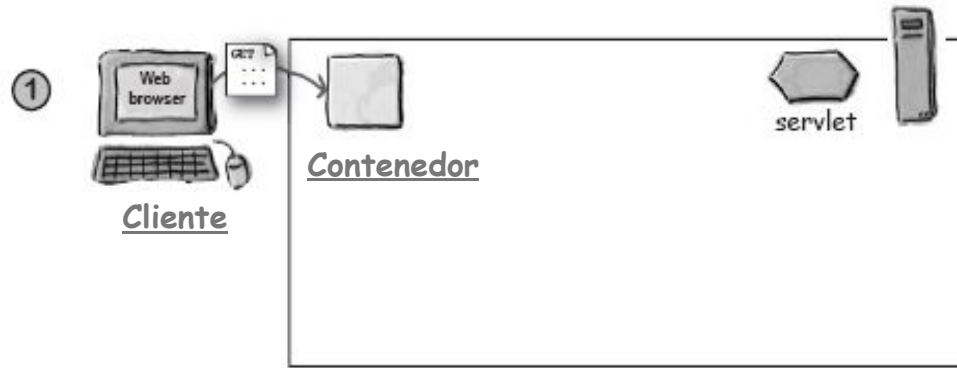
- El Contenedor Web gerencia el ciclo de vida de cada servlet, invocando a 3 métodos definidos en la interface `javax.servlet.Servlet: init(..), service(..) y destroy(..)`
- El Contenedor Web, es responsable de la carga e instanciación de los servlets, que puede suceder en el arranque, cuando una aplicación es actualizada (recargada) o puede ser postergada 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.



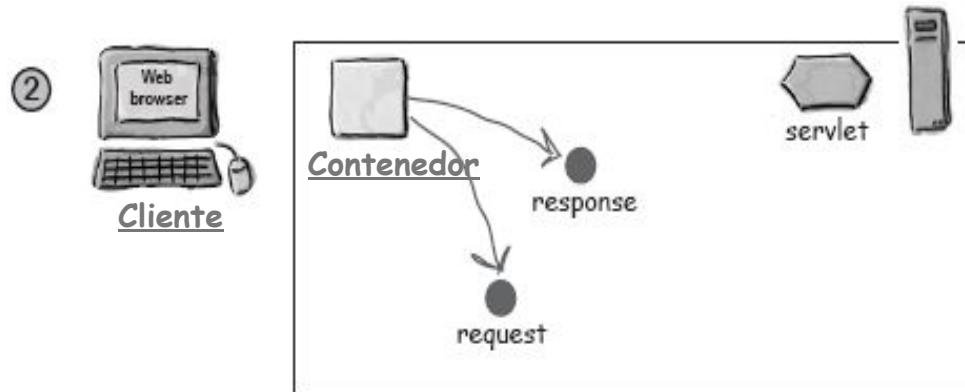
El programador de servlets, debe considerar los efectos colaterales que tiene el procesamiento concurrente y tomar las previsiones necesarias sobre los recursos compartidos (acceso a variables de instancia, variables de clase, recursos externos como archivos, etc.)

Contenedor Web

Los servlet son controlados por el Contenedor



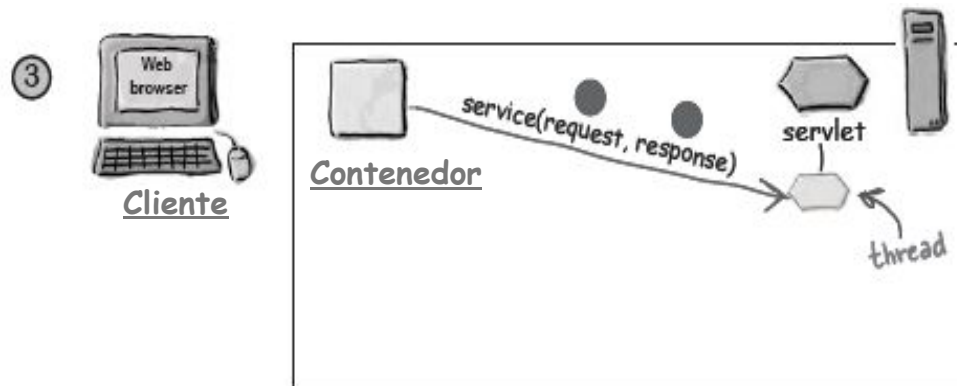
El usuario presiona sobre un link que tiene una URL a un Servlet o se invoca a través de un formulario



El Contenedor crea 2 objetos:

1) `HttpServletResponse`: representa la respuesta que se le enviará al cliente

2) `HttpServletRequest`: representa el requerimiento del cliente.



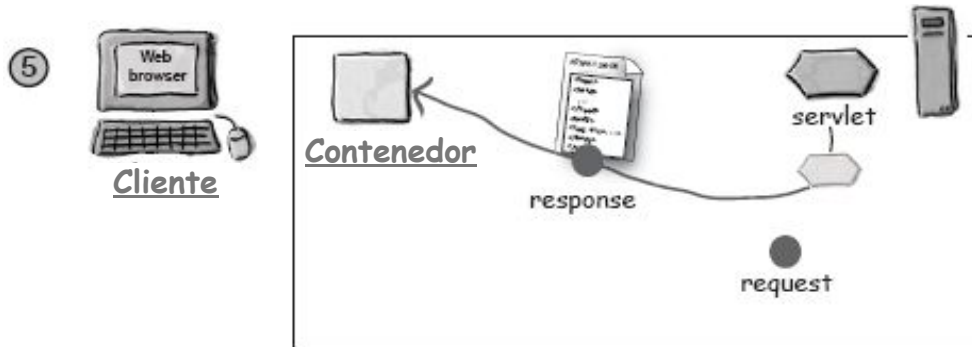
El Contenedor encuentra el servlet correcto usando la URL del requerimiento. Luego crea o aloca un thread para ese requerimiento e invoca al método `service()` pasándole los objetos `request` y `response` como argumento.

Contenedor Web

Los servlet son controlados por el Contenedor



El contenedor invoca al método `service()` del servlet, el cual determina que método invocar dependiendo del método HTTP (get o post) enviado por el cliente.



El servlet usa el objeto `response` para escribir la respuesta al cliente. La respuesta regresa a través del contenedor.



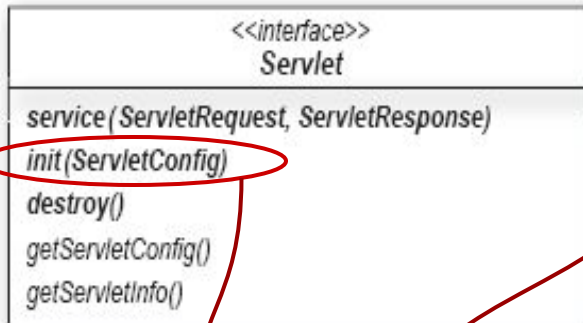
El método `service()` finaliza. El thread termina o retorna al pool de threads del contenedor. El cliente recibe la respuesta.

Servlets

Los servlets heredan los métodos del Ciclo de Vida

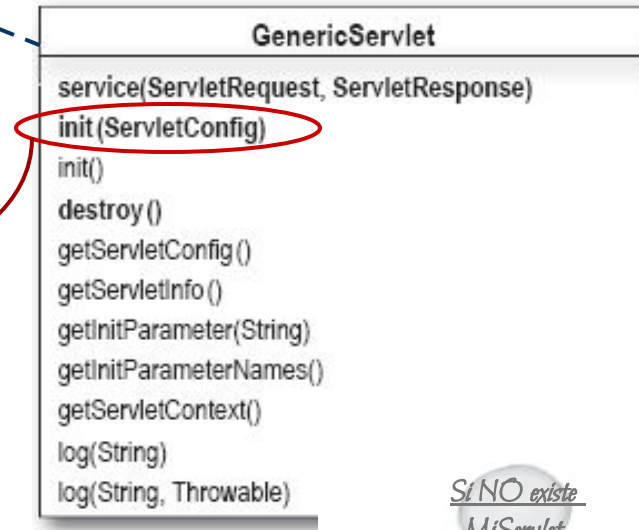
La interface Servlet
(`javax.servlet.Servlet`)

Todos los servlet tienen estos métodos, los 3 primeros son los del ciclo de vida.



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)

La clase `GenericServlet`
(`javax.servlet.GenericServlet`)
Es una clase abstracta que implementa los métodos de la interface `Servlet`.

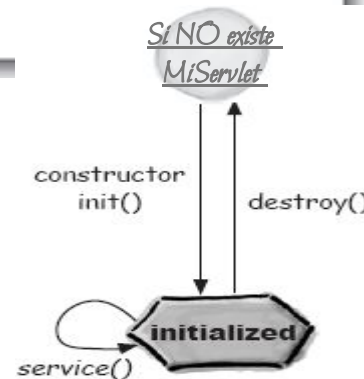


La clase `HttpServlet`
(`javax.servlet.http.HttpServlet`)

Es una clase abstracta que implementa el método `service()` para que decida qué método invocar, basado en el método HTTP.



Se sobrescriben los métodos necesarios.



Interfaces de programación

La interfaz Servlet – La clase HttpServlet

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

- **javax.servlet**: contiene la interfaz básica de servlets, llamada **Servlet**, la cual es la abstracción central de la API de servlets.

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

- **javax.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 HttpServlet provee una implementación específica para HTTP de la interface javax.servlet.Servlet. Agrega métodos adicionales, que son invocados automáticamente por el método **service()** para ayudar al procesamiento de requerimientos HTTP. Es la clase a partir de la cual se crean la mayoría de los servlets HTTP.

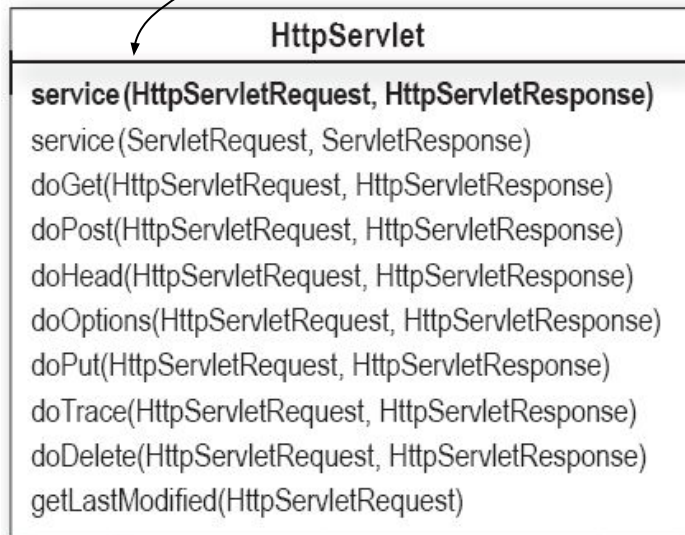
```
public void service(HttpServletRequest req, HttpServletResponse res) throws . . .
```

Interfaces de programación

La clase HttpServlet – El método service(...)

¿Cómo funciona el método **service(...)** de la clase HttpServlet?

El método **service(...)** mapea cada método del requerimiento HTTP con un método java de la clase HttpServlet.



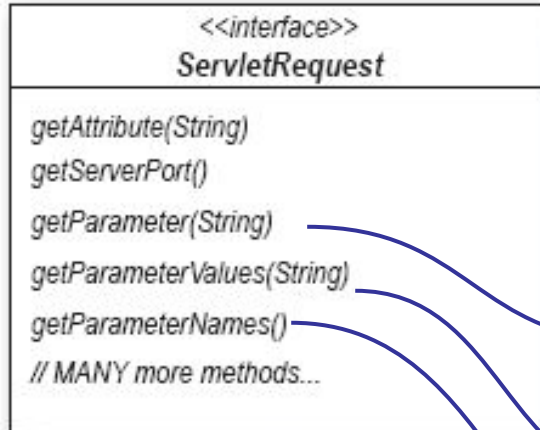
- La secuencia de métodos **service()** --> **doGet()** ó **service()** --> **doPost()** sucede cada vez que un cliente hace un requerimiento.
- Cada vez que un **doGet()** o **doPost()** ejecutan, lo hacen en un thread separado.
- Cuando se escribe un Servlet, los métodos que se sobrescriben son **doGet()** y **doPost()**, los restantes están relacionados con la programación más cercana al protocolo HTTP.

Interfaces de programación

Las interfaces `HttpServletRequest`

ServletRequest interface

(`javax.servlet.ServletRequest`)



El requerimiento HTTP de un cliente está representado por un objeto **`HttpServletRequest`**. Un objeto **`HttpServletRequest`** se puede usar para recuperar el header del requerimiento HTTP, recuperar los parámetros del requerimiento HTTP, asociar atributos con el requerimiento, redireccionar requerimientos entre servlets, recuperar la sesión del usuario, etc.

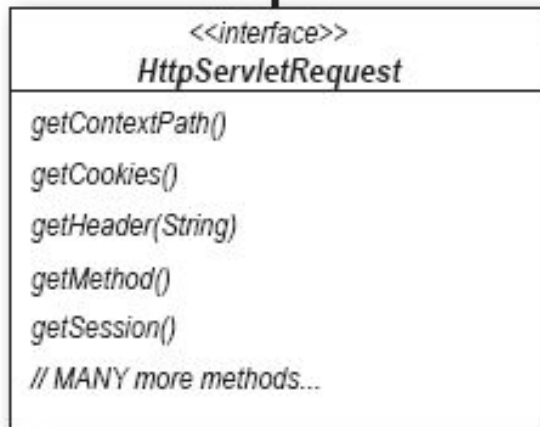
Devuelve un **`String`** con el valor del parámetro del requerimiento con la clave dada. Si hay múltiples valores para esa parámetro, devuelve el primero.

Retorna un arreglo de **`Strings`** que contiene todos los valores de un parámetro del request con la clave dada o null si el parámetro no existe.

Devuelve un List de **`Strings`** con los nombres de todos los parámetros del requerimiento.

HttpServletRequest interface

(`javax.servlet.http.HttpServletRequest`)

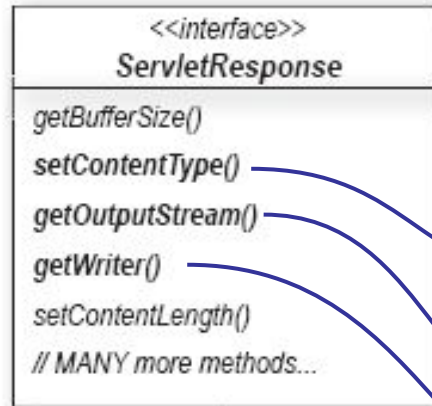


Interfaces de programación

Las interfaces HttpServletResponse

ServletResponse interface

(javax.servlet.ServletResponse)



El objeto **HttpServletResponse** representa la respuesta que se le enviará al cliente. Por defecto, la respuesta HTTP está vacía. Para generar una respuesta customizada, es necesario usar los métodos **getWriter()** o **getOutputStream()**, para obtener un stream de salida donde escribir contenido.

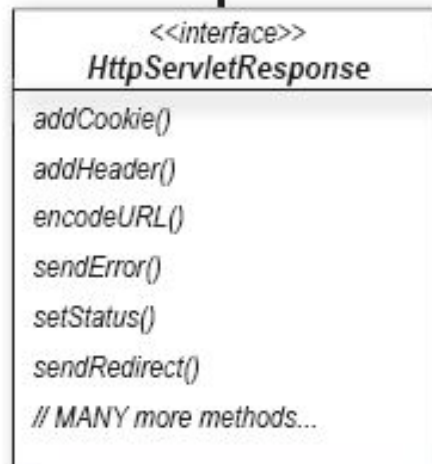
Permite setear el tipo MIME de la respuesta HTTP (text/html, image/JPG, etc.) antes de devolver la respuesta.

El objeto **ServletOutputStream** es usado para enviar al cliente datos binarios (imágenes por ejemplo).

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

HttpServletResponse interface

(javax.servlet.http.HttpServletResponse)



Un ejemplo simple

Un servlet que recupera parámetros del requerimiento

Este Servlet genera una página HTML usando un parámetro del requerimiento

```
<html>
<body>
<form action="ServletHola" method="post">
  Ingresá tu nombre: <input type="text" name="nombre">
  <input type="submit" value="Enviar">
</form>
</body>
</html>
```

saludo.html

HttpServlet, extiende
GenericServlet, la cual
implementa la interfaz
Servlet

```
package servlets;
public class ServletHola extends javax.servlet.http.HttpServlet {
    protected void doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) throws
        ServletException, IOException {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        out.println("<h1> Hola " + request.getParameter("nombre") + " </h1>");
        out.print(" </body></html>");
        out.close();
    }
}
```

A partir del objeto response se puede obtener un objeto
PrintWriter que nos permite escribir texto HTML en la
respuesta

A partir del objeto request se puede obtener los
parámetros del requerimiento

El archivo descriptor de la Aplicación Web

- El archivo descriptor de la aplicación web, **web.xml**, define TODO lo que el servidor necesita conocer sobre la aplicación web.
- Es estándar y se ubica SIEMPRE en la carpeta **/WEB-INF/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 los archivos **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.

Los IDEs (Eclipse, IntelliJ, JDeveloper, etc.) 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.

Usando el archivo descriptor de la Aplicación Web

- 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 descriptor de la aplicación web o **web.xml**. (también se lo puede hacer mediante anotaciones). Además, el archivo **.class** del servlet se debe ubicar en la carpeta estándar **/WEB-INF/classes** de la aplicación web, junto con otras clases Java. Cualquier contenido de la carpeta **/WEB-INF** **no está accesible directamente por un cliente http**.
- El archivo **web.xml** usa los elementos **<servlet>** y **<servlet-mapping>** para declarar los servlets que serán cargados en memoria por el contenedor web y definir URL's serán atendidas por cada servlet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ... http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
<servlet>
  <servlet-name>ServletHola</servlet-name>
  <servlet-class>servlets.ServletHola</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ServletHola</servlet-name>
  <url-pattern>/ServletHola</url-pattern>
</servlet-mapping>
</web-app>
```

web.xml

Se mapea un servlet con una URL
Este es un mapeo 1 a 1, /ServletHola
(siempre empieza con /)

URL completa del servlet: **http://www.servidor.gov.ar:8080/appPruebas/ServletHola**

URL de la aplicación web

url-pattern o mapping

Parámetros de Inicialización

ServletHola

Este Servlet retorna una página HTML con un mensaje concatenando un parámetro de inicialización con un parámetro del requerimiento.

```
public class ServletHola extends HttpServlet {
    private String saludo;
    public void init(){
        saludo = this.getServletConfig().getInitParameter("saludo");
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ..{
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>"+saludo+request.getParameter("nombre")+" </h1>");
        out.println("</body></html>");
        out.close();
    }
}
```

También se podría hacer sin sobrescribir el método `init()` y recuperar los valores de inicialización cuando se quiera usar, de alguna de estas dos maneras:

```
this.getInitParameter("saludo")
this.getServletConfig().getInitParameter("saludo")
```

```
...
<servlet>
  <servlet-name>MiServlet</servlet-name>
  <servlet-class>servlets.ServletHola</servlet-class>
  <init-param>
    <param-name>saludo</param-name>
    <param-value>Hola</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>MiServlet</servlet-name>
  <url-pattern>/ServletHola</url-pattern>
</servlet-mapping>
...
```

web.xml

Servlets con Anotaciones

Una alternativa al web.xml

A partir de la versión de Servlet 3.0 se pueden utilizar anotaciones para la configuración de los Servlets. Las anotaciones en la API de servlets se utilizan para reemplazar a las declaraciones/los mapeos del archivo `web.xml`.

¿Qué son las anotaciones?

- Las anotaciones son metadatos que nos permiten agregar información a nuestro código fuente para ser usado posteriormente –en tiempo de compilación o en tiempo de ejecución-.
- Las anotaciones fueron incorporadas al lenguaje java en la versión 5. La motivación de las anotaciones es la tendencia a combinar metadatos con código fuente, en lugar de mantenerlos en archivos descriptores separados.

La API de Servlets 3.0 o superior

Anotaciones

Las anotaciones **se declaran** de manera parecida a las interfaces, solo que el signo **@** precede a la palabra clave **interface**. Se compilan a archivos **.class** de la misma manera que las clases e interfaces.


```
package java.lang;
import java.lang.annotation.*;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override {
}
```

Las anotaciones **se utilizan** en el código fuente precediendo a la declaración de la clase, atributos y métodos. En este caso, la anotación **@Override** es para métodos y se usa así:

```
public class Paciente {
    @Override
    public String toString() {
        return super.toString();
    }
}
```

Precede a un
método



La anotación **@Override** indica que se está sobrescribiendo un método de la superclase. Si un método está precedido por esta anotación pero NO sobrescribe el método de la superclase, los compiladores deben generar un mensaje de error y la clase no compila.

@Target: indica dónde se aplican las anotaciones (métodos, clases, variables de instancia, variables locales, paquetes, constructores, etc.)

@Retention: indica dónde están disponibles las anotaciones y cuánto se mantiene la información de las anotaciones. Esto permite determinar si pueden ser leídas sólo por el compilador o por el intérprete en tiempo de ejecución. Los valores posibles son: **RetentionPolicy.SOURCE**, **RetentionPolicy.CLASS** y **RetentionPolicy.RUNTIME**.

La API de Servlets 3.0 o superior

Anotaciones

Este es un ejemplo de una declaración de la anotación `@Column` para el mapeo de objetos con tablas de una base de datos, donde tiene entre otros el método `name()` para identificar en nombre de la columna en la tabla de la base de datos.

Definición de la anotación `@column`

```
import javax.persistence.*;
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@interface Column extends Annotation{

    public String name() default "";
    . . .
}
```

La declaración de una anotación al igual que las interfaces tiene métodos abstractos pero además puede tener valores por defecto.

Uso de la anotación `@column`

```
package taller;

import javax.persistence.*;
@Entity
@Table(name="MENSAJES")
public class Mensaje {

    @Column(name="MENSAJE_ID")
    private Long id;
    . . .
}
```

Preceden a la declaración de la clase

La anotación tiene una lista entre paréntesis de pares **elemento-valor**. Los valores de los elementos deben ser constantes definidas en compilación

La API de Servlets 3.0+

Un Servlet con Anotaciones

La anotación **@WebServlet** es usada para declarar la configuración de un Servlet. Si no se especifica el atributo **name** se usa el nombre de la clase.

```
package misServlet;
```

```
@WebServlet(  
    urlPatterns = { "/ServletHola" },  
    initParams = {  
        @WebInitParam(name = "saludo", value = "Hola")  
    })
```

El atributo **urlPatterns** define un conjunto de url-patterns que pueden ser usadas para invocar al Servlet.

La anotación **@WebInitParam** se usa para definir los parámetros de inicialización del servlet

```
public class ServletFecha extends HttpServlet{  
    private String saludo;
```

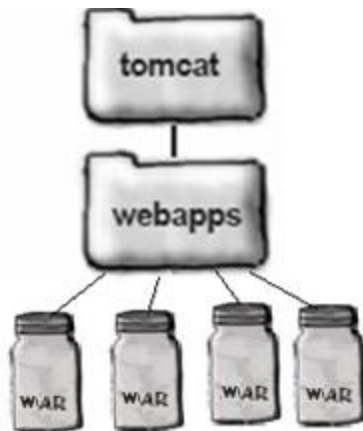
```
    public void init(){  
        saludo = this.getServletConfig().getInitParameter("saludo");  
    }
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ..{  
        PrintWriter out = response.getWriter();  
        out.println("<html><body>");  
        out.println("<h1>"+saludo+request.getParameter("nombre")+" </h1>");  
        out.println("</body></html>");  
        out.close();  
    }  
}
```

¿Cómo se hace el “deploy” de una aplicación?

- Las aplicaciones web JAVA pueden empaquetarse 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, donde la raíz, es el “*context root*” de la aplicación web.
- El archivo WAR es un archivo JAR que contiene un módulo web: páginas HTML, archivos de imágenes, JSPs, clases, páginas de estilo, código JavaScript, el directorio **WEB-INF** y sus subdirectorios (classes, lib, tag, el archivo web.xml, etc.)
- Los archivos WAR están definidos oficialmente en la especificación de Servlets a partir de la versión 2.2. **Son estándares**. Todos los contenedores que implementan la especificación de la API de Servlets 2.2 y superiores deben soportar archivos WAR.
- Los IDEs proveen opciones que permiten construir el WAR en forma automática. Se puede crear el archivo WAR usando la herramienta *jar* del JDK.

En el servidor Tomcat, el archivo WAR de la aplicación web se debe copiar en el directorio **webapps**



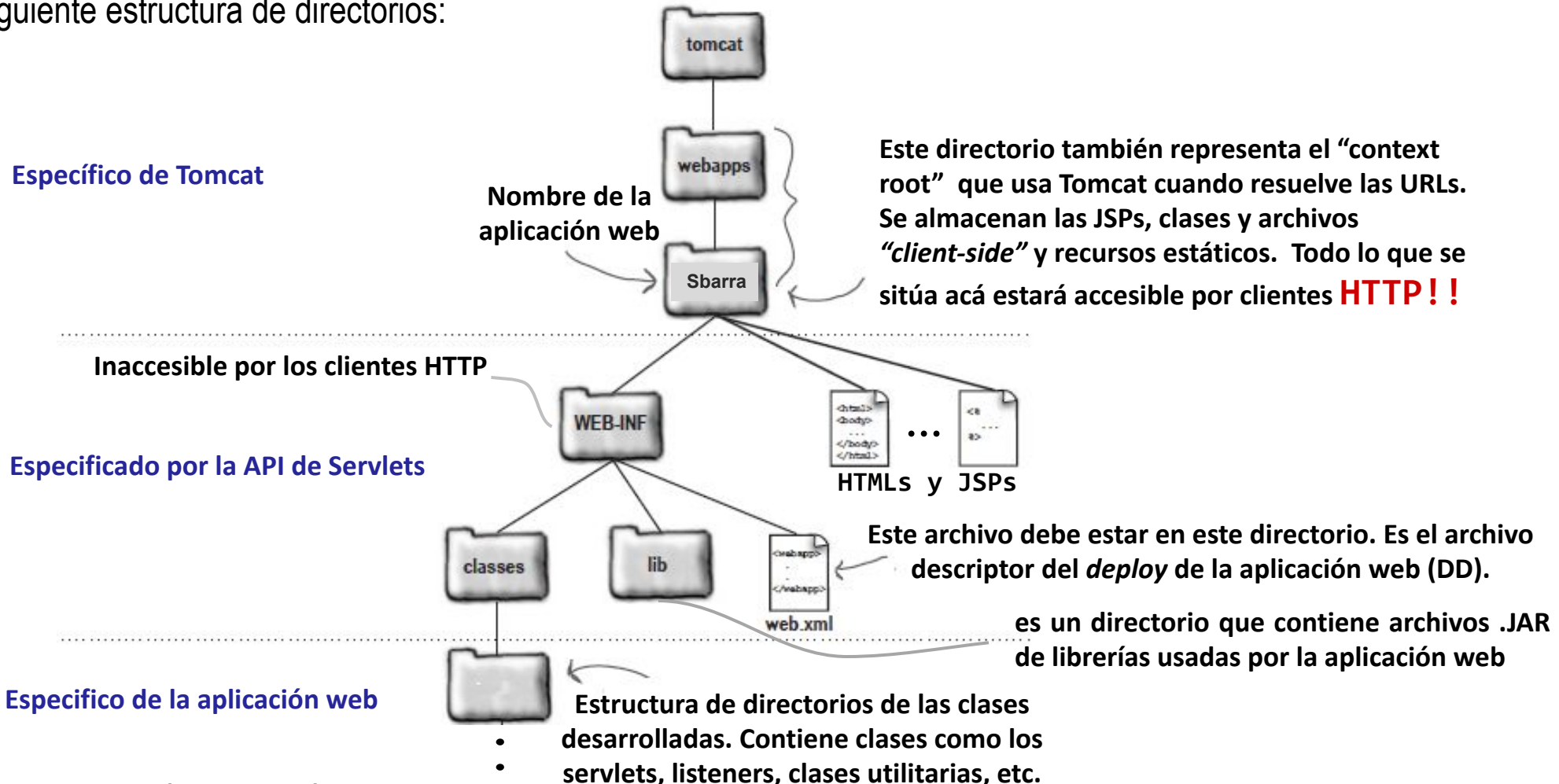
Cuando Tomcat arranca, automáticamente expande a partir de webapps el contenido de cada uno de los archivos .war al formato “desempaquetado”.

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.

El Módulo Web

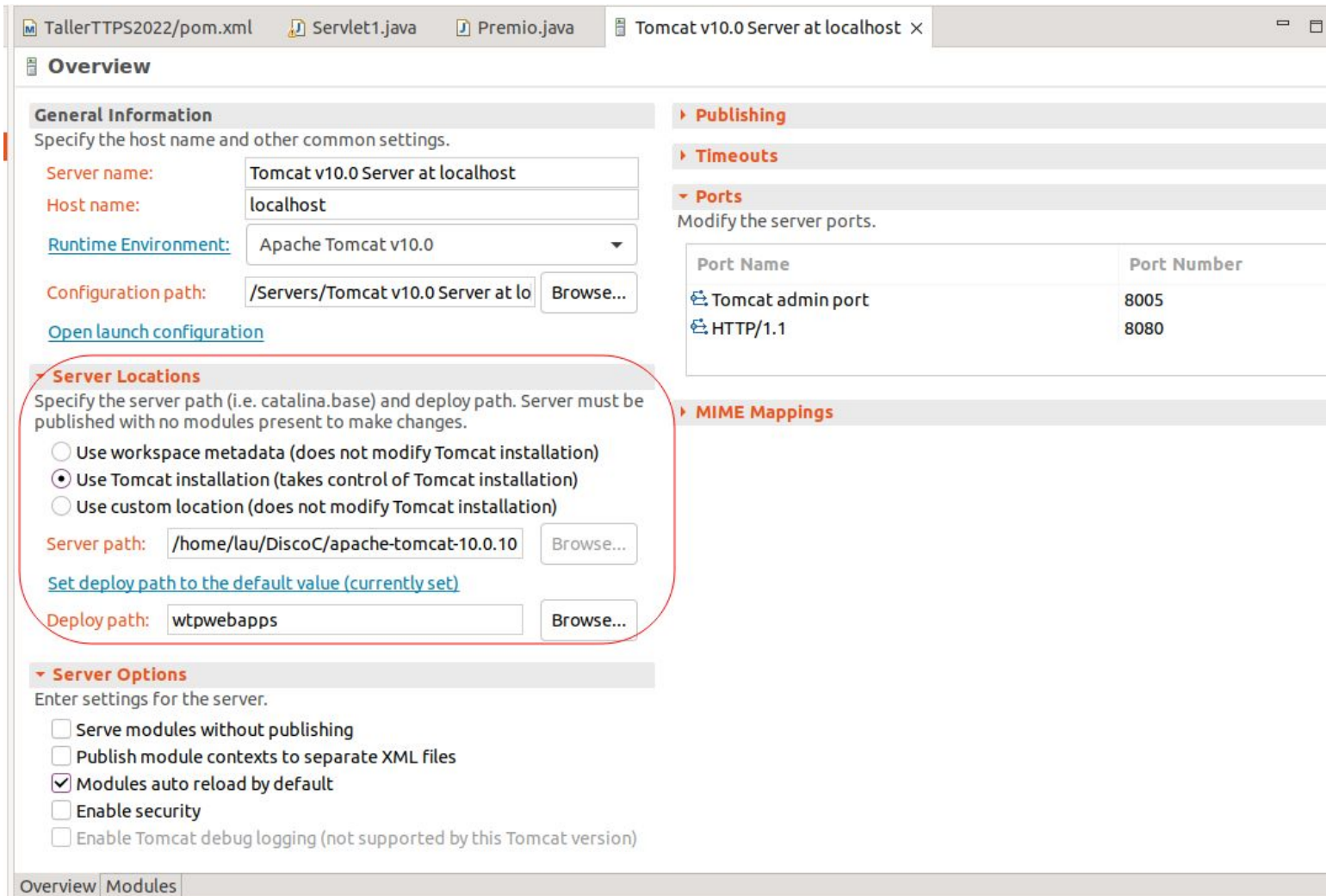
Un módulo web es una unidad “*desplegable*” de recursos web (componentes web y archivos estáticos que pueden referenciarse por una URL). También puede contener clases utilitarias “*server-side*” (por ej: javaBeans) y clases “*client-side*” (applets y clases utilitarias).

De acuerdo a la especificación de Servlets, un módulo web se corresponde con una aplicación web y tiene la siguiente estructura de directorios:



¿Cómo se hace el “deploy” de una aplicación desde Eclipse?

- Cuando se ejecuta una aplicación web en un servidor, se puede configurar el lugar donde se hará el deploy de tal aplicación.



La plataforma Java EE ya no será mantenida por ORACLE



Java EE -> Jakarta EE

Java EE se ha retirado y ahora la plataforma estándar se llama Jakarta EE

- Cuál es el motivo?

ORACLE, dueños de Java, deciden no continuar con la definición de la documentación y el desarrollo de la plataforma EE y transferir Java EE a una base de código abierto.

En coordinación con los socios de Java EE, Red Hat e IBM, se decidió transferir Java EE junto con la implementación de referencia completa y el Kit de compatibilidad de tecnología (TCK) a la Fundación Eclipse.

El motivo de esta selección es que la Eclipse tiene gran experiencia y participación en Java EE, lo que permitió una rápida transición.

- Por qué Jakarta?

Jakarta era una marca registrada de Apache, con una historia significativa por las contribución a las soluciones Java de código abierto. Apache Jakarta se retiró en 2011. Los derechos del nombre se concedieron a Eclipse.



Java 2 EE -> Jakarta EE

Desde J2EE hasta Jakarta EE

1999

Sun Microsystems lanza J2EE como una plataforma para construir aplicaciones empresariales con JAVA

2010

Oracle adquiere a Sun Microsystems

2019

JAKARTA EE 8 es released con toda la funcionalidad equivalente a Java EE 8 pero con una nueva licencia.

2022

Salió JAKARTA EE 10, primer released desde que Java EE 8 se pasó a Eclipse

2006

Nueva versión de J2EE con nuevas técnicas de programación (anotaciones, tipos genéricos, etc.) y es renombrada como JEE

2017

Oracle dona Java EE a la **Fundación Eclipse**

2020

Salió JAKARTA EE 9 con la misma funcionalidad de **Jakarta 8 pero con el nuevo espacio de nombres Jakarta**

Java EE -> Jakarta EE

Qué cambios involucra?
Renombrar javax por jakarta

```
1 package misServlets;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9
10 /**
11  * Servlet implementation class PruebaJakarta
12  */
13 @WebServlet("/PruebaJakarta")
14 public class PruebaJakarta extends HttpServlet {
15     private static final long serialVersionUID = 1L;
16
17     /**
18      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
19      */
20     protected void doGet(HttpServletRequest request, HttpServletResponse response) {
21         // TODO Auto-generated method stub
22         response.getWriter().append("Served at: ").append(request.getDate());
23     }
24
25     /**
26      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
27      */
28     protected void doPost(HttpServletRequest request, HttpServletResponse response) {
29         // TODO Auto-generated method stub
30         doGet(request, response);
31     }
32 }
```

```
1 package misServlets;
2
3 import java.io.IOException;
4 import jakarta.servlet.ServletException;
5 import jakarta.servlet.annotation.WebServlet;
6 import jakarta.servlet.http.HttpServlet;
7 import jakarta.servlet.http.HttpServletRequest;
8 import jakarta.servlet.http.HttpServletResponse;
9
10 /**
11  * Servlet implementation class PruebaJakarta
12  */
13 @WebServlet("/PruebaJakarta")
14 public class PruebaJakarta extends HttpServlet {
15     private static final long serialVersionUID = 1L;
16
17     /**
18      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
19      */
20     protected void doGet(HttpServletRequest request, HttpServletResponse response) {
21         // TODO Auto-generated method stub
22         response.getWriter().append("Served at: ").append(request.getDate());
23     }
24
25     /**
26      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
27      */
28     protected void doPost(HttpServletRequest request, HttpServletResponse response) {
29         // TODO Auto-generated method stub
30         doGet(request, response);
31     }
32 }
```

Java EE -> Jakarta EE

Tener en cuenta

Eclipse IDE	Contenedor web	
Eclipse soporta la creación automática de componentes de la API de Servlet de ORACLE javax.servlet.* javax.servlet.http.*	Tomcat 9 (e inferiores) Implementa javax.servlet	No se debe hacer ninguna modificación al crear componentes desde el IDE. Se trabaja con Java EE 8 máximo.
	Tomcat 10+ Implementa jakarta.servlet y no implementa javax.servlet	Se debe hacer un renombre de paquetes javax -> jakarta Se puede trabajar con Jakarta EE 8 - Jakarta EE 9 - Jakarta EE 10

Java EE Platform (Oracle)

- <https://javaee.github.io/tutorial/>

Jakarta EE Platform (Eclipse Foundation)

- <https://jakarta.ee/specifications/platform/8/platform-spec-8.html#architecture>

Compatibilidades

- <http://tomcat.apache.org/whichversion.html>

Referencias

El Lenguaje HTML

- <http://www.w3c.org/html>
- <http://www.htmlquick.com/es/reference.html>
- <http://www.w3.org/TR/html401/struct/tables.html#h-11.2.3>

HTML5 y CSS

- <http://www.w3.org/html/wg/html5/>
- <http://www.w3schools.com/css/default.asp>

Servlets y JavaServer Pages, Jayson Falkner, Kevin Jones

Head First Servlets & JSP, Bryan Basham, Kathy Sierra, Bert Bates. O'Reilly

Herramientas necesarias para el desarrollo de aplicaciones web:

- **Eclipse**, IDE para desarrollar aplicaciones Java EE.
- Apache **Tomcat 9**, (implementa las especificaciones Java EE8).
- Apache Tomcat 10+ (implementa las especificaciones Jakarta EE 8)
- La plataforma estándar, **JSE 11+ (JDK)**