

Taller de Tecnologías de Producción de Software

Cursada 2019

Docentes

**Laura Fava, Jorge Rosso, Vanessa Aybar, Luciano
Nomdedeu, Manuel Ortiz, Ezequiel Boccalari**

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:

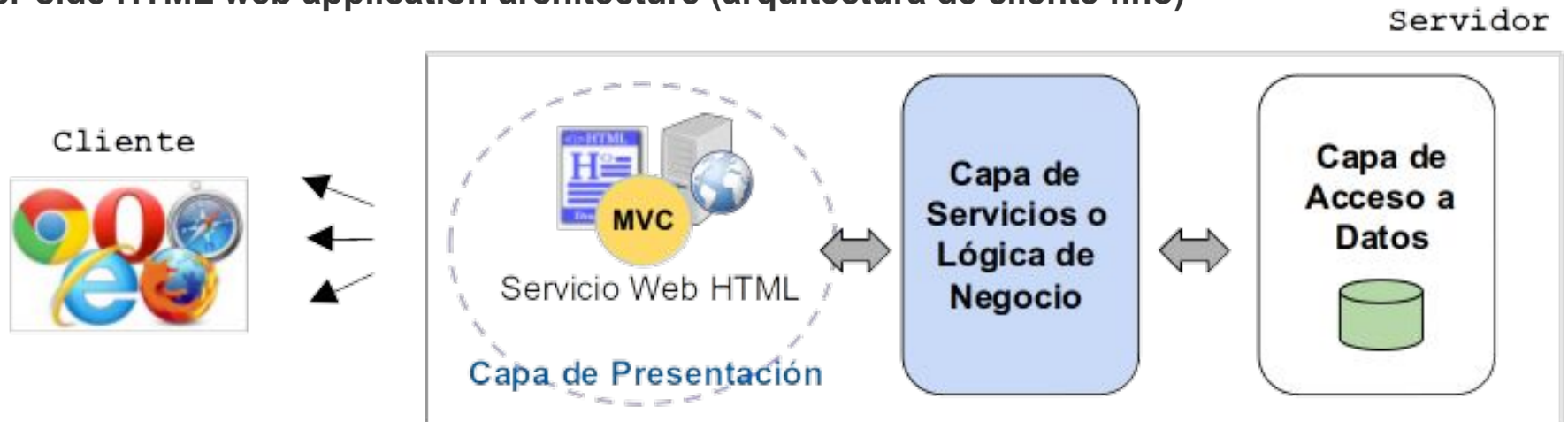
- **Frontend (client-side code):** Es el código que ejecuta en el navegador de la máquina del cliente y permite hacer peticiones y recibir/ver respuestas a esas peticiones.
- **Backend (server-side code):** Es el código que está en el o los servidores, los cuales procesan y responden a los requerimientos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



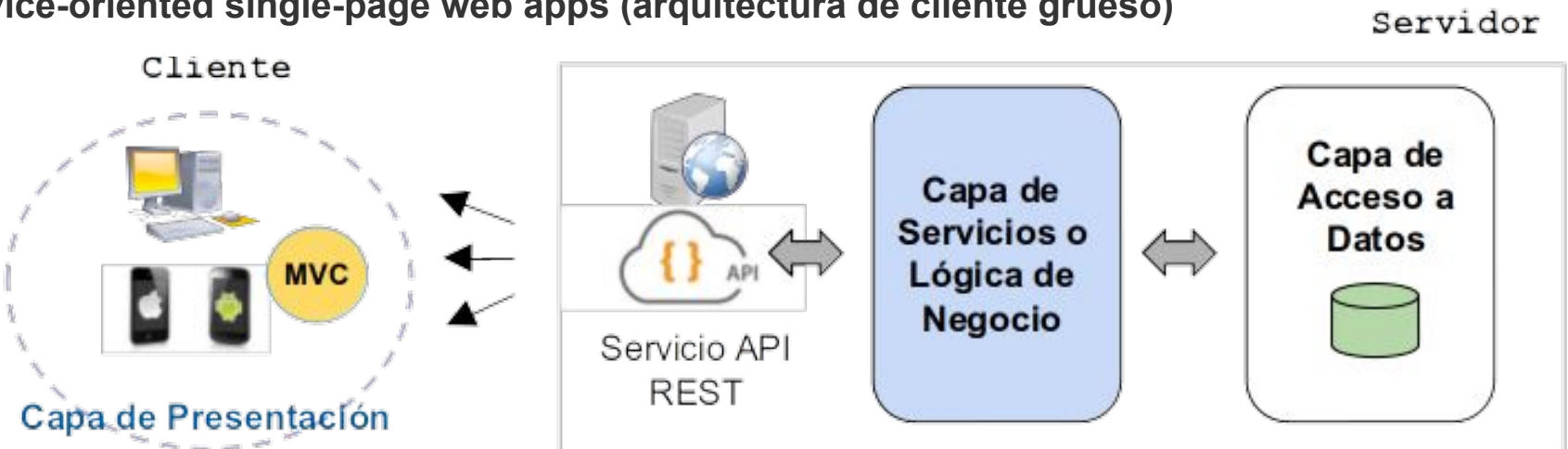
Aplicaciones web

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*.

¿Cual 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.

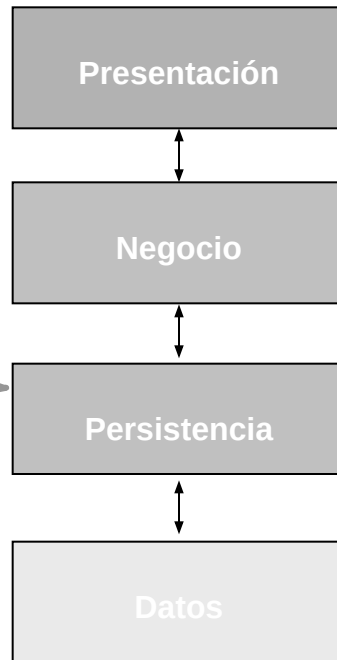
Arquitectura de una aplicación

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

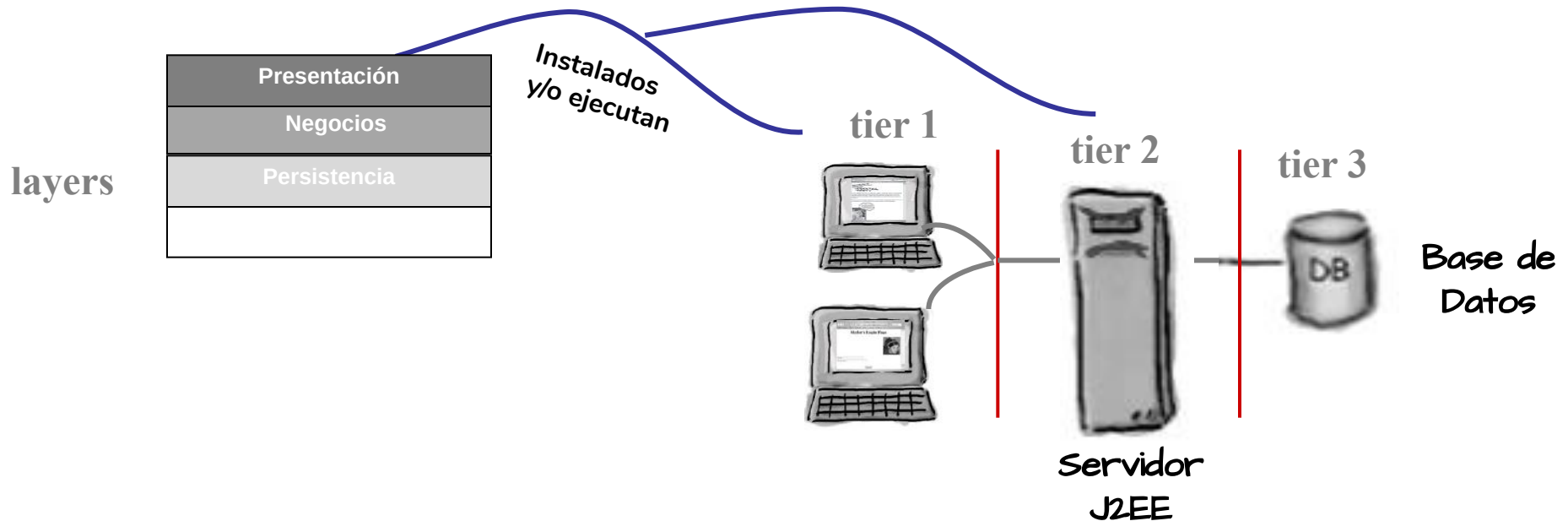
Representa los datos subyacentes. Es la representación persistente de los datos del sistema.

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” comunmente en una máquina cliente, un servidor JEE y un servidor de datos.

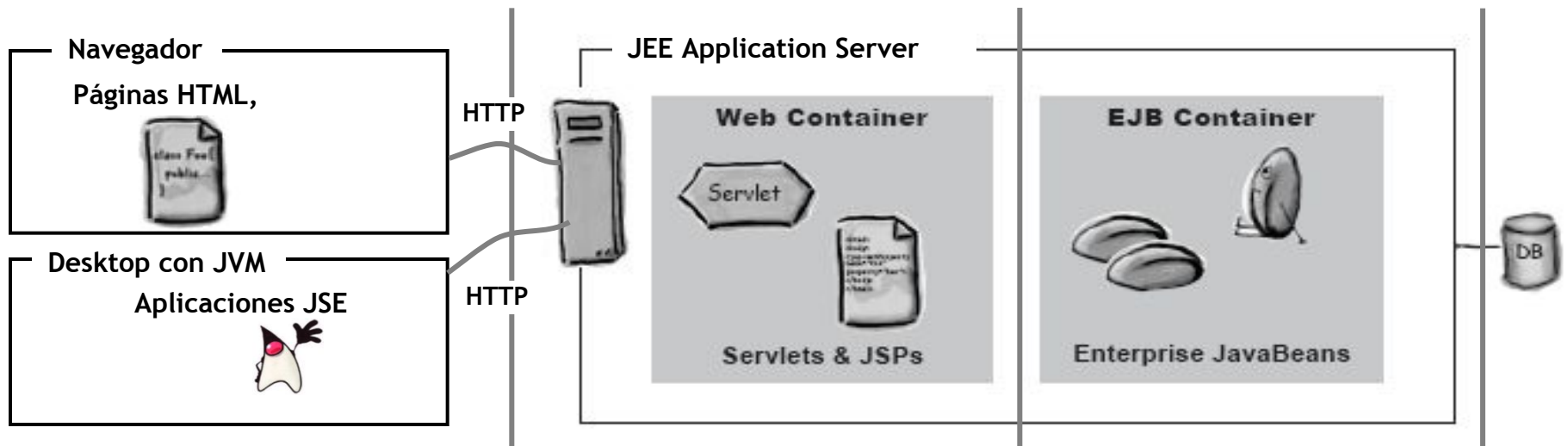
Arquitectura de Java EE

Componentes

La plataforma Java EE 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, listeners) , JSPs y JavaServer Faces
- **Componentes Empresariales:** Enterprise JavaBeans



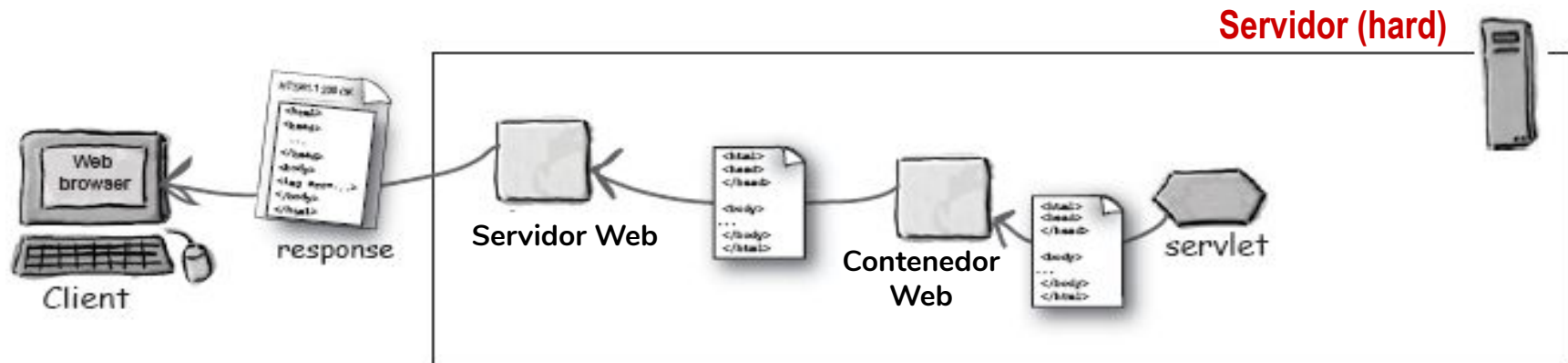
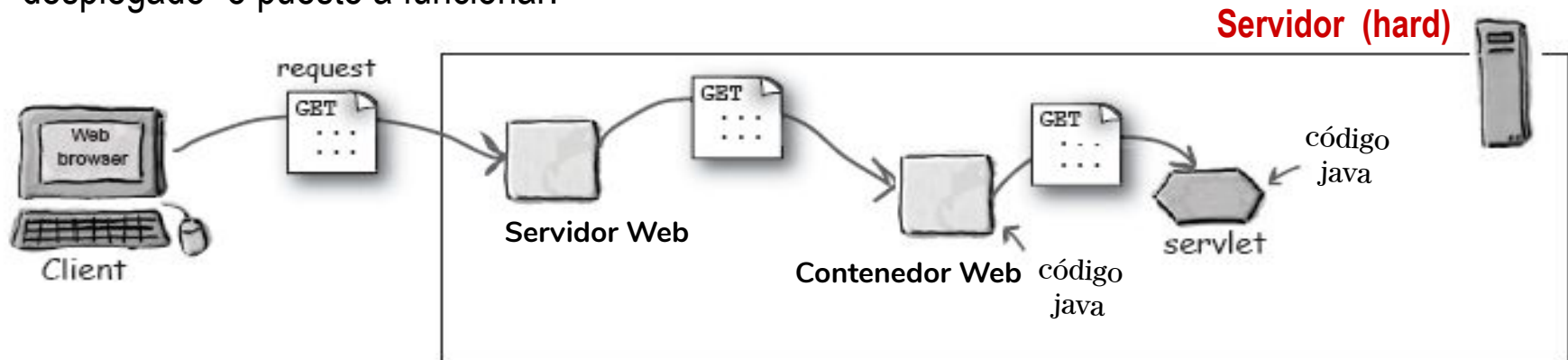
Una aplicación JEE está basada en componentes, donde una componente JEE es una unidad funcional de software, que debe correr dentro de un contenedor y que puede comunicarse con el resto de las componentes.

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 JEE

Servidores certificados que cumplen la especificación J2EE:






	Código Fuente Abierto	Propietarios
Contenedor web (solamente)	Tomcat (Apache) Jetty (Eclipse Foundation) Resin (Caucho Technology)	
Contenedor JEE completo (Application Server)	GlassFish JBoss AS JOnAS Geronimo (Apache) Resin	WebSphere Application Server (IBM) WebLogic (BEA System) iPlanet (SUN & Netscape) OC4J (Oracle) Resin Pro



Arquitectura JEE

Entornos para desarrollo de aplicaciones JEE

Los entornos de desarrollo o IDEs (Integrated Development Environment) para desarrollo de aplicaciones JEE:

IDE	Propietario	
Eclipse	-	
NetBeans 8.1	SUN – ORACLE	
JDeveloper	ORACLE	
IntelliJ IDEA	-	
IBM Rational Application Developer (RAD)	IBM	

Eclipse IDE para desarrolladores usando la plataforma empresarial JEE (Eclipse IDE for Enterprise Java Developers)

<https://www.eclipse.org/downloads/packages/release/2020-06/r/eclipse-ide-enterprise-java-developers>

Clientes y servidores

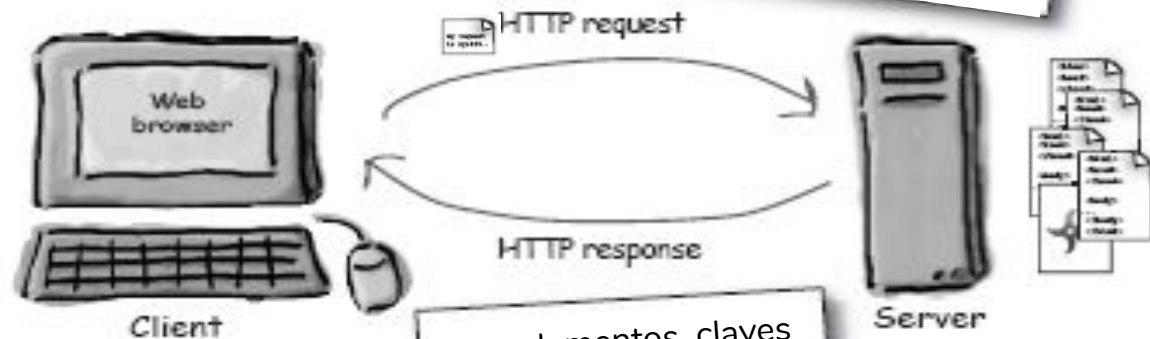
conocen HTTP y HTML

¿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 claves de un **request**:

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



Los elementos claves 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 internos, los 4xx son que no encontró lo que se pide, los 3xx son redirecciones y los 2xx ok

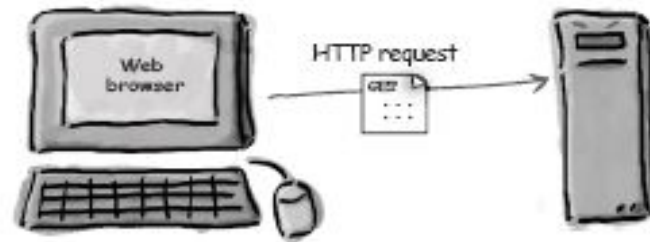
¿Qué es HTML?

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.

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 mas 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

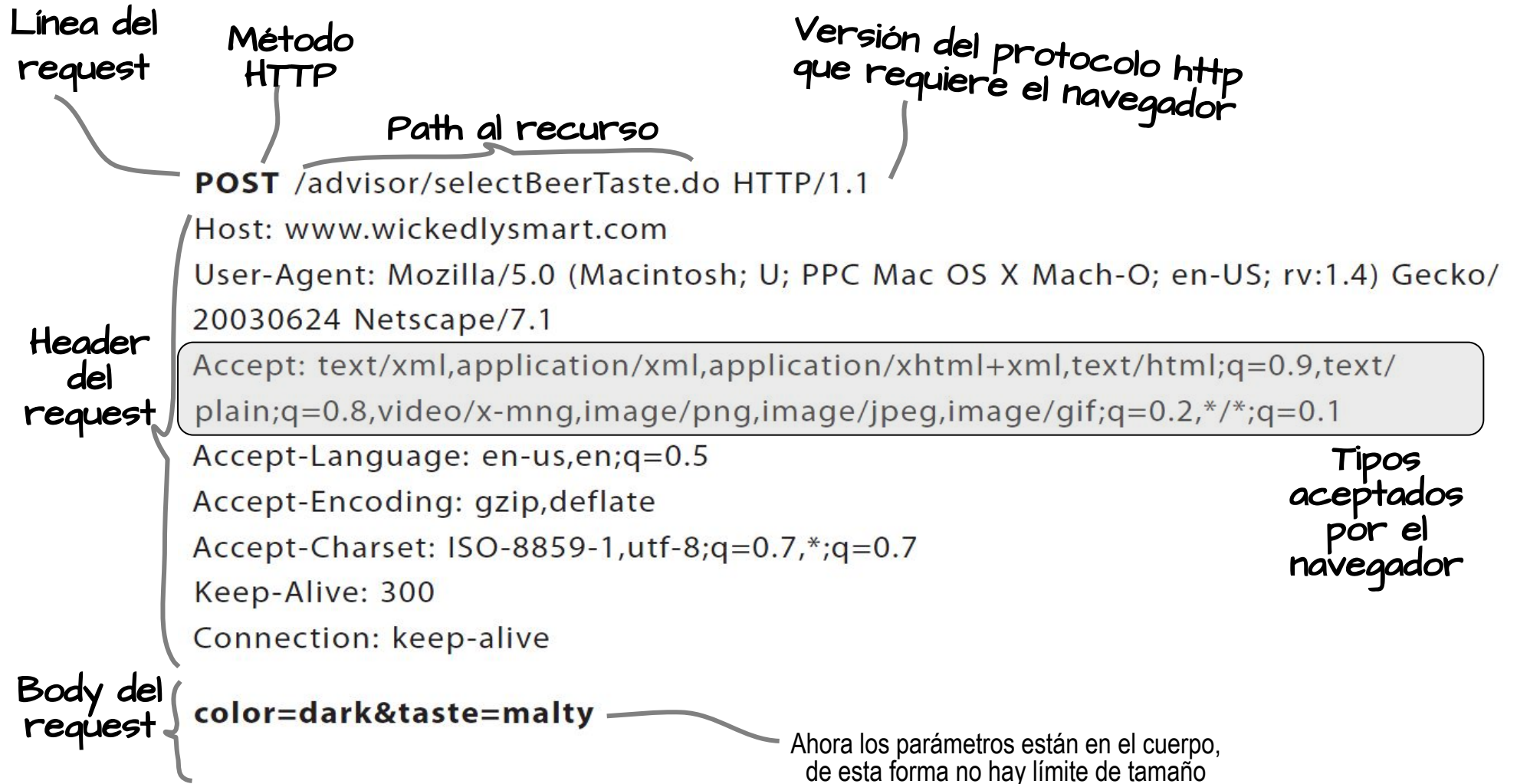
HTTP Request

Anatomía de un requerimiento HTTP GET



HTTP Request

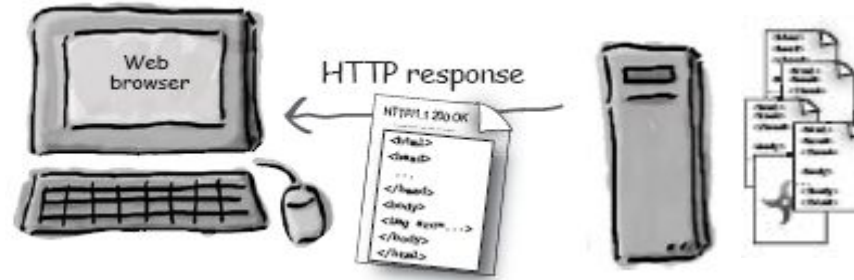
Anatomía de un requerimiento HTTP POST



HTTP Response

¿qué tiene una respuesta HTTP?

Una respuesta HTTP *puede* contener HTML. El HTTP le 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

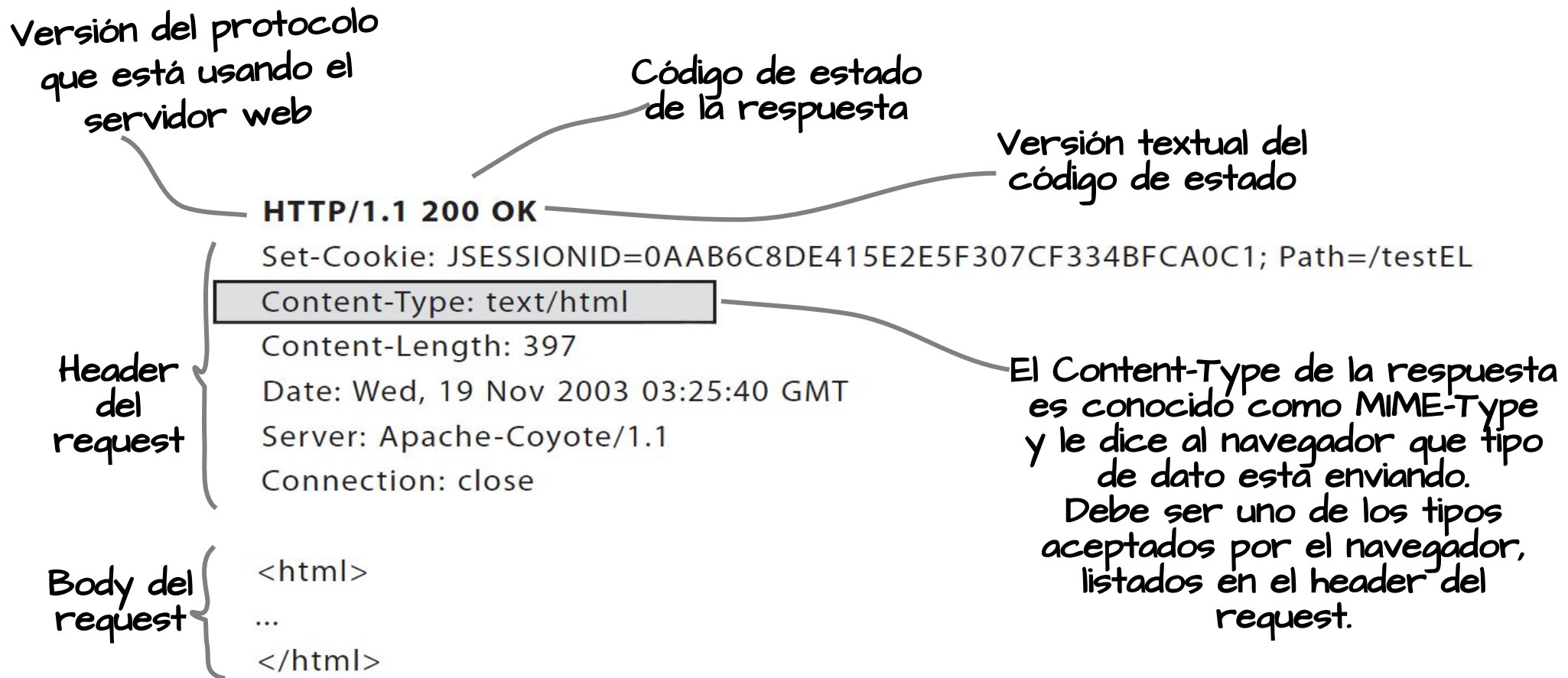
```
<P>
  Esto es un ejemplo de un documento HTML.
</P>

<FORM ACTION="chequearLogin" METHOD="POST">
...
</FORM>
</HTML>
```

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

HTTP Response

Anatomía de un response HTTP

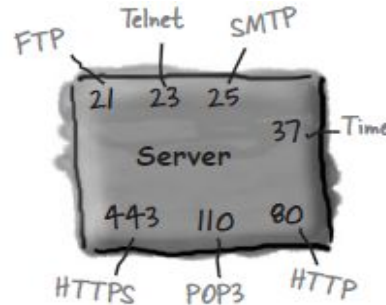


URL: Uniform Resource Locators

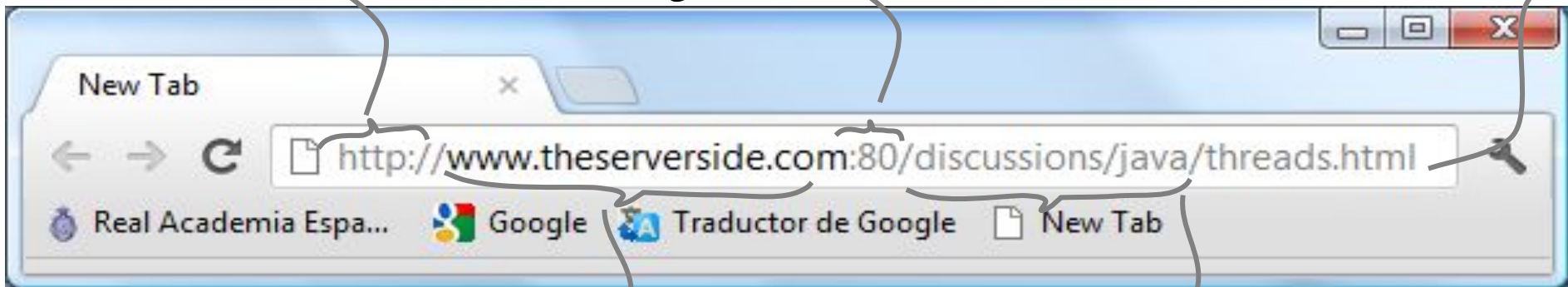
Cada recurso en la web tiene su propia y única dirección en el formato URL.

Protocolo: le indica al servidor que protocolo de comunicación será utilizado

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/>

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 lo llama 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 “submite” un
formulario, crea un parámetro en el
requerimiento para cada campo en el
formulario (FORM)

Login.html

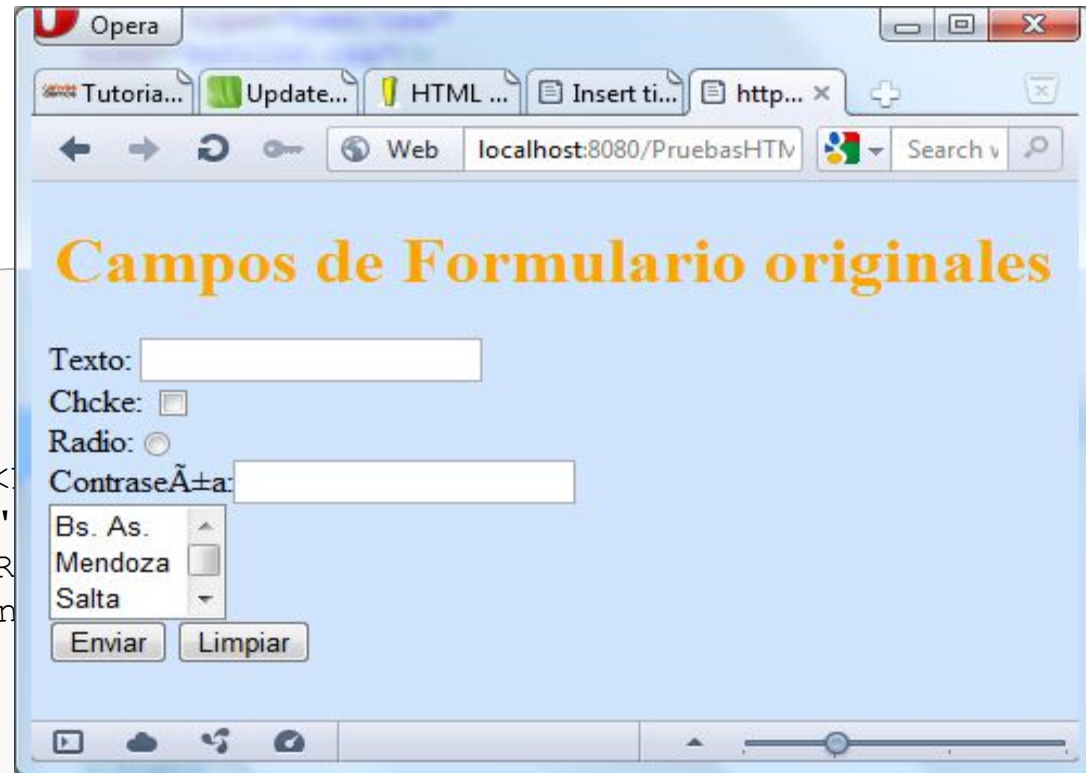
HTML (HyperText Markup Language)

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"></INPUT>
Chcke: <INPUT type="checkbox" name="comp"></INPUT>
Radio:<INPUT type="radio" name="sexo"></INPUT><BR>
Contraseña:<INPUT type="password" name="n"></INPUT>
<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>
```



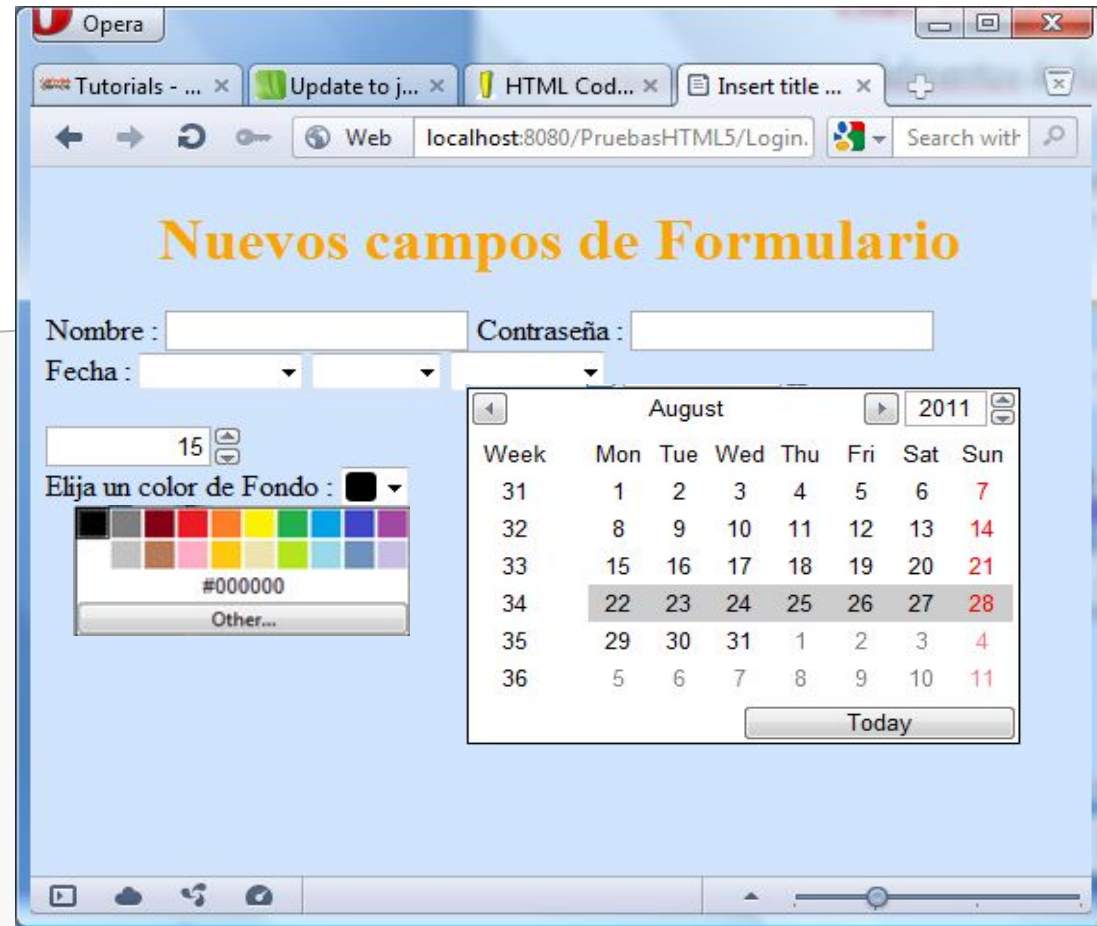
HTML (HyperText Markup Language)

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 del nuevo HTML 5

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

- **<audio>**: Distintos tipos de sonidos, música, transmisión de audio. Los formatos propuestos son: ogg, mp3, wav
- **<video>**: Este tag permite integrar el video con un tag del HTML. Aún no hay una postura definitiva sobre los formatos aceptados.

```
<!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>
```

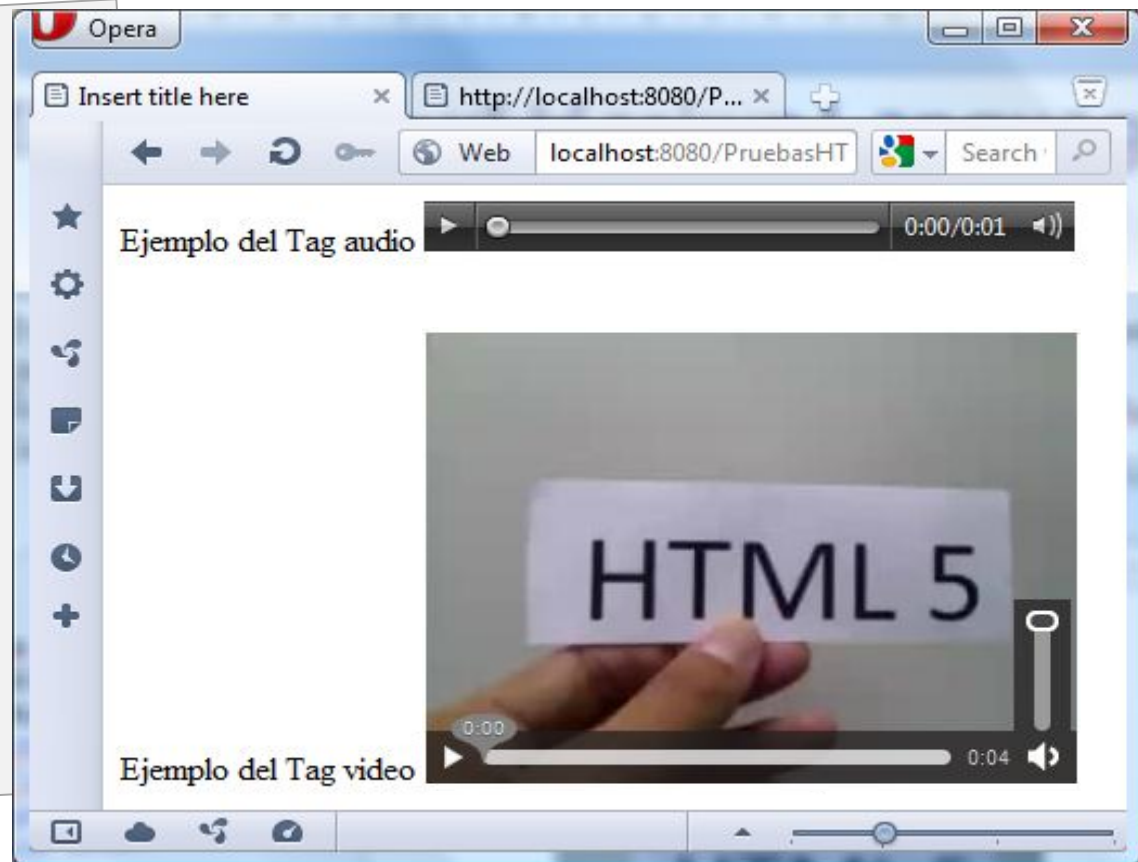
```
<BR>
```

Ejemplo del Tag video

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

```
</body>
```

```
</html>
```

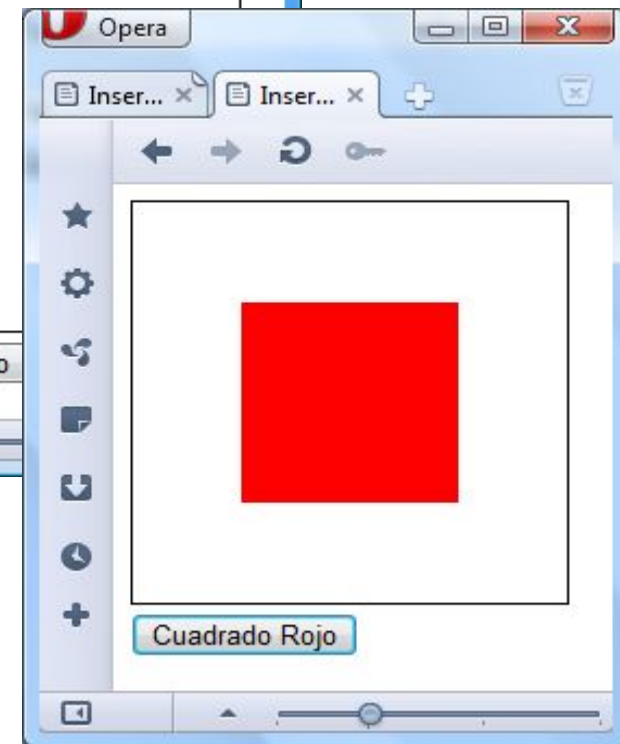
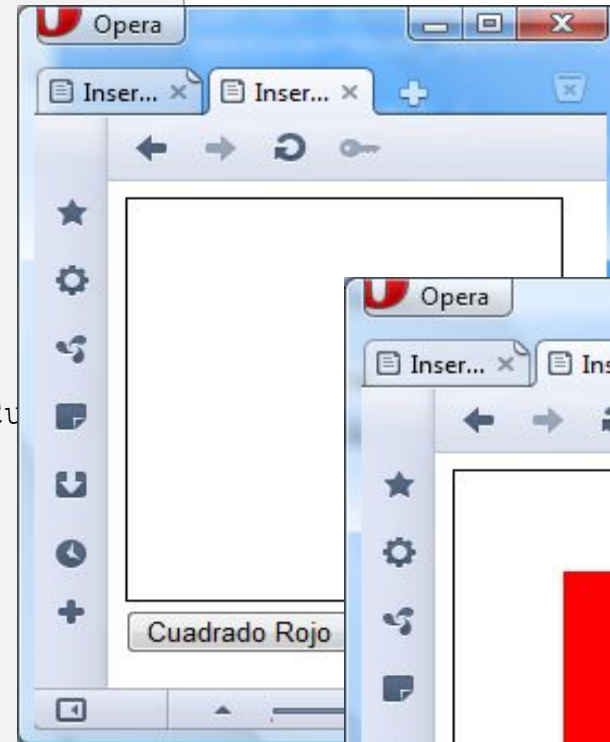


HTML (HyperText Markup Language)

Otras componentes del nuevo HTML 5

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.

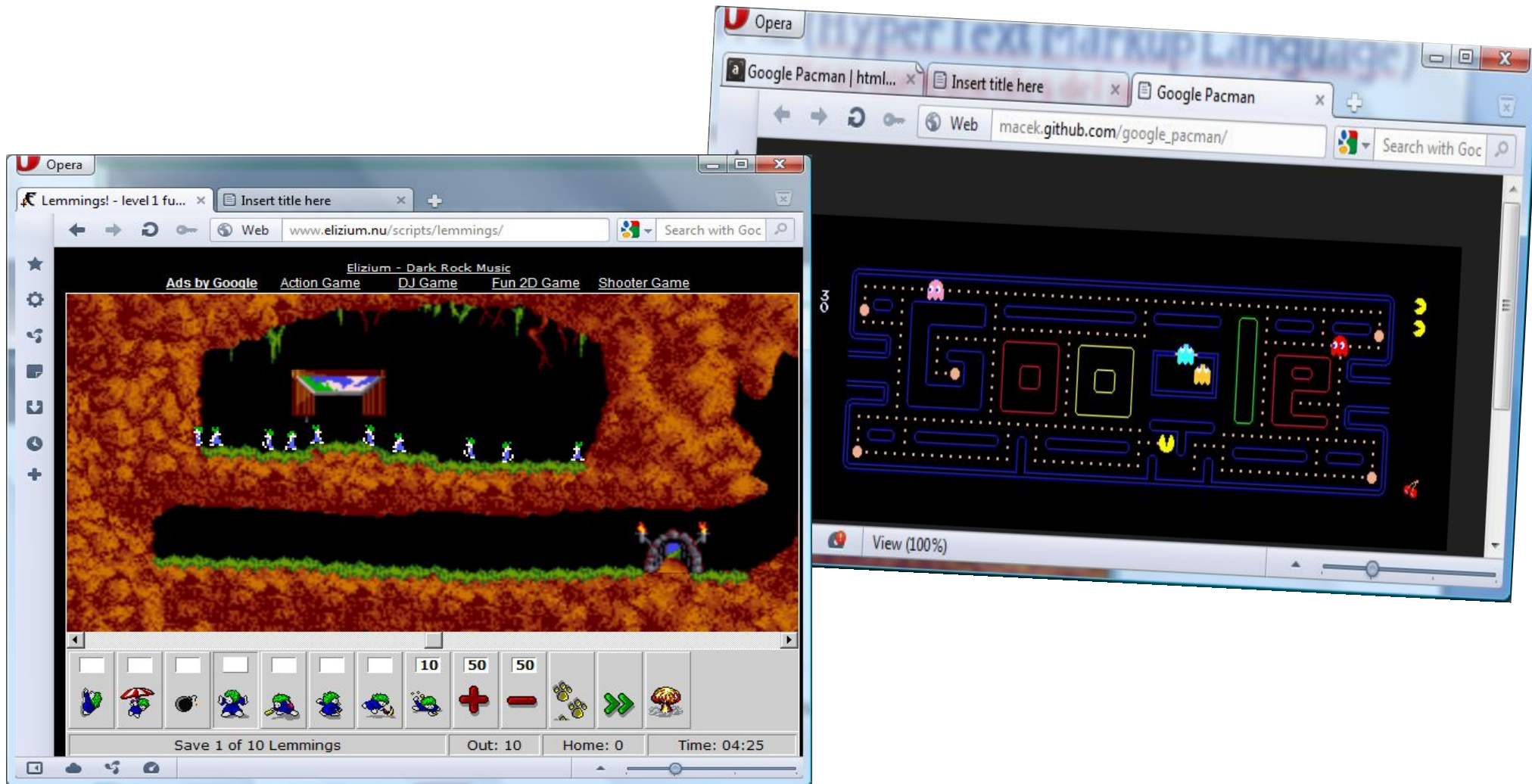
```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<canvas id="c1" width="200" height="200"
style="border:solid 1px #000000;"></canvas>
<button onclick="draw_square();return true;">Cu
Rojo</button>
<script>
function draw_square() {
  var c1 = 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 del nuevo HTML 5

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.



Pueden indicarse pares de propiedad: valor separados por ;

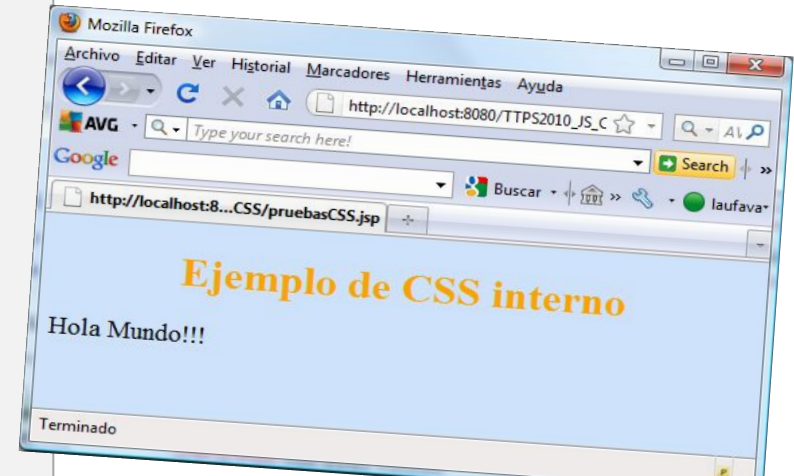
HTML (HyperText Markup Language)

Cascading Style Sheets(CSS)

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

Los tags
<style>,</style>
permiten definir un
estilo embebido

```
<html><head>                                     pruebaCSS.jsp
<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>
```



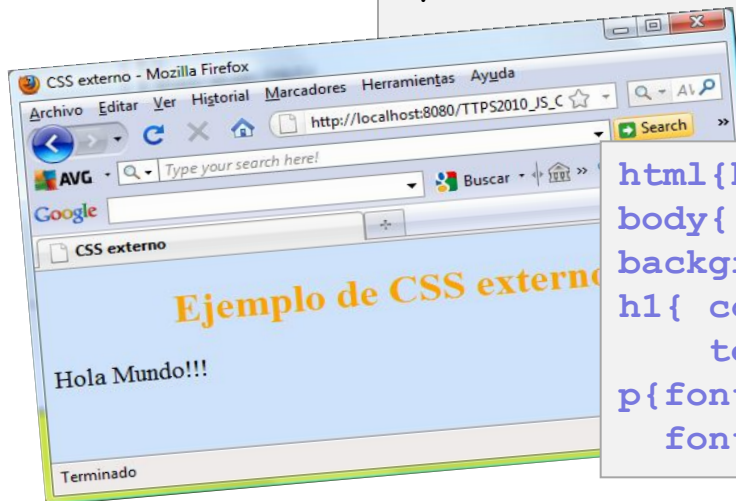
```
<html><head>
<link rel="stylesheet"
      type="text/css"
      href="unEstilo.css"/>
<title>CSS externo</title>
</head>
<body>
<h1>Ejemplo de CSS externo</h1>
<p>Hola Mundo!!!</p>
</body></html>
```

pruebaCSS.jsp

CSS externo

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

unEstilo.css



HTML (HyperText Markup Language)

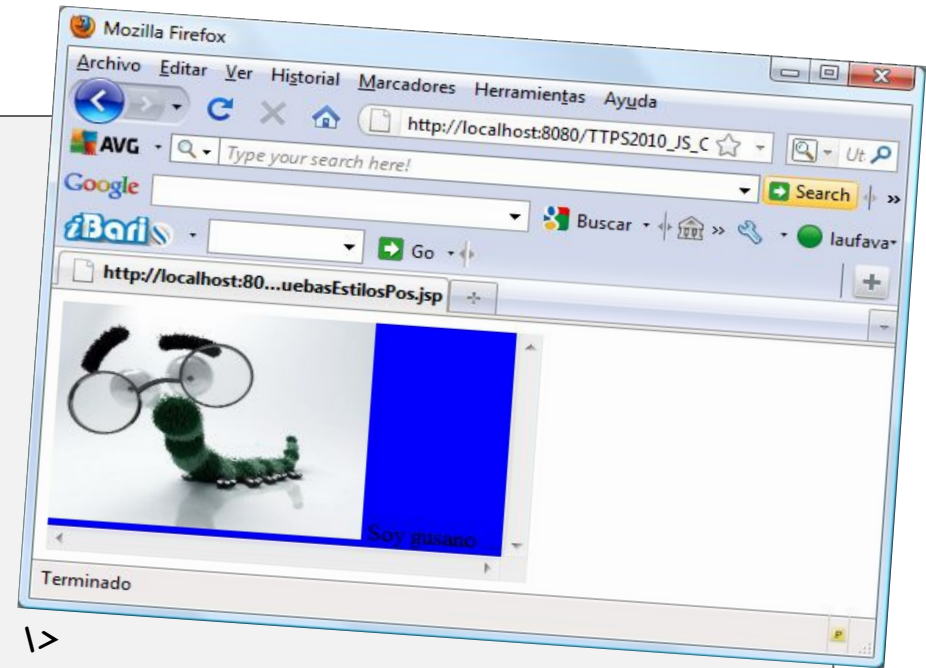
Cascading Style Sheets(CSS)

Es muy común agrupar un conjunto de elementos o tags HTML para aplicarles a todos un mismo estilo. Para ello existe el tag <div> que define una sección en un documento HTML.

También acá se puede ver que se puede definir el estilo directamente en el elemento.

```
<html>
<head>
<script type="text/javascript">
  function imagen() {
    document.getElementById('imagen').src =
      "imagenes/minibuscador.jpg";
  }
</script>
</head>
<body onload="imagen()">
<div style="position:absolute; top:100; left:100;
  text-align:center; background-color: blue;
  overflow:scroll">
  <img id='imagen' height="150" width="200" border="0" \>
    Soy gusano ...
</div>
</body>

</html>
```



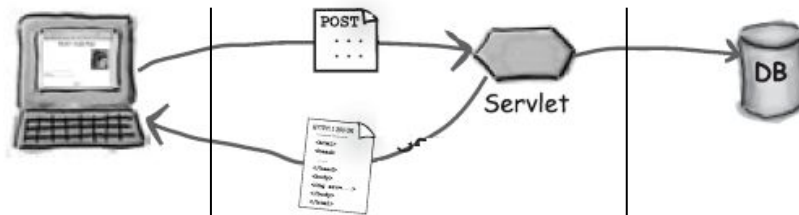
Servlets

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 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 Servlets es la 4.0, está especificada en la JSR 369 perteneciente a la JEE 8

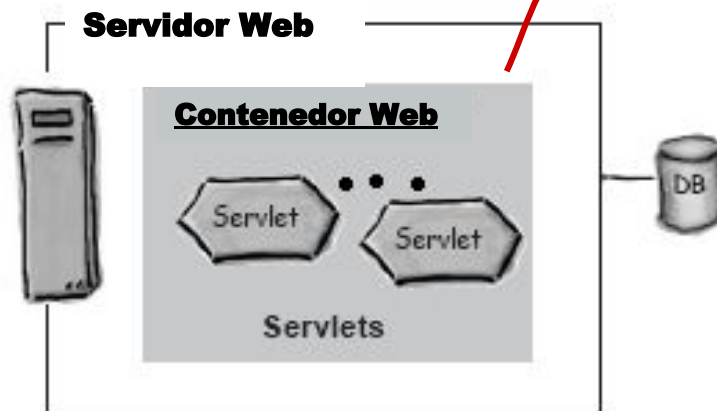
Servlets

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.



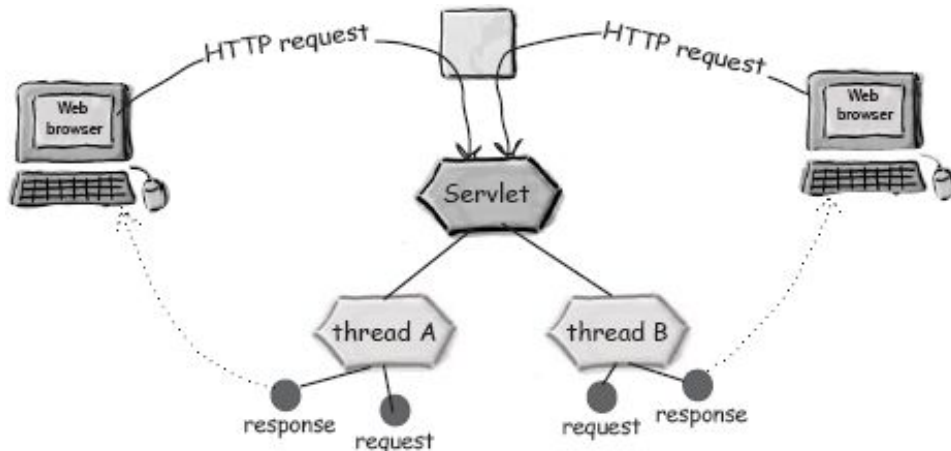
El Contenedor Web interactúa con los servlets invocando métodos de gerenciamiento o métodos callback. Estos métodos definen la interface entre el Contenedor y los servlets(API de servlets).

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

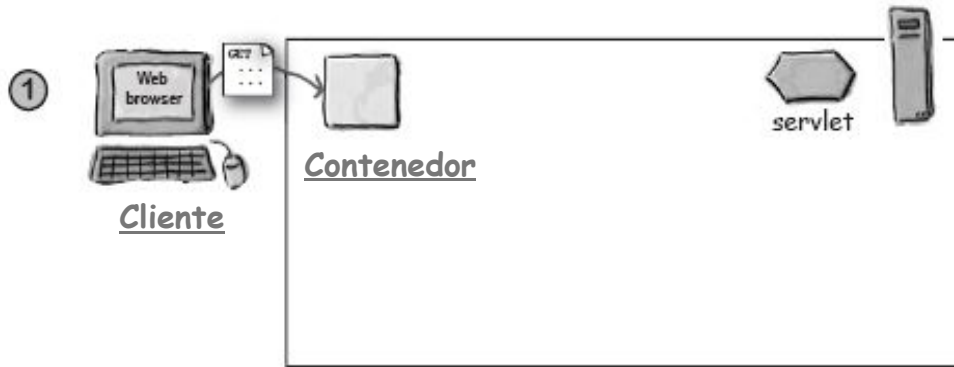
- El Contenedor Web gerencia el ciclo de vida de cada servlet, invocando a 3 métodos definidos en la interface `javax.servlet.Servlet: init(..)`, `service(..)` y `destroy(..)`
- El Contenedor Web, es responsable de la carga e instanciación de los servlets, que puede suceder en el arranque, cuando una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



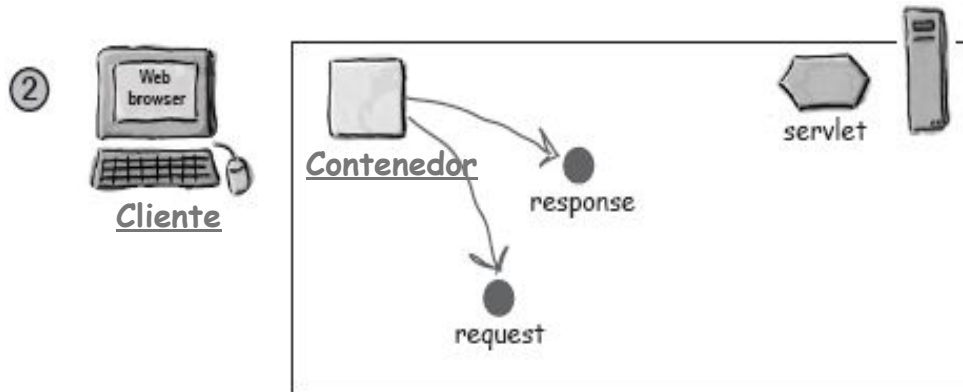
El programador de servlets, debe considerar los efectos colaterales que tiene el procesamiento concurrente y tomar las previsiones necesarias sobre los recursos compartidos (acceso a variables de instancia, variables de clase, recursos externos como archivos, etc.)

Contenedor Web

Los servlet son controlados por el Contenedor



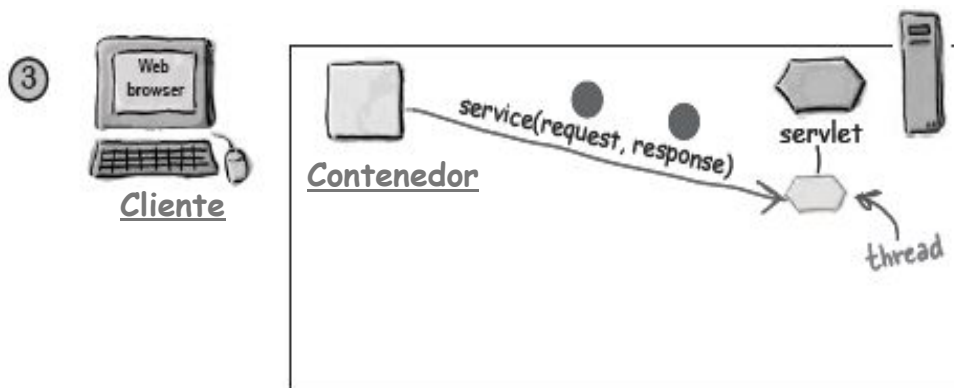
El usuario presiona sobre un link que tiene una URL a un Servlet o se invoca a través de un formulario



El Contenedor crea 2 objetos:

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

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



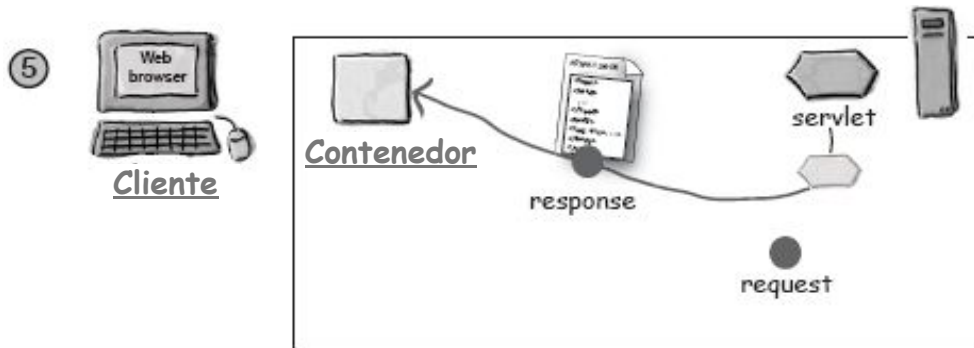
El Contenedor encuentra el servlet correcto usando la URL del requerimiento. Luego crea o aloca un thread para ese requerimiento e invoca al método `service()` pasándole los objetos `request` y `response` como argumento.

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.



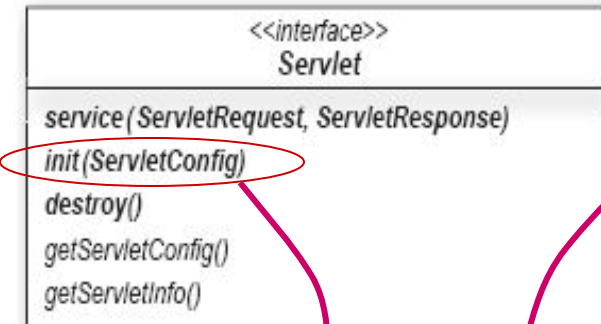
El método `service()` finaliza. El thread termina o retorna al pool de threads del contenedor. El cliente recibe la respuesta.

Servlets

Los servlets heredan los métodos del Ciclo de Vida

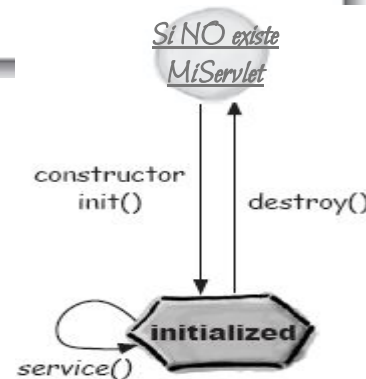
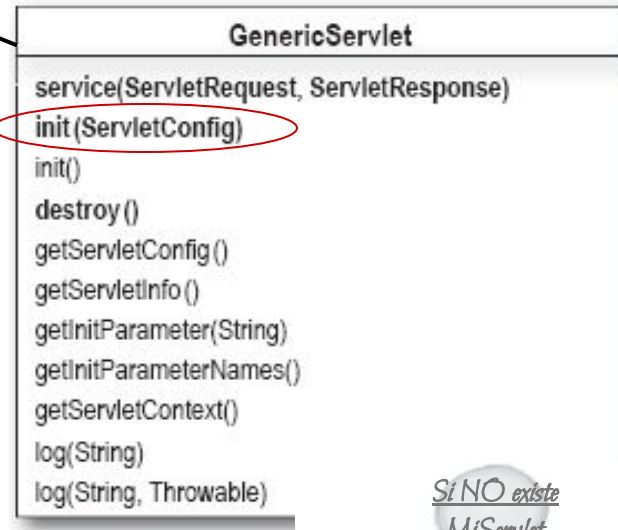
La interface Servlet
(`javax.servlet.Servlet`)

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 sobrescribe para proveer código de inicialización (acceso a DB, guardar objetos en el contexto, etc.)

La interface de programación

La interface Servlet – La clase HttpServlet

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

- **javax.servlet**: contiene la interface 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.

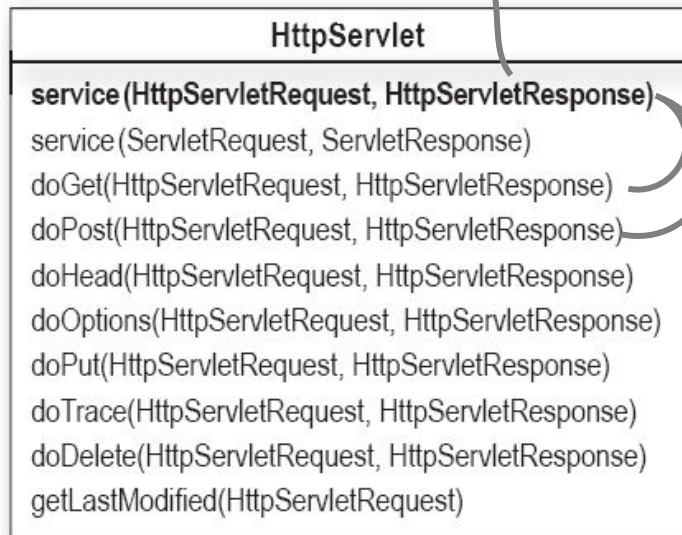
```
public void service(HttpServletRequest req, HttpServletResponse res) throws . . .
```

La interface 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.



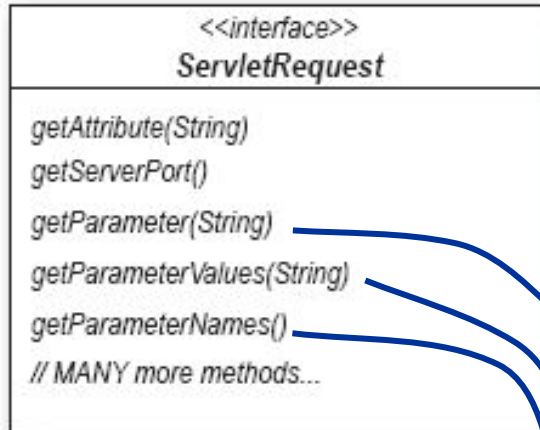
- La secuencia de métodos **service()**-->**doGet()** ó **service()**-->**doPost()** sucede cada vez que un cliente hace un requerimiento.
- Cada vez que un **doGet()** o **doPost()** ejecutan, lo hacen en un thread separado.
- Cuando se escribe un Servlet, los métodos que se sobrescriben son **doGet()**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

La interface 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.

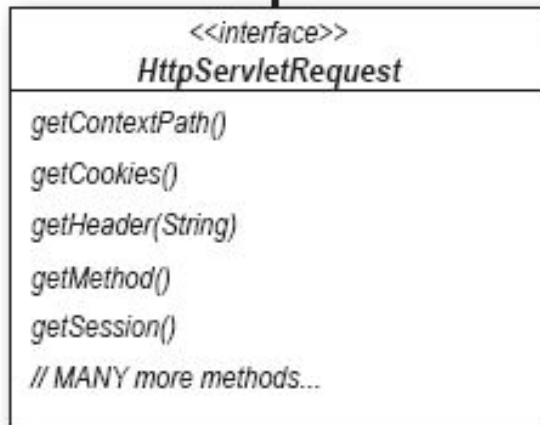
Devuelve un `String` con el valor del parámetro del requerimiento con la clave dada. Si hay múltiples valores para esa parámetro, devuelve el primero.

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 una lista de `Strings` con los nombres de todos los parámetros del requerimiento.

HttpServletRequest interface

(`javax.servlet.http.HttpServletRequest`)

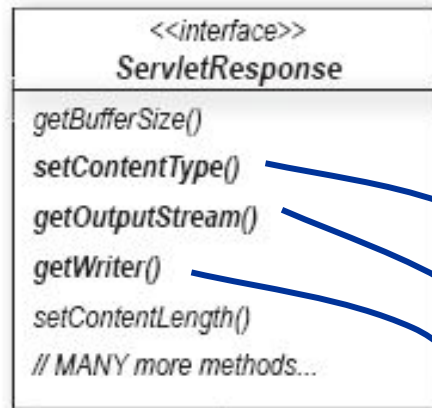


La interface de programación

Las interfaces HttpServletResponse

ServletResponse interface

(javax.servlet.ServletResponse)



HttpServletResponse interface

(javax.servlet.http.HttpServletResponse)



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.

Un ejemplo simple

Un servlet que recupera parámetros del requerimiento

Este Servlet genera una página HTML usando un parámetro del requerimiento

saludo.html

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

HttpServlet, extiende
GenericServlet, la cual
implementa la interface
Servlet

```
package servlets;
public class ServletHola extends javax.servlet.http.HttpServlet {
    protected void doPost(HttpServletRequest request, 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, 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 para mapearlos con una URL o conjunto de URL's respectivamente.

```
<?xml version="1.0" encoding="UTF-8"?>
```

web.xml

```
<web-app ... http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
```

```
<servlet>
```

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

```
  <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>
```

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

```
</web-app>
```

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

URL de la aplicación web

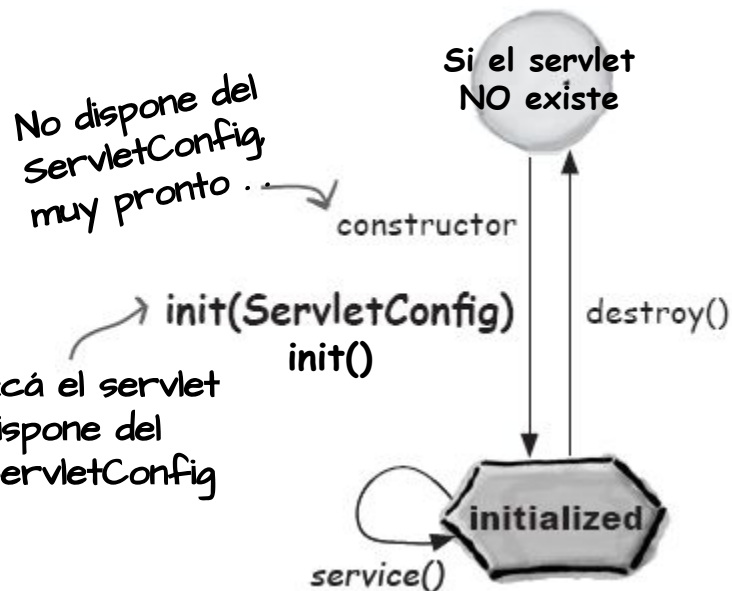
url-pattern o mapping

Parámetros de Inicialización del Servlet

Hasta ahora vimos que mediante los métodos `doGet()` y `doPost()` podemos tomar parámetros del requerimiento, pero además, los servlet pueden recuperar parámetros de inicialización desde el archivo `web.xml`. Cuando el contenedor web inicializa un servlet, crea un objeto `ServletConfig` y se lo pasa al servlet en el método `init()`.

A partir de este objeto se pueden recuperar parámetros desde el archivo `web.xml` usando el método `getInitParameter(String s)`

Para definir parámetros de configuración inicial de un servlet, se usan los sub-elementos `<init-param>`, `<param-name>` y `<param-value>` en el `web.xml`.



```
<servlet>                                     web.xml
  <servlet-name>ServletHola</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>ServletHola</servlet-name>
  <url-pattern>/ServletHola</url-pattern>
</servlet-mapping>
...
```

Parámetros de Inicialización

ServletHola

Este Servlet retorna una página HTML con un mensaje tomado de parámetros inicialización y con la hora/fecha actual.

```
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 podría **no sobrescribirse el método init()** y recuperar los valores de inicialización cuando se quiera usar, de alguna de estas dos maneras:

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

```
<servlet>web.xml
  <servlet-name>ServletHola</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>ServletHola</servlet-name>
  <url-pattern>/ServletHola</url-pattern>
</servlet-mapping>
...
```

Parámetros de Inicialización del Servlet

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

NOTA: es recomendable sobrescribir el **init()** sin parámetros. Si sobrescriben el **init(ServletConfig c)** deben invocar al **init(config)** de la superclase para que el servlet quede bien inicializado!!

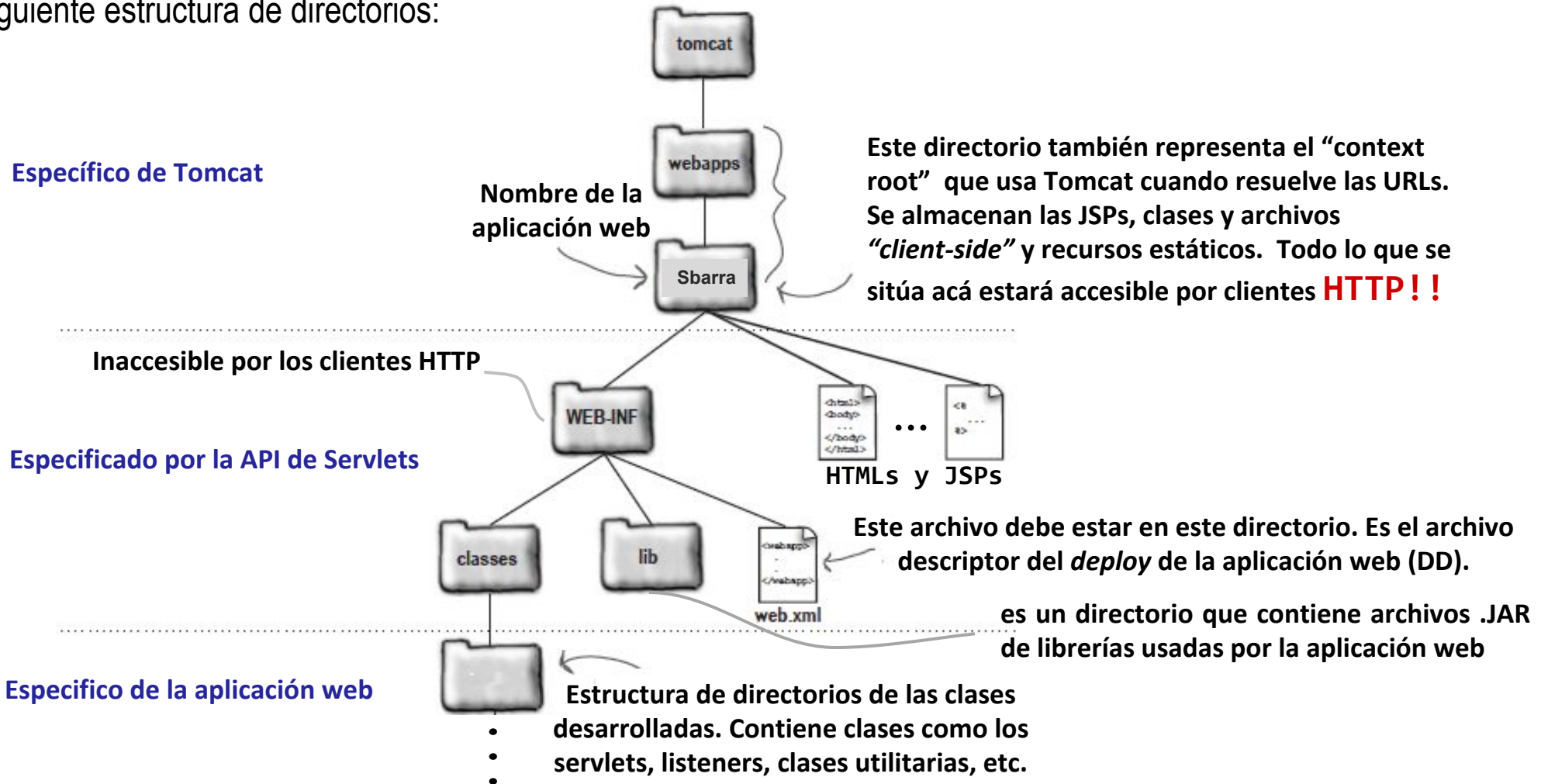
Si se quiere invocar al **getInitParameter()** desde el **doGet()** / **doPost()** y en el **init(ServletConfig config)** sobrescrito no se invocó al **init(ServletConfig)** de la super clase, la ejecución del servlet disparará este error:



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:



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:

- Apache **Tomcat 9**, (implementa las especificaciones Servlet 4.0 y JSP 2.3 del JCP).
- **Eclipse**, IDE para desarrollar aplicaciones Java EE.
- La plataforma estándar, **JSE 11+ (JDK)**