

# **TEMAS SELECTOS DE BASES DE DATOS**

# **TEMAS SELECTOS DE ING. DE SOFTWARE**

## **FACULTAD DE INGENIERÍA**



- Parte 1:
  - Antecedentes: Java
- Parte 2:
  - Acceso a base de datos con JDBC
- Parte 3:
  - Hibernate
- Parte 4:
  - Aplicaciones Web



PARTE 4

# 10. INTRODUCCIÓN AL DESARROLLO DE APLICACIONES WEB CON JAVA



- Introducción al desarrollo web con Java
  - Protocolo HTTP y sus métodos
  - HTML y CSS
  - Formularios HTML
    - *Peticiones HTTP*
    - *ServletRequest*
    - *HttpServletRequest*
  - Ciclo de vida de un servlet
    - *Servlet*
    - *ServletConfig*
    - *GenericServlet*
    - *HttpServlet*





- HTTP es un protocolo simple de petición / respuesta.
- Un cliente es por lo general (No siempre) un web browser.
- Los web browsers o navegadores hacen uso de este protocolo cuando navegamos por el internet.
- Las peticiones enviadas por un cliente web las realiza empleando alguno de los 7 métodos HTTP.
- El cliente recibe una respuesta con el código que indica que sucedió con la petición y así como los datos que fueron solicitados.



# ¿Cómo funciona una petición http?

- Las reglas fueron definidas por el World Wide Web consortium.
- Una petición consiste de:
  - Una línea que contiene la petición
    - *Método HTTP*
    - *Destino de la petición (URI)*
    - *La versión del protocolo HTTP*
  - Encabezado
  - Cuerpo de la petición o mensaje (opcional).



- Request line:
  - GET http://www.osborne.com/index.html HTTP/1.1
- Header
  - Accept: image/\*, application/vnd.ms-excel, \*/\*
  - Accept-Language: en-gb
  - Accept-Encoding: gzip, deflate
  - User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
  - Host: www.osborne.com
  - Connection: Keep-Alive
- Body
  - [BLANK LINE]
  - "<html><head>. . .</html>



- GET
- POST
- HEAD
- OPTIONS
- PUT
- DELETE
- TRACE





- GET

- Es el método mas básico, se emplea para solicitar información (solo lectura) al servidor
- Considerado idempotent (al enviar n veces la petición al servidor se obtienen resultados idénticos, no se realiza ningún cambio del lado del servidor).
- Existen 3 formas básicas para generar una petición empleando el método GET:
  - *Escribiendo el URL en el navegador*
  - *Haciendo clic en una liga*
  - *Presionando el botón enviar en una forma HTML.*
- Los parámetros se envían en el url:
  - **`http://dictionary.com/search?q=java&version=14`**



- Se emplea cuando se desea actualizar información en el servidor, típicamente en una base de datos.
- Los parámetros no se envían en el URL, sino en el cuerpo de la petición. Beneficios:
  - Oculta los parámetros de envío
  - El URL tiene una longitud limitada.
- Típicamente para enviar una petición POST al servidor se emplea un formulario HTML al presionar el botón «enviar»



- HTML es un lenguaje de programación sencillo pensado para presentar información en la red.
- HTML (HyperText Markup Language), como su nombre indica es un lenguaje de marcas para la creación de hipertextos.
- Estas marcas representan fragmentos de texto destacado de una forma especial que permiten la definición de las distintas instrucciones, tanto los efectos a aplicar sobre el texto como las distintas estructuras del lenguaje.
- A estas marcas se le conocen como etiquetas.



- Todo texto que se encuentre entre los caracteres `<` y `>` se considerará una etiqueta.
- Si la etiqueta es inválida del lenguaje HTML no se tomará en cuenta, sin causar ningún tipo de error, dejándose el texto o las etiquetas a las que afectaba como si no existiera la etiqueta extraña.
- Cuando se comete un error sintáctico al expresar una etiqueta o un atributo no se produce ningún error, simplemente no se obtendrá el efecto que deseábamos.





- Una etiqueta es un texto incluido entre los símbolos menor que < y mayor que >. El texto incluido dentro de los símbolos será explicativo de la utilidad de la etiqueta. Por ejemplo:
  - **<B>Texto que será en negrita</B>**
- Las etiquetas pueden presentar modificadores que llamaremos atributos que permitirán definir diferentes posibilidades de la instrucción HTML. Ejemplo:

**<a href="http://www.synergyj.com">Pagina principal</a>**



- Representan un contenedor de elementos gráficos empleados para capturar o seleccionar información en una pagina web.
- Se representa por el elemento <form>
- Principales atributos:
  - action: recurso a solicitar
  - method: el método HTTP a ejecutar.
- Ejemplo:

```
<form action = "http://localhost:8080/someServlet" method = "post">  
.... Elementos html ...  
</form>
```



- ¿Cómo enviar los datos que el usuario de la pagina web ha capturado?
- Este tipo de acciones se logran presionando algun botón de envió del formulario.
- Un botón es representado por una etiqueta `<input>` y existen tres tipos:

```
<input type="submit" name= "formSubmission" value="Send Details" />
```

```
<input type="reset" value="Resetea Campos" />
```

```
<input type="button" value= "Actualiza campo oculto" onclick= "ActualizaCampoOculto();" />
```



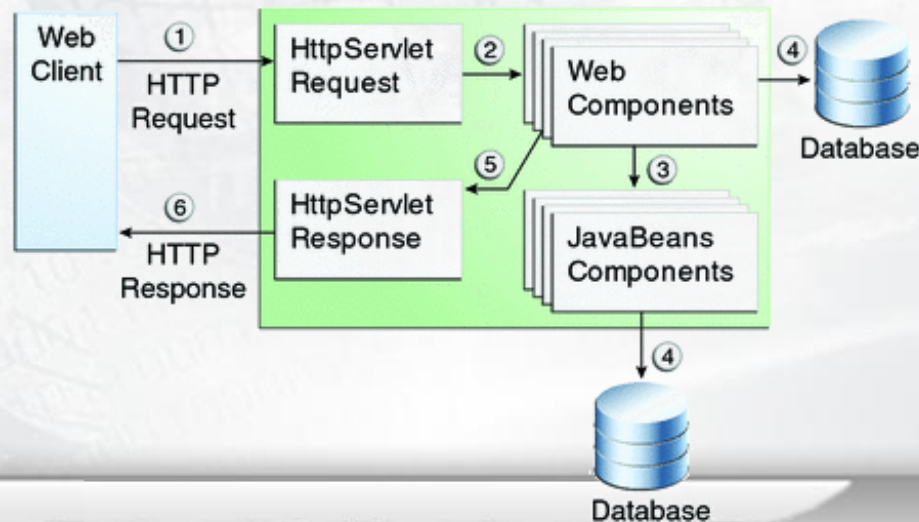
- Existe una gran variedad de servidores web que entienden el protocolo HTTP.
- Estos servidores manejan únicamente contenido estático. Carecen de manejo dinámico.
- Surgimiento de tecnologías para el manejo de contenido dinámico: CGI, FastCGI, ASP, PHP, Perl, ColdFusion.
- Principales problemas con tecnologías como CGI: Performance, esfuerzo considerable para desarrollo, etc.
- En cuanto a la plataforma Java, existe toda una especificación que dicta la forma y las reglas para desarrollar aplicaciones web escritas en **Java**.
- La especificación forma parte del estándar JEE.





# JEE y el manejo de peticiones HTTP

- Las peticiones HTTP se representan a través de la interface **HttpServletRequest**.
- La respuesta a enviar al cliente se representa por la interface **HttpServletResponse**.
- Los servidores exponen componentes para poder recibir este tipo de peticiones a través de los **Servlets**
- A todos estos elementos se es conoce como **componentes web**.



- Contiene una JVM.
- Surge el concepto de **Servlet**:
  - Programa escrito en Java con capacidades para entender el protocolo HTTP.
  - Genera contenido dinámico y envía la respuesta al cliente.
  - Capacidad de atender múltiples peticiones concurrentes.
- A estos servidores comunmente se es conoce como contenedores web:
  - Tomcat (apache), Glassfish (oracle), Weblogic (Oracle), JBoss (JBoss), Web Sphere (IBM), etc.

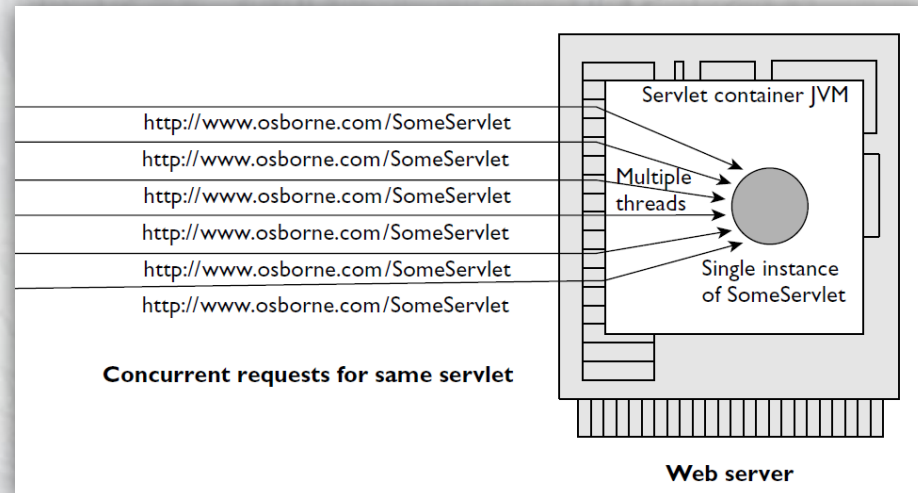


- Las instancias de un Servlet son creadas y administradas por un contenedor.
- Las instancias viven dentro del contexto de un contenedor web.
- Un contenedor de web es un programa que sirve para dar vida a los servlets.
- Un contenedor nos ofrece:
  - Soporte de comunicación.
  - Manejo del ciclo de vida de un servlet.
  - Soporte de Multithreading.
  - Seguridad declarativa.
  - Soporte para JSPs.



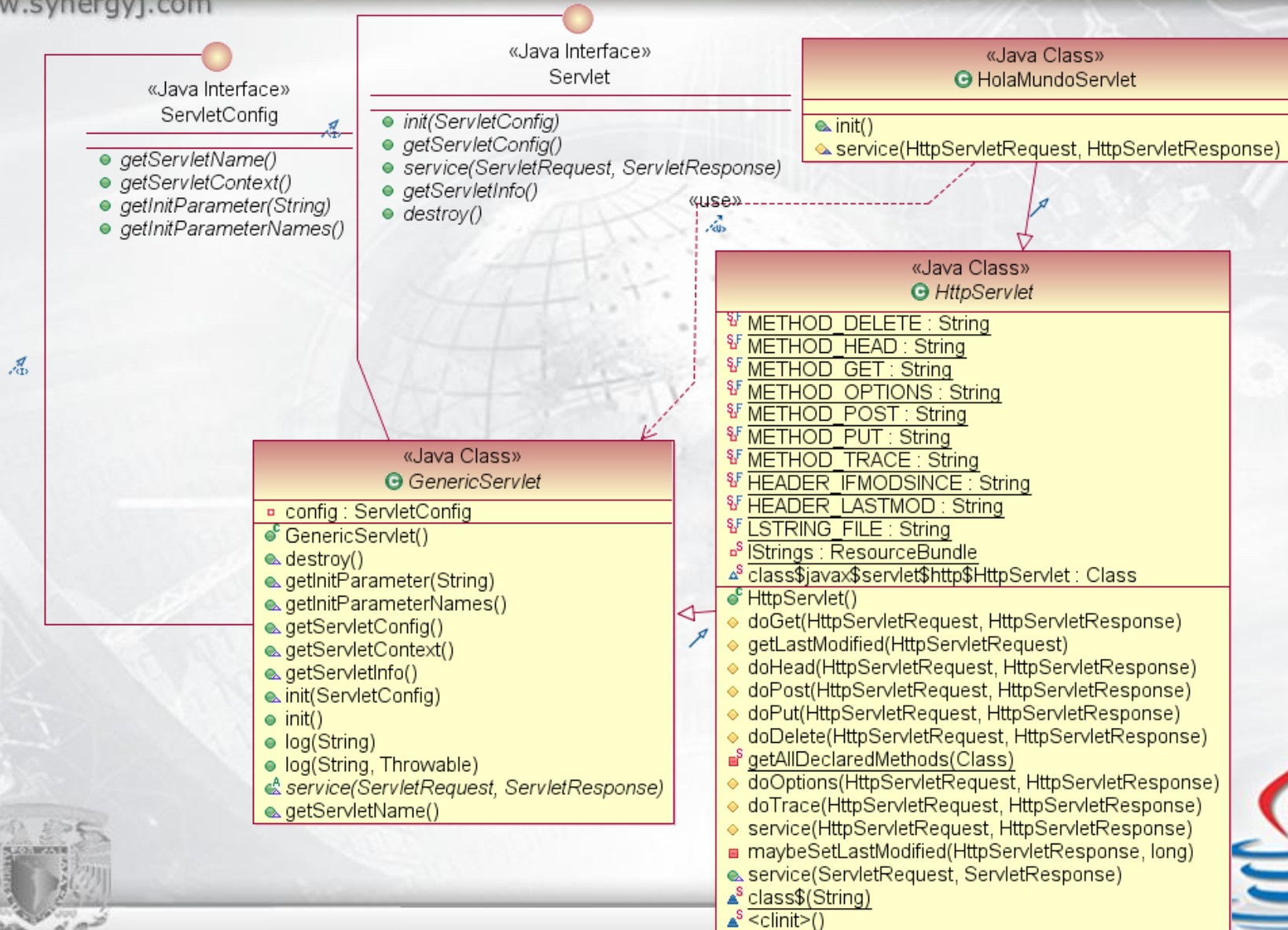
# Contenedor de Servlets (cont).

- Interacción con peticiones:
  - El cliente realiza una petición hacia un recurso en el servidor
  - El contenedor verifica si la petición es para un servlet y crea 2 objetos
    - *HttpServletRequest*
    - *HttpServletResponse*
  - El contenedor localiza al servlet solicitado, genera un hilo de ejecución e invoca los métodos del servlet para atender la petición.





# Ciclo de vida de un Servlet: Jerarquía



- ¿Qué sucede con los servlets al recibir una petición HTTP ?
- Por cada método HTTP, el servlet define un método doXXX encargado de atender las peticiones:
  - doGet
  - doPost
  - doPut
  - doDelete, etc.
- Todos estos métodos definen 2 parámetros:
  - Petición (javax.servlet.HttpServletRequest)
  - Respuesta (javax.servlet.HttpServletResponse).

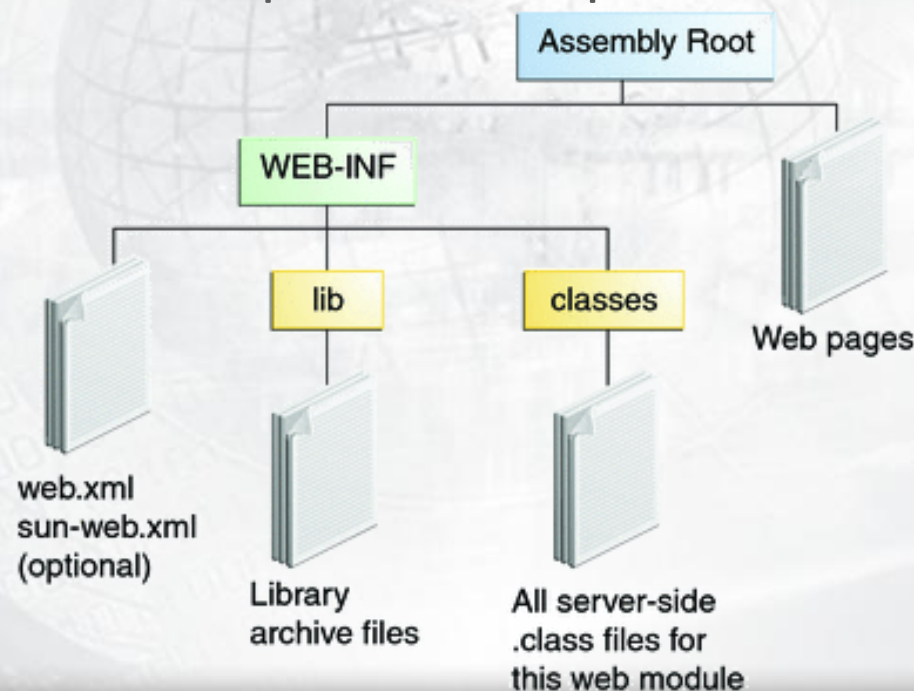


- Continuando con la historia..
  - El método **service** determina el metodo HTTP asociado e invoca a **doXXX**
  - El servlet usa la respuesta (**HttpServletResponse**) para escribir la respuesta al cliente a través del contenedor.
  - El método **service** termina el hilo muere o se regresa al pool, la petición y la respuesta salen del ámbito y están listas para ser recolectadas por el garbage colector.
  - El cliente obtiene su respuesta.





- En Java, todas las aplicaciones web se representan a través de archivos con extensión .war (web application)
- Un archivo war es similar a un archivo jar , pero con una estructura específica como se muestra en la figura.
- El archivo web.xml es un archivo xml que contiene toda la configuración de la aplicación. A partir de JEE 5 es opcional.





- Tomcat es un contenedor web muy ligero y sencillo de usar para iniciar con el desarrollo de aplicaciones web escritas en Java. Para instalarlo seguir las siguientes instrucciones:
  - De la pagina <http://tomcat.apache.org/download-70.cgi> obtener el archivo zip, o tar.gz dependiendo del sistema operativo en su versión 7.0.14 o la mas reciente que se encuentre Dentro de la sección Binary Distributions.
  - Extraer el contenido del archivo en cualquier carpeta (en windows se recomienda en c:/ y en linux en /opt).
  - Asegurarse de configurar la variable de entorno JAVA\_HOME que apunte al directorio de instalación del JDK, por ejemplo:
    - `export JAVA_HOME=/opt/jdk1.6.0_17`
  - Ejecutar, de preferencia a línea de comandos, el archivo startup.sh o startup.bat , pro ejemplo:
    - `sh startup.sh`



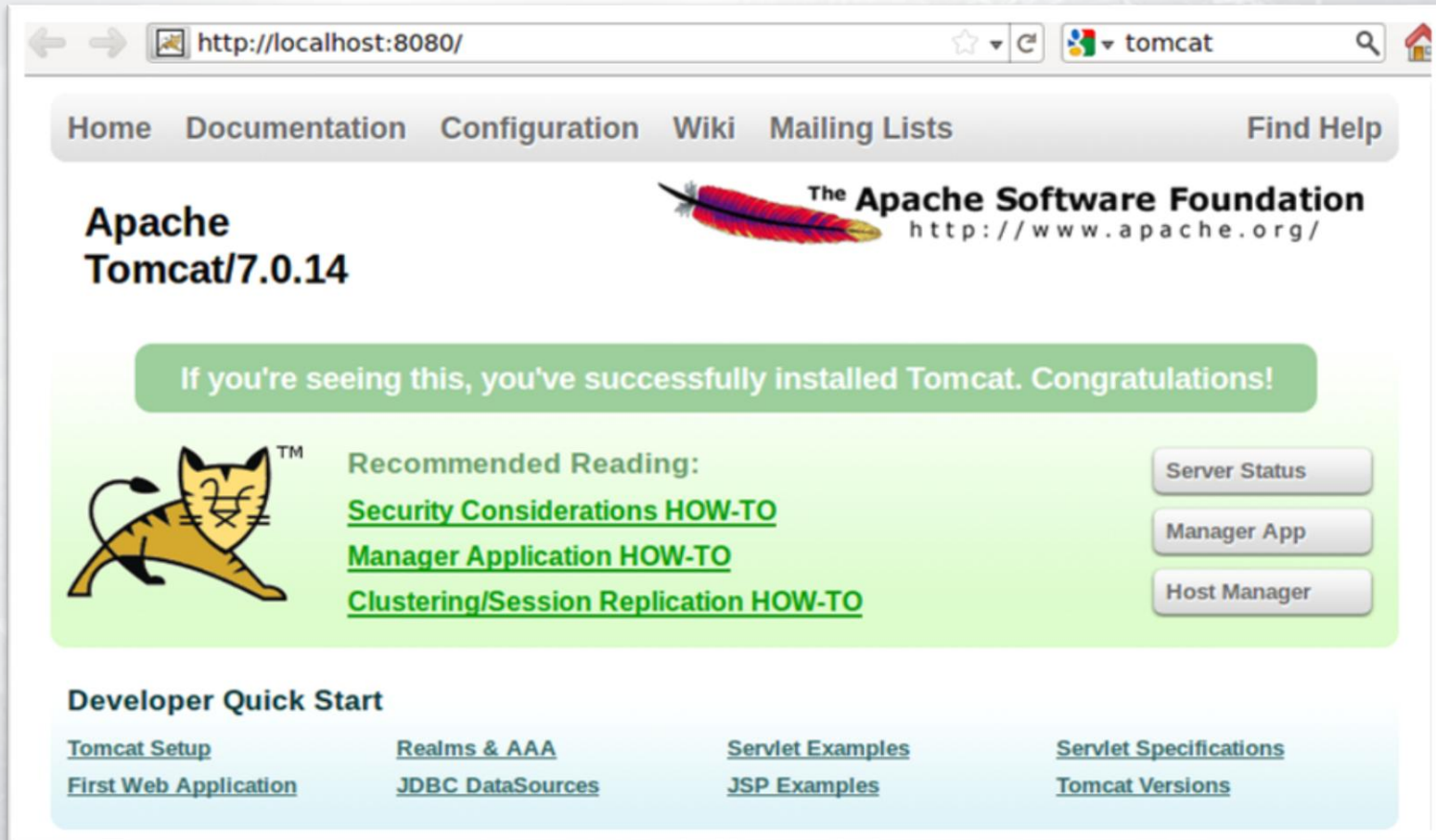
- Al ejecutar la instrucción anterior, se obtiene algo como lo siguiente:

```
jorge@lap-xps:/opt/apache-tomcat-7.0.14$ cd bin/  
jorge@lap-xps:/opt/apache-tomcat-7.0.14/bin$ sh startup.sh  
Using CATALINA_BASE:   /opt/apache-tomcat-7.0.14  
Using CATALINA_HOME:   /opt/apache-tomcat-7.0.14  
Using CATALINA_TMPDIR: /opt/apache-tomcat-7.0.14/temp  
Using JRE_HOME:        /opt/jdk1.6.0_17  
Using CLASSPATH:       /opt/apache-tomcat-  
7.0.14/bin/bootstrap.jar:/opt/apache-tomcat-7.0.14/bin/tomcat-juli.jar
```

- Para comprobar que el servidor inicio correctamente, abrir un navegador web con la siguiente url, aparecerá la siguiente pantalla:

– <http://localhost:8080>






The screenshot shows a web browser window with the address bar set to `http://localhost:8080/`. The browser's address bar includes navigation buttons, a star icon, a refresh icon, a tab labeled 'tomcat', and a search icon. The page content features a navigation bar with links: Home, Documentation, Configuration, Wiki, Mailing Lists, and Find Help. Below this is the Apache Software Foundation logo and the text 'The Apache Software Foundation' with the URL `http://www.apache.org/`. The main heading reads 'Apache Tomcat/7.0.14'. A green banner states: 'If you're seeing this, you've successfully installed Tomcat. Congratulations!'. To the left of the 'Recommended Reading' section is the Tomcat logo (a stylized orange cat). The 'Recommended Reading' section lists three links: [Security Considerations HOW-TO](#), [Manager Application HOW-TO](#), and [Clustering/Session Replication HOW-TO](#). To the right of these links are three buttons: 'Server Status', 'Manager App', and 'Host Manager'. Below the recommended reading is a 'Developer Quick Start' section with a grid of links: [Tomcat Setup](#), [Realms & AAA](#), [Servlet Examples](#), [Servlet Specifications](#), [First Web Application](#), [JDBC DataSources](#), [JSP Examples](#), and [Tomcat Versions](#).

Home Documentation Configuration Wiki Mailing Lists Find Help

The Apache Software Foundation  
<http://www.apache.org/>

## Apache Tomcat/7.0.14

If you're seeing this, you've successfully installed Tomcat. Congratulations!



Recommended Reading:

- [Security Considerations HOW-TO](#)
- [Manager Application HOW-TO](#)
- [Clustering/Session Replication HOW-TO](#)

Server Status  
Manager App  
Host Manager

### Developer Quick Start

<a href="#">Tomcat Setup</a>	<a href="#">Realms &amp; AAA</a>	<a href="#">Servlet Examples</a>	<a href="#">Servlet Specifications</a>
<a href="#">First Web Application</a>	<a href="#">JDBC DataSources</a>	<a href="#">JSP Examples</a>	<a href="#">Tomcat Versions</a>





- El siguiente paso es crear una aplicación web que haga uso de un Servlet sencillo que imprima el mensaje «Hola Mundo»
- Para realizar lo anterior, se empleará eclipse empleando la extensión para JEE. Es posible emplear NetBeans. Las siguientes instrucciones aplican para Eclipse:
  - Dentro de la pagina <http://www.eclipse.org/downloads/> seleccionar Eclipse IDE for Java EE Developers para el sistema operativo correspondiente.



**Eclipse IDE for Java EE Developers**, 206 MB  
Downloaded 985,066 Times [Details](#)



**Windows 32 Bit**  
**Windows 64 Bit**






- Extraer el contenido en cualquier directorio y ejecutar el archivo llamado eclipse.
- Se abrirá la interfaz gráfica muy similar a la empleada en temas anteriores.
- Para crear un módulo web:
  - Del menú file seleccionar new -> Dynamic web project
  - Configurar los valores como aparecen en la siguiente pantalla:



- La ubicación puede variar, seleccionar siguiente.

**Dynamic Web Project** 

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location

☐ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

EAR project name:

Working sets

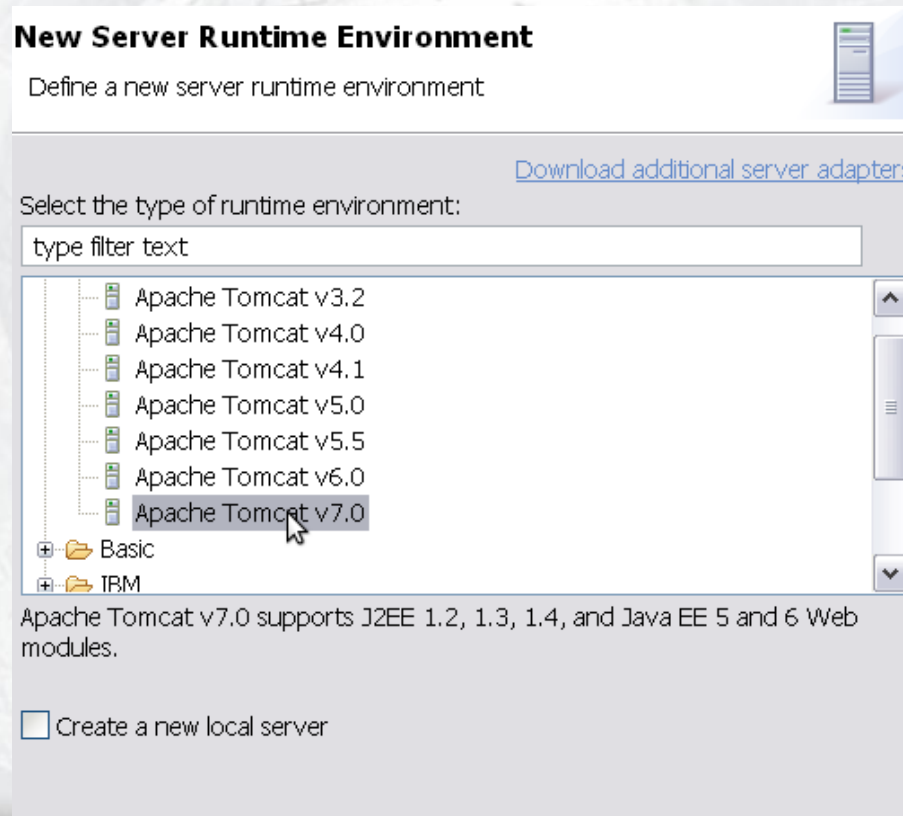
☐ Add project to working sets

Working sets:



# Creando un proyecto web (cont.)

- Para seleccionar el valor del campo target runtime, presionar el botón New Runtime..
- Seleccionar Tomcat 7 como se observa en la figura.



# Creando un proyecto web (cont.)

- Al seleccionar el servidor, se deberá indicar la ruta donde esta instalado el servidor Tomcat.

**Tomcat Server**

Specify the installation directory

Name:  
Apache Tomcat v7.0

Tomcat installation directory:  
C:\desarrollo\apache-tomcat-7.0.14

JRE:  
Workbench default JRE





# Creando un proyecto web (cont.)

- Continuando con la configuración, dejar los valores por default, presionar siguiente..

**Java**  
Configure project for building a Java application.

Source folders on build path:

src

Add Folder...  
Edit...  
Remove

Default output folder:  
build/classes



# Creando un proyecto web (cont.)

- En esta pantalla se configura el directorio dentro del proyecto donde se guardarán las páginas html, jsps, imágenes, etc. Dejar los valores por default y presionar finish.
- Observar que en este punto se especifica la generación del archivo web.xml, para este ejemplo no es requerido, sin embargo marcar la opción para futuros ejemplos.
- El context root es una cadena que se escribirá en el navegador para identificar a la aplicación web (es una especie de directorio virtual, no necesariamente tiene que existir).

## Web Module

Configure web module settings.



Context root: ejemplos-web

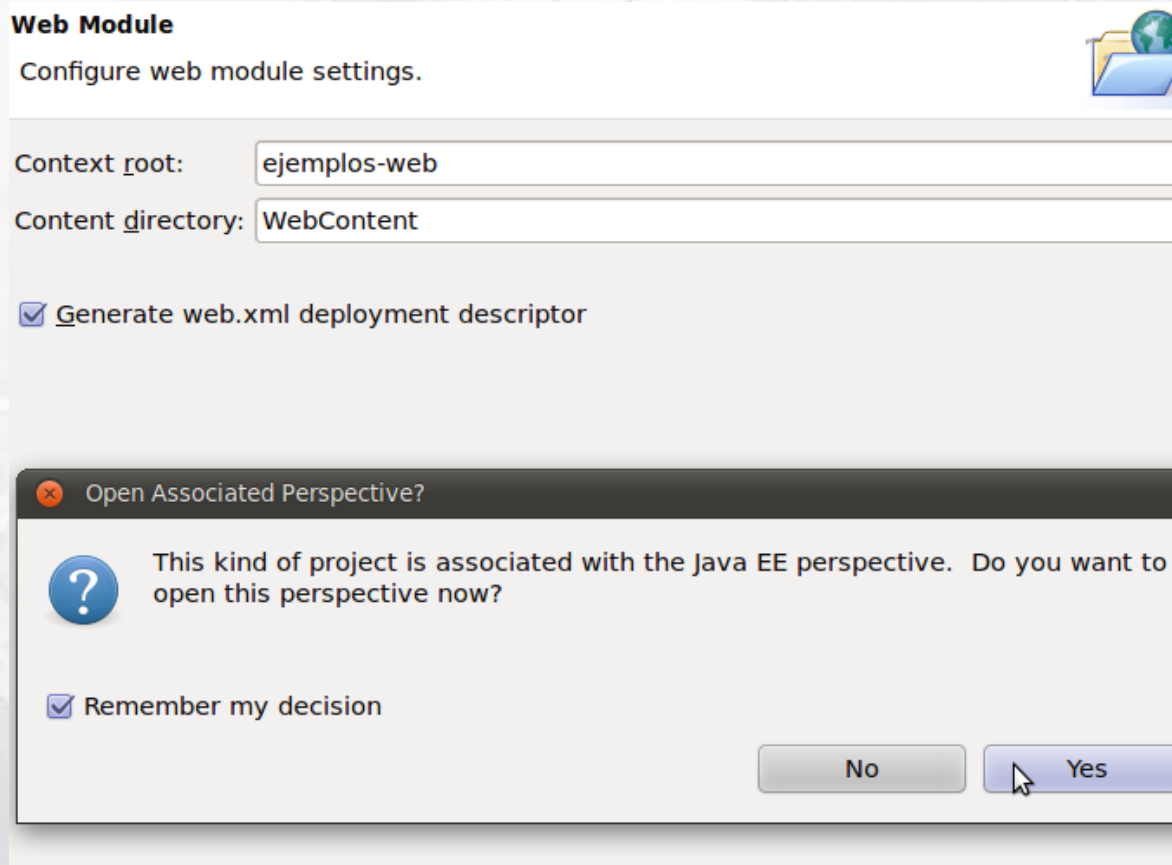
Content directory: WebContent

☒ Generate web.xml deployment descriptor



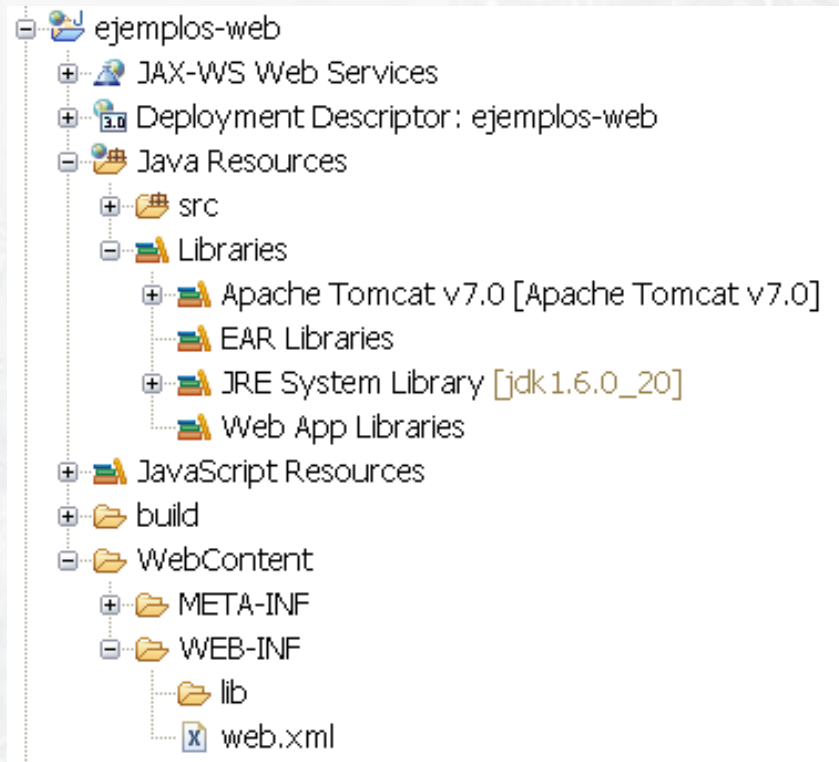
# Creando un proyecto web (cont.)

- Eclipse preguntará si desea emplear la perspectiva para desarrollo JEE, presionar Yes.



# Creando un proyecto web (cont.)

- Al terminar, la estructura del proyecto será la siguiente:



- El siguiente paso es crear un nuevo paquete:  
`mx.edu.unam.fi.temas.web.servlets`





# Creando al servlet HolaMundoServlet

- Haciendo clic derecho sobre el paquete creado, seleccionar la opción new -> Servlet, llenar los valores de los campos como se indica:

6 <display-name>ejemplos-web</display-name>  
7 <welcome-file-list>  
8 <welcome-file>index.html</welcome-file>

**Create Servlet**  
Specify class file destination.

Project: ejemplos-web

Source folder: /ejemplos-web/src Browse...

Java package: mx.edu.unam.fi.temas.web.servlets Browse...

Class name: HolaMundoServlet

Superclass: javax.servlet.http.HttpServlet Browse...


☐ Use an existing Servlet class or JSP

Class name: HolaMundoServlet Browse...



## Creando al servlet HolaMundoServlet (cont.)

- En la siguiente pantalla dejar los valores por default.
- Observar el campo URL Mappings, se especifica la cadena que hay que escribir en el navegador para invocarlo.

**Create Servlet** 

Enter servlet deployment descriptor specific information.

**Name:**

**Description:**

**Initialization parameters:**

Name	Value	Description

[Add...](#)  
[Edit...](#)  
[Remove](#)

**URL mappings:**

/HolaMundoServlet
-------------------

[Add...](#)  
[Edit...](#)  
[Remove](#)



# Creando al servlet HolaMundoServlet (cont.)

- Indicar los métodos doXXX que se emplearán para generar el contenido del Servlet: doGet y doPost.

## Create Servlet

Specify modifiers, interfaces to implement, and method stubs to generate.



Modifiers: ☒ public ☐ abstract ☐ final

Interfaces:

Add...

Remove

Which method stubs would you like to create?

☐ Constructors from superclass

☒ Inherited abstract methods

☐ init

☐ destroy

☐ getServletConfig

☐ getServletInfo

☐ service

☒ doGet

☒ doPost

☐ doPut

☐ doDelete

☐ doHead

☐ doOptions

☐ doTrace



```
package mx.edu.unam.fi.temas.web.servlets;
```

```
import java.io.IOException;
```

```
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.WebServlet;
```

```
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
```

```
@WebServlet("/HolaMundoServlet")
```

```
public class HolaMundoServlet extends HttpServlet {
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
PrintWriter out;
```

```
out = response.getWriter();
```

```
out.write("<html><head><title>Mi primer Servlet</title></head>"
```

```
+ "<body><h3>Hola Mundo</h3></body></html>");
```

```
}
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
System.out.println("Invocando a doGet, se hace la misma acción para los 2 métodos");
```

```
doGet(request, response);
```

```
}
```

```
}
```

Observar que en este método se puede programar el contenido que se va a enviar al navegador web, es decir el HTML.





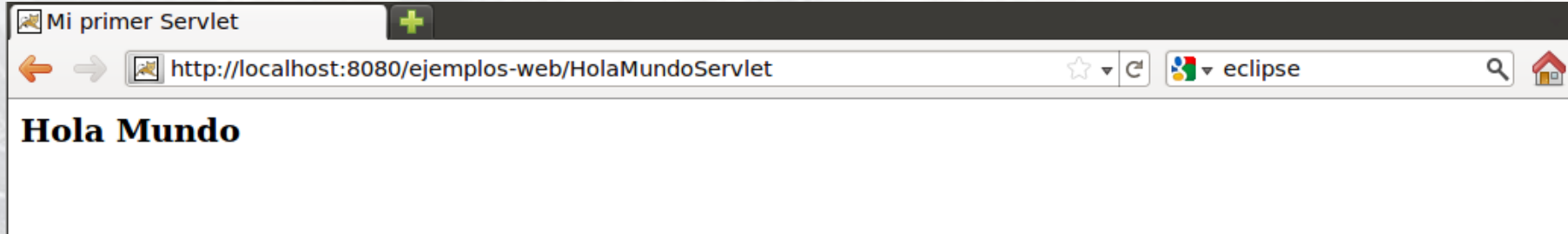
# Instalando la aplicación en tomcat.

- Para realizar la generación del archivo war mismo que será instalado en Tomcat, seguir las siguientes instrucciones:
  - Dar clic derecho en el proyecto web, seleccionar la opción export -> WAR file, puede ser cualquier ruta.



# Instalando la aplicación en Tomcat (cont).

- Finalmente, copiar el archivo generado **ejemplos-web.war** en el directorio **webapps** dentro del directorio de instalación de Tomcat.
- Verificar si el servidor ha sido iniciado. En caso contrario iniciarlo.
- Para invocar al servlet, en un navegador web, indicar la siguiente ruta. Observar que el URL se forma empleando la cadena que representa al context root de la aplicación (ejemplos-web), junto con la cadena especificada en **@WebServlet**



# Obteniendo datos de un formulario HTML

- Ejemplo: Suponer la siguiente página HTML, generar un Servlet que reciba los datos de la forma, extraiga los valores y los muestre en una nueva pagina.

**Ejemplo envío de datos por formulario HTML**

Campo de texto:

Password:

Multilínea:

Radio buttons:  
☐ A ☐ B ☐ C ☐ D ☐ E

Check boxes:  
☐ 1 ☐ 2 ☐ 3 ☐ 4

Combo box:

✕ Encontrar:  ◀ Anterior ▶ Siguiente 🟡 Marcar todo



```
<form name="form1" method="post" action="/ejemplos-web/FormularioServlet">
  <p>Campo de texto:<br /><input name="texto" type="text" id="texto" /><br />
    Password:<br /><input name="password" type="password" id="password" /><br />
    Multilínea:<br />
    <textarea name="textarea" cols="30" rows="5" id="textarea"></textarea><br />
    Radio buttons:<br />
    <input name="radioLetras" type="radio" value="A" />A
    <input name="radioLetras" type="radio" value="B" />B
    <input name="radioLetras" type="radio" value="C" />C
    <input name="radioLetras" type="radio" value="D" />D
    <input name="radioLetras" type="radio" value="E" />E<br />
    Check boxes:<br />
    <input name="checkNumero" type="checkbox" id="checkNumero" value="1" />1
    <input name="checkNumero" type="checkbox" id="checkNumero" value="2" />2
    <input name="checkNumero" type="checkbox" id="checkNumero" value="3" />3
    <input name="checkNumero" type="checkbox" id="checkNumero" value="4" />4<br />
    Combo box:<br />
    <select name="combo">
      <option value="0">SELECCIONE</option>
      <option value="1">UNO</option>
      <option value="2">DOS</option>
      <option value="3">TRES</option>
    </select>
  </p>
  <p align="center"><input type="submit" name="Submit" value="Enviar datos" /></p>
</form>
```





# Obteniendo datos de un formulario HTML (cont.)

- Guardar el contenido del archivo html dentro del directorio WebContent con el nombre **formulario.html**
- El siguiente punto es la creación del Servlet que recupere los valores de cada uno de los elementos de la forma.
- Observar que se emplea el valor del atributo **name**, para determinar el nombre del parámetro.
- Observar que algunos elementos como los checkboxes, tienen el mismo nombre de parámetro y se pueden enviar varios valores (un parámetro puede tener mas de un valor).
- Observar el valor del atributo action del elemento <form>, indica el URL, en este caso la ruta de un Servlet al que se le enviarán los valores de la forma:
  - **/ejemplos-web/FormularioServlet**
- Para obtener el valor de los parametros se emplean los métodos de HttpServletRequest:
  - **getParameter**
  - **getParameterValues**



# Obteniendo datos de un formulario HTML (cont.)

```
@WebServlet("/FormularioServlet")
public class FormularioServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        StringBuilder sb = new StringBuilder();
        String paramValue;
        String[] checkValues;
        PrintWriter out;

        System.out.println("Recibiendo datos del formulario.");
        sb.append("<html><head><title>Datos del formulario</title></head>");
        sb.append("<body>").append("<h3><div align=\"center\">Datos del formulario</h3><br/>");
        // observar que se emplea el nombre del elemento html
        paramValue = request.getParameter("texto");
        sb.append("Campo de texto: ").append(paramValue).append("<br/>");
        paramValue = request.getParameter("password");
        sb.append("Password: ").append(paramValue).append("<br/>");
        paramValue = request.getParameter("textarea");
        sb.append("Campo multi-linea: ").append(paramValue).append("<br/>");
        paramValue = request.getParameter("radioLetras");
        sb.append("Valor del radio button seleccionado: ").append(paramValue).append("<br/>");
```

Valor del atributo name del elemento html:

```
<input name="texto" type="text" id="texto" />
```



# Obteniendo datos de un formulario HTML (cont.)

```
// obtención de los valores de los checkboxes, pueden ser varios.
checkValues = request.getParameterValues("checkNumero");
sb.append("Valores de los checkboxes: ");
for (String valor : checkValues) {
    sb.append(valor).append(" ");
}
sb.append("<br/>");
// valor del combo
paramValue = request.getParameter("combo");
sb.append("Valor del combo: ").append(paramValue).append("<br/>");

sb.append("</body></html>");
// enviando el contenido al browser
out = response.getWriter();
out.write(sb.toString());
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    throw new UnsupportedOperationException("Este Servlet solo atiende peticiones POST.");
}
}
```





# Algunas desventajas del ejemplo anterior.

- El ejemplo anterior muestra una forma relativamente sencilla para extraer los datos de un formulario HTML.
- Sin embargo, para armar la respuesta a enviar al navegador web, en el código del servlet se crea una cadena con tags html.
- Esto es funcional pero se considera como mala práctica. Escribir html dentro de una clase Java es tedioso y complicado.
- Lo correcto es que todo el código HTML se encuentre en archivos HTML o en su defecto en JSPs (Java Server Pages).
- Los JSPs son paginas HTML con capacidades de mostrar contenido dinámico enviado por un servlet.
- Hay muchas cosas mas acerca de los servlets, lo visto hasta ahora es lo mínimo indispensable para intercambiar información entre un navegador y el servidor.





# Introducción a Java Server Pages

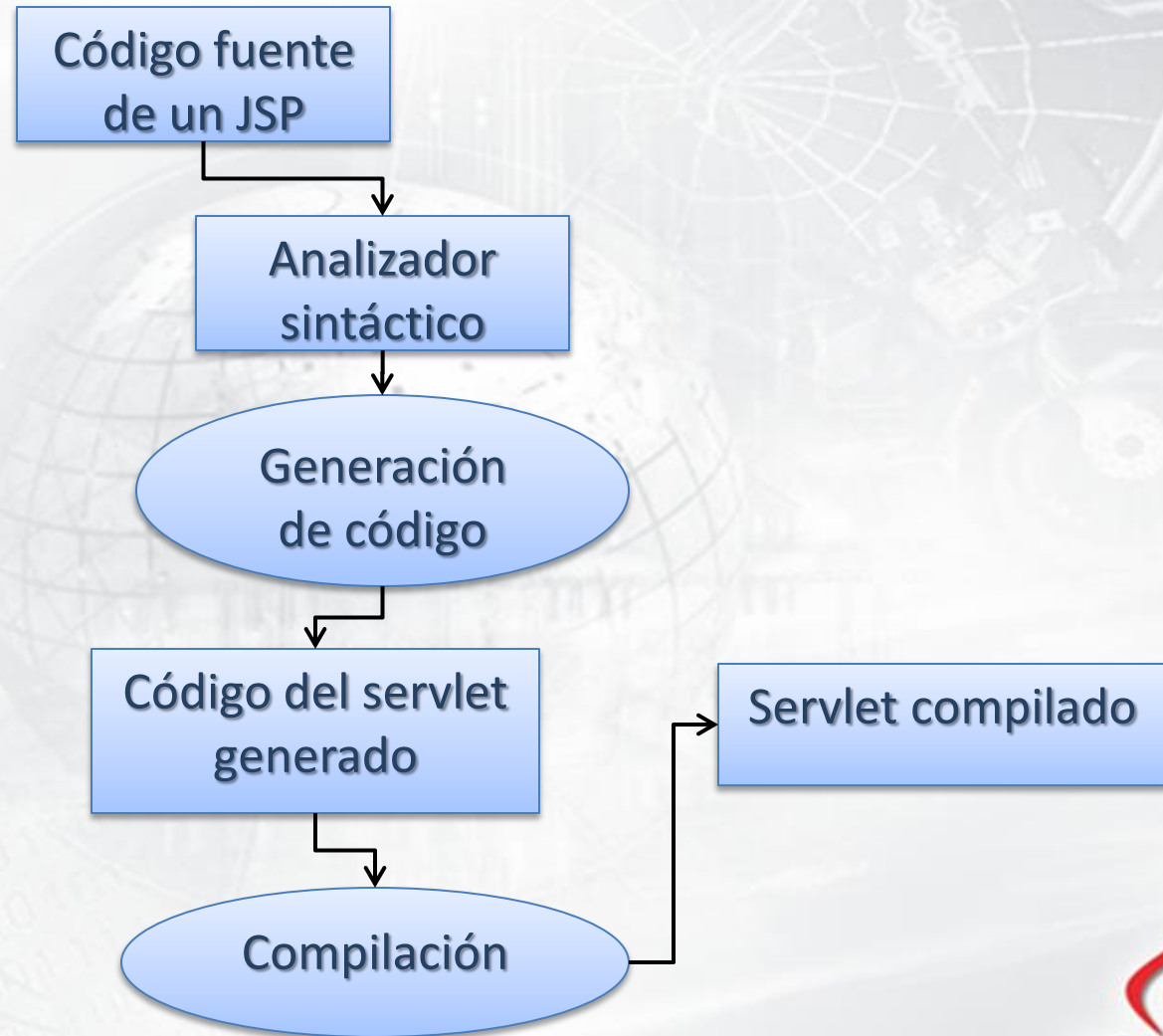
- La tecnología JSP (Java Server Pages) permite incluir contenido dinámico en un archivo html. Ahora podemos ver código java mezclado en HTTP.
  - ¿Qué es mejor Servlet o JSP?
- En la realidad una aplicación web contiene un 95% de JSP y un 5% o menos de servlets. El siguiente ejemplo, muestra un JSP que imprime la hora. (observar las líneas resaltadas).
- Para probar, guardar el contenido siguiente en WebContent/jsp/muestraHora.jsp
- Actualizar el archivo war en tomcat, ejecutar el jsp con la siguiente ruta: <http://localhost:8080/ejemplos-web/jsp/imprimeHora.jsp>

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page language="java"%>
<%@page import="java.util.Date"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP que imprime la fecha y hora.</title>
</head>
<body> <h1>La hora actual es:</h1>
<%=new Date() %>
</body>
</html>
```

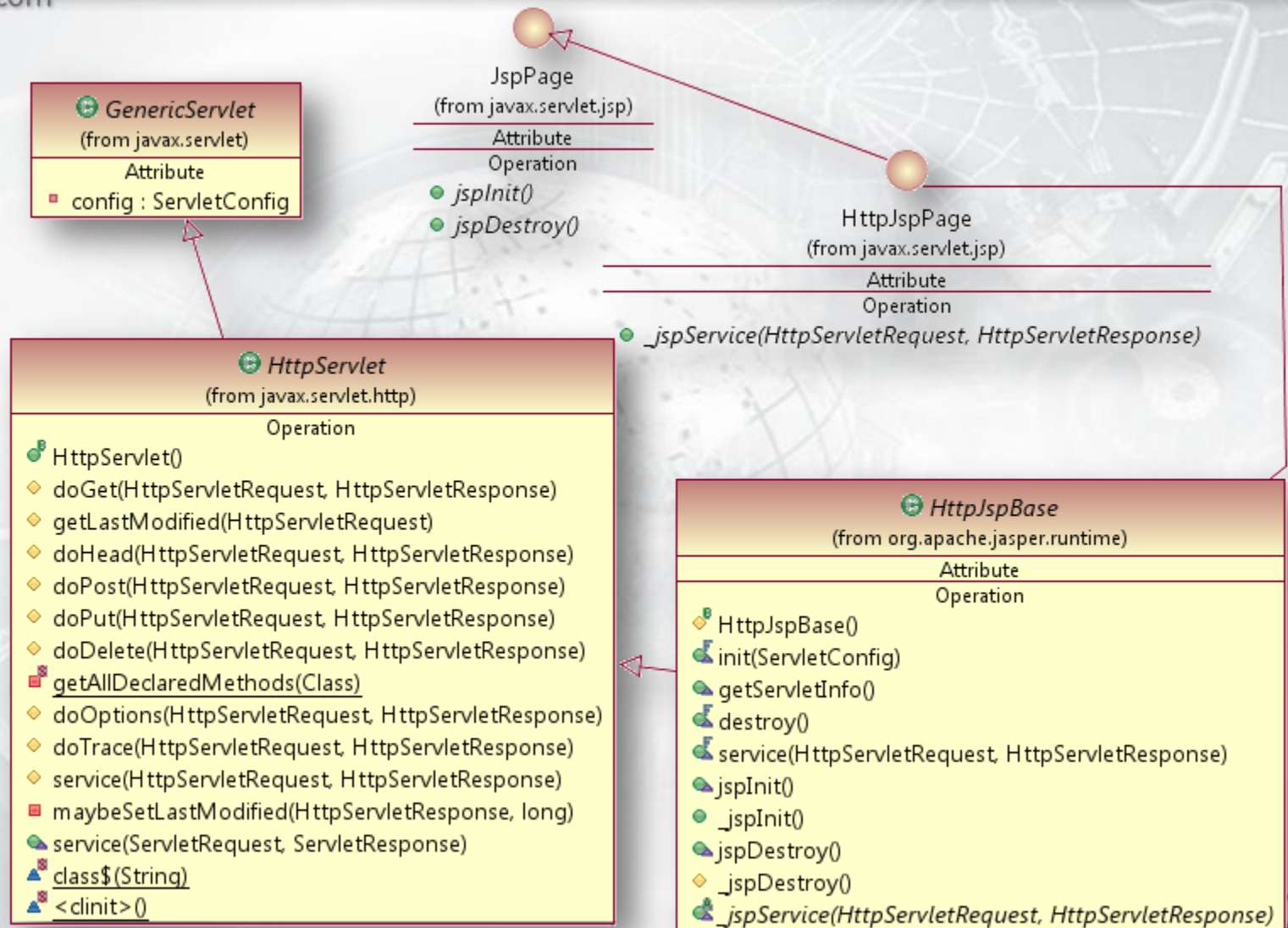


- Aunque es posible incluir código java dentro del HTML, para generar páginas dinámicas, la idea es separar los trabajos del diseñador y del programador.
- A través de nuevas versiones se hace énfasis en esta separación a través de nuevos elementos como son expression language, tags, etc.
- ¿Cómo es posible que un JSP llegue a ser un Objeto JAVA?
- En realidad el contenedor web tiene mecanismos de traducción para convertir un JSP en algo conocido en este caso a un servlet.
  - **Un JSP es un Servlet**





# Jerarquía de clases para JSPs en Tomcat





# El método \_jspService, ejemplo de traducción:

- Contiene todo el código HTML incluido en el JSP.
- Traduce las instrucciones a código Java.
- Para ejecutar un JSP, basta con ponerlo en un lugar accesible dentro de la aplicación web con extensión .jsp
- Ejemplo:

```
<html>
  <head>
    <title>Simple JSP Example</title>
  </head>
</html>
```

Traduce a:

```
out.write("<html><head><title>Simple JSP Example</title></head></html>");
```

```
<% = new Date() %>
```

Traduce a:

```
out.write(new Date());
```



## Archivo fuente

Plantilla o template del JSP (HTML)

### Elementos

#### Directivas

#### Elementos de scripting:

- Expressions
- Scriptlets
- Declarations
- Comentarios

EL (Expression Language)

### Actions

Standard

Custom

Para efectos del curso solo se revisarán los elementos de scripting básicos.



- Es todo aquello que requiere ser traducido, no puede enviarse al response de forma directa.
- Los elementos se reconocen por el uso de alguna secuencia de caracteres de inicio y fin .
- Existen 3 tipos de elementos:
  - **Directivas:** Proporcionan información global al JSP
  - **Actions:** Se emplean para realizar la inclusión de contenido dinámico en el JSP.
  - **Scripting:** Se emplean para presentar contenido dinámico y realizar lógica de procesamiento.



- Existen 4 tipos:

- Expresiones `<%= %>`
- Scriptlets `<% %>`
- Declarations `<%! %>`
- Comentarios `<%-- --%>`





- Expresiones:
  - Evalúan alguna expresión cuyo resultado se envía directamente a la salida de la pagina.
  - La expresión debe generar algún valor.
  - No terminar la expresión con “;”
  - Se emplean los elementos `<%= %>`
  - Ejemplo:
    - `<%= new Date() %>` se traduce a:
    - `out.print(new java.util.Date());`
  - La variable **out** es del tipo **JspWriter**, y siempre esta disponible en el JSP.

## JspWriter

(from javax.servlet.jsp)

### Operation

- ✦ JspWriter(int, boolean)
- 🌱 `newLine()`
- 🌱 `print(boolean)`
- 🌱 `print(char)`
- 🌱 `print(int)`
- 🌱 `print(long)`
- 🌱 `print(float)`
- 🌱 `print(double)`
- 🌱 `print(char[])`
- 🌱 `print(String)`
- 🌱 `print(Object)`
- 🌱 `println()`
- 🌱 `println(boolean)`
- 🌱 `println(char)`
- 🌱 `println(int)`
- 🌱 `println(long)`
- 🌱 `println(float)`
- 🌱 `println(double)`
- 🌱 `println(char[])`
- 🌱 `println(String)`
- 🌱 `println(Object)`
- 🌱 `clear()`
- 🌱 `clearBuffer()`
- 🌱 `flush()`
- 🌱 `close()`
- 🌱 `getBufferSize()`
- 🌱 `getRemaining()`
- 🌱 `isAutoFlush()`

- Scriptlets
  - Permite incluir varias sentencias Java dentro del método
  - El código debe estar delimitado por `<% %>`
  - Las instrucciones son directamente incluidas en el servlet resultante, por lo que cada instrucción debe incluir el “;”
  - Cualquier línea de código en java puede ser incluida en un scriptlet.
  - ¿Qué ventaja o desventaja tendrá lo anterior ?



- Declarations:
  - Se emplean para insertar código fuera del método **`_jspService()`**
  - Se identifican por los delimitadores **`<%! %>`**
  - Los elementos que se pueden incluir dentro de estos delimitadores son aquellos que pueden estar fuera de **`_jspService:`**
    - *Metodos de instancia y/o clase.*
    - *Atributos de instancia y/o clase.*



- Se emplean para evitar la traducción de algun elemento del JSP.
- Se identifican por los delimitadores `<%--` y `--%>`
- La restricción que tienen es que no pueden existir comentarios anidados.
- ¿Qué ocurre con el siguiente código?

```
<%-- <% for (int i = 0; i < 10; i++) {  
<%-- if(i==3){ System.out.println(  
"i is 3!");} --%>  
System.out.println(  
"i squared is " + i * i);  
} --%>
```





# Scripting elements: Resumen

- Resumen elementos scripting de los JSPs

Tipo de elemento	Inicia con	Termina con	Punto y coma después de las sentencias	Código generado dentro de <code>_jspService</code>
expresion	<code>&lt;%=</code>	<code>%&gt;</code>	NO	SI
scriptlet	<code>&lt;%</code>	<code>%&gt;</code>	SI	SI
Declaración	<code>&lt;%!</code>	<code>%&gt;</code>	SI	NO
Comentario	<code>&lt;%--</code>	<code>--%&gt;</code>	NA	NO

- Ejemplo: Programa que convierte una lista de números en diferentes unidades de medición:
  - Generar 2 JSPs. El primer JSP (**`seleccionaMedida.jsp`**) mostrará un combo que permita establecer la unidad de medida de conversión.
  - El segundo JSP (**`convierteMedida.jsp`**) realizará la conversión de los números con base a la unidad de medida seleccionada.



# JSP #1, selección de la unidad de medida

```

<!-- Declaración de un atributo de clase. -->
<%!private static int[] medidas = new int[] { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 };%>
<html>
<head><title>Conversion de unidades de medicion</title></head>
<body>
<div align="center"><b>Conversion de unidades de medicion</b></div>
<form action="convierteMedida.jsp"><br>
  Seleccione la unidad de medicion a convertir para los siguientes numeros<br>
  <select name="unidad">
    <option value="mm" selected="selected">milimetros</option>
    <option value="cm">centimetros</option>
    <option value="m">metros</option>
    <option value="km">kilometros</option>
  </select><br>
  <ul>
    <% for (int i = 0; i < medidas.length; i++) { %>
      <li><%=medidas[i] %></li>
    <% } %>
  </ul>
  <br>
  <div align="center"><input type="submit" value="convertir valores"></div>
</form>
</body>
</html>

```

Observar el elemento action de la forma de captura. Observar el uso de <%! %> y de <% %>



```

<!-- Declaración de un atributo de clase. -->
<%!private static int[] medidas = new int[] { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 };%>
<% String unidad = request.getParameter("unidad"); %>
<html>
<head><title>Programa de conversion de unidades</title></head>
<body>
<br>Realizando conversion de KM a diversas unidades<br>
Unidad seleccionada: <%=unidad%>
<ul>
<% for (int i = 0; i < medidas.length; i++) {
    if (unidad.equals("mm")) { %>

        <li><%=medidas[i]%> = <%=medidas[i] * (1000 * 1000)%> milimetros</li>

<%} else if (unidad.equals("cm")) { %>

        <li><%=medidas[i]%> = <%=medidas[i] * (100 * 1000)%> centimetros</li>

<%} else if (unidad.equals("m")) { %>

        <li><%=medidas[i]%> = <%=medidas[i] * 1000)%> metros</li>

<%} else { %>

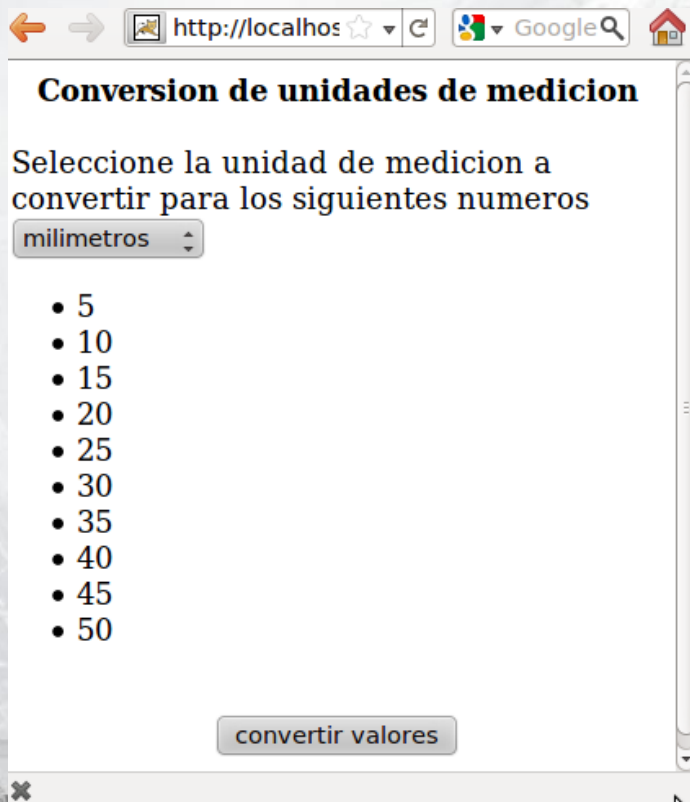
        <li><%=medidas[i]%> = <%=medidas[i]%> kilometros</li>

<%
    }
}%>
</ul><div align="center"><a href="seleccionaMedida.jsp">Regresar</a></div>
</body>
</html>

```



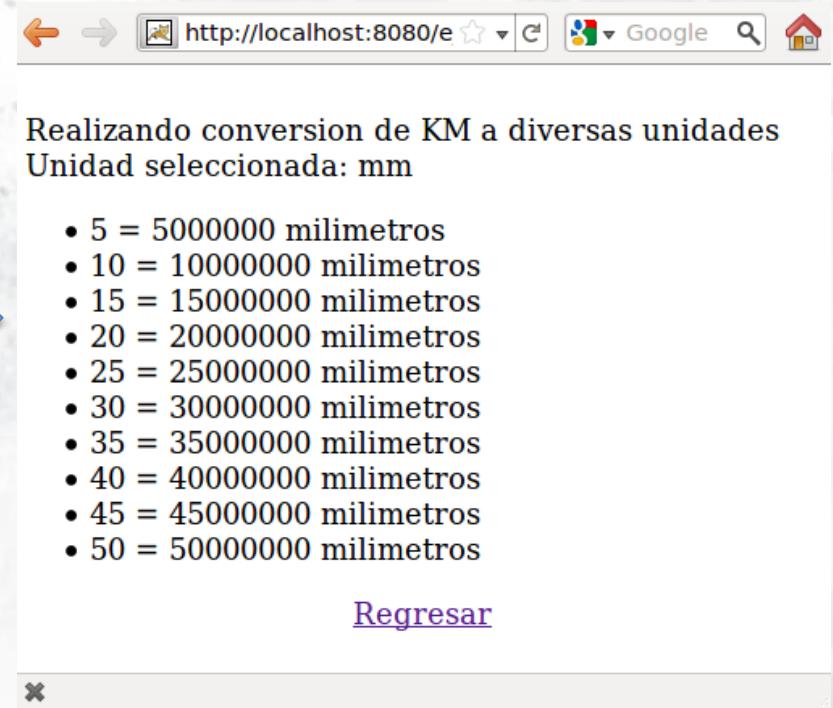
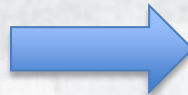
- Las siguientes pantallas muestran la salida de los 2 JSPs anteriores url:
  - <http://localhost:8080/ejemplos-web/jsp/seleccionaMedida.jsp>



**Conversion de unidades de medicion**

Seleccione la unidad de medicion a convertir para los siguientes numeros

- 5
- 10
- 15
- 20
- 25
- 30
- 35
- 40
- 45
- 50



**Realizando conversion de KM a diversas unidades**

Unidad seleccionada: mm

- 5 = 5000000 milimetros
- 10 = 10000000 milimetros
- 15 = 15000000 milimetros
- 20 = 20000000 milimetros
- 25 = 25000000 milimetros
- 30 = 30000000 milimetros
- 35 = 35000000 milimetros
- 40 = 40000000 milimetros
- 45 = 45000000 milimetros
- 50 = 50000000 milimetros

[Regresar](#)



# Los inconvenientes con los Scriptlets

- Aunque resuelven el problema de tener código Java mezclado con HTML en un Servlet, ahora se tiene código Java en un JSP.
- Sin embargo como se puede observar, se debe tener cuidado con el manejo de los elementos de scripting.
- Lo anterior reduce la facilidad para mantener el HTML. ¿Qué pasa si se desea modificar el diseño de la pagina con todos estos elementos agregados ? (es prácticamente imposible !).
- Para dar solución a estos inconvenientes, se emplean elementos como los siguientes:
  - Custom tags, por ejemplo JSTL.
  - Lenguaje simple para hacer referencia a objetos: Expression Language (EL).
- Estos elementos salen del alcance de este tema. La siguiente lámina observa un JSP sin elementos de script, en su lugar emplea JSTL y EL para un ciclo for.



# Ejemplo de un JSP con JSTL y EL

```
<c:forEach var="alumno" items="${listaAlumnos}" varStatus="estatus">
  <c:choose>
    <c:when test="${curso eq jb}">
      <br><c:out value='Java Basico' />
    </c:when>
    <c:when test="${curso eq ji}">
      <br><c:out value='Java Intermedio' />
    </c:when>
    <c:when test="${curso eq sp}">
      <br><c:out value='Spring' />
    </c:when>
    <c:when test="${curso eq hb}">
      <br><c:out value='Hibernate' />
    </c:when>
    <c:when test="${curso eq fb}">
      <br><c:out value='frameworks web' />
    </c:when>
    <c:otherwise>
      <br>curso no valido
    </c:otherwise>
  </c:choose>
  <c:set var="sizeCursos" value="${cursoEstatus.count}" scope="request"/>
</c:forEach>
```



- En un escenario típico, toda la validación y procesamiento inicial de la información se debe realizar en un Servlet o en frameworks mas especializados, en un componente llamado **Controlador de peticiones web**, mismo que a su vez invoca otros componentes del sistema para realizar el procesamiento completo y de requerir, la persistencia en base de datos. Ejemplos de estos frameworks:
  - Struts
  - SpringMVC
  - Grails, etc.
- El JSP solo deberá contener la programación necesaria para mostrar y llenar los elementos de una pagina web con datos dinámicos.
- Por lo anterior, se requiere de un mecanismo de comunicación entre el **controlador** y los JSPs.





# Enviando objetos a los JSPs (cont.)

- Esta comunicación se realiza enviando objetos al JSPs los cuales se organizan en 4 alcances (Scopes):
  - Application
  - Session
  - Request
  - Page
- Para efectos del curso solo se revisará el alcance **request**, y el controlador estará representado por un **Servlet**.
- Este alcance se representa a través del objeto **HttpServletRequest**.
- Tanto los servlets como los JSPs pueden emplear los siguientes métodos para enviar y recibir objetos durante una petición http:
  - `void setAttribute(String attributeName, Object attributeValue)`
  - `Object getAttribute(String attributeName)`
- Como se puede observar, estos métodos aceptan cualquier objeto como atributo. (Similar a un Map).





# Enviando Objetos a los JSPs: Ejemplo

- Retomando el caso de estudio CENAT, suponer que se desea mostrar un JSP en el que se seleccione un estado de la republica mexicana para mostrar una lista de todos los centros teatrales que pertenezcan a dicho estado.
- La consulta (simulada), será invocada empleando un servlet que a su vez enviará una lista de objetos a otro JSP para mostrar el resultado.
- Esta lista de objetos se envía al JSP asociándola al scope (alcance) request, representada por el objeto **HttpServletRequest**. Se emplea el método **setAttribute** como se muestra:



# JSP #1 Captura de la entidad: buscaCentroTeatral.jsp

www.synergyj.com

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Busqueda de centros teatrales</title>
</head>
<body>
<p><strong>Buscando centros teatrales por entidad</strong></p>
<form id="form1" name="form1" method="post" action="/ejemplos-web/BusquedaCentroTeatralServlet">
  <p>Seleccione el estado de la Rep&uacute;blica Mexicana<br />
    <select name="entidadID">
      <option value="1">AGUASCALIENTES</option>
      <option value="2">BAJA CALIFORNIA</option>
      <option value="3">BAJA CALIFORNIA SUR</option>
      <option value="4">CAMPECHE</option>
      <option value="5">COAHUILA</option>
      <option value="6">COLIMA</option>
      <option value="7">CHIAPAS</option>
      <option value="8">CHIHUAHUA</option>
      <option value="9">DISTRITO FEDERAL</option>
      <option value="10">DURANGO</option>
      <option value="11">GUANAJUATO</option>
      <option value="12">GUERRERO</option>
      : : : :
      : : : :
    </select></p>
    <p align="center"><input type="submit" name="Submit" value="Buscar Centros" /></p>
  </form>
</body>
</html>
```

Observar que esta forma invoca al servlet  
**BusquedaCentroTratralServlet**



# Servlet que recibe la petición para realizar la consulta.

www.synergyj.com

```
@WebServlet("/BusquedaCentroTeatralServlet")
public class BusquedaCentroTeatralServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        int entidadID;
        List<CentroTeatral> listaCentros;
        RequestDispatcher dispatcher;

        entidadID = Integer.parseInt(request.getParameter("entidadID"));
        System.out.println("Invocando la busqueda de centros teatrales para la entidad: "
            + entidadID);
        // aqui en este punto se podría invocar a un servicio o clase para realizar
        // la búsqueda hacia la base de datos, por ejemplo empleando hibernate. Aqui
        // solo se simula la obtención de una lista ficticia.
        listaCentros = buscaCentrosTeatrales(entidadID);
        System.out.println("Enviando lista al JSP");
        dispatcher = request.getRequestDispatcher("/jsp/listaCentrosTeatrales.jsp");
        // sube el objeto al scope Request.
        request.setAttribute("listaCentros", listaCentros);
        // realiza un forward al jsp que mostrara la lista obtenida.
        dispatcher.forward(request, response);
        // Observar que no hay codigo html en este servlet, solo se manda la lista
        // para que en el jsp se itere.
    }
}
```

Observar el uso del objeto **RequestDispatcher**, y el envío de información al JSP **listaCentrosTeatrales.jsp** empleando **setAttribute** del objeto **request**.



# Código BusquedaCentroTeatralServlet (cont.)

```
private List<CentroTeatral> buscaCentrosTeatrales(int entidadID) {
    CentroTeatral centro;
    List<CentroTeatral> listaCentros;

    listaCentros = new ArrayList<CentroTeatral>();
    // centro 1
    centro = new CentroTeatral();
    centro.setCentroTeatralID(1);
    centro.setCiudad("Centro de la ciudad");
    centro.setClave("CR-0003");
    centro.setCodigoPostal("01010");
    centro.setColonia("Colonia centro");
    centro.setEntidadID(entidadID);
    centro.setFechaCreacion(new Date(0));
    listaCentros.add(centro);
    // centro 2
    centro = new CentroTeatral();
    centro.setCentroTeatralID(1);
    centro.setCiudad("Sur de la ciudad");
    centro.setClave("CR-0004");
    centro.setCodigoPostal("83298");
    centro.setColonia("Sur 13");
    centro.setEntidadID(entidadID);
    centro.setFechaCreacion(new Date(0));
    listaCentros.add(centro);
    return listaCentros;
}
```

Este método simula la búsqueda en la base de datos. Aquí se podría invocar a una clase o servicio que invoque la búsqueda empleando por ejemplo Hibernate. Para integrar a Hibernate, solo es necesario copiar todas las librerías al directorio **WebContent/WEB-INF/lib** y agregarlas al classpath del proyecto.





# Java™ JSP #2 Lista de centros: listaCentrosTeatrales.jsp

www.synergyj.com

```
<%@page import="mx.edu.unam.fi.temas.web.entidades.CentroTeatral"%>
<%@page import="java.util.List"%>
<%
    List<CentroTeatral> listaCentros;
    listaCentros = (List<CentroTeatral>)request.getAttribute("listaCentros");
%>
<html>
<head>
<title>Resultados de busqueda</title></head>
<body>
<p><strong>Busqueda de Centros teatrales.</strong></p>

<%if(listaCentros== null || listaCentros.size()==0){ %>
    <div align="center">
        <b>No se encontraron centros teatrales para la entidad seleccionada</b></div>
<%} else {%>

<table width="600" border="0">
<tr>
    <td bgcolor="#ABB5EB">id</td>
    <td bgcolor="#ABB5EB">Ciudad</td>
    <td bgcolor="#ABB5EB">Clave</td>
    <td bgcolor="#ABB5EB">Codigo Postal </td>
    <td bgcolor="#ABB5EB">Colonia</td>
    <td bgcolor="#ABB5EB">Entidad </td>
    <td bgcolor="#ABB5EB">Fecha de creaci&ocirc;n </td>
</tr>
```

Observar la forma en la que se recuperan los objetos del alcance (scope) a nivel del objeto **request**. Se emplea el método **getAttribute** y se aplica casting a la lista ya que el método regresa un **Object**.



# JSP #2 listaCentrosTeatrales.jsp (cont.)

```
<%for(CentroTeatral centro:listaCentros){ %>
<tr>
<td><%=centro.getCentroTeatralID()%></td>
<td><%=centro.getCiudad()%></td>
<td><%=centro.getClave()%></td>
<td><%=centro.getCodigoPostal()%></td>
<td><%=centro.getColonia()%></td>
<td><%=centro.getEntidadID()%></td>
<td><%=centro.getFechaCreacion()%></td>
</tr>
<!-- fin del for -->
<%}%>
</table>

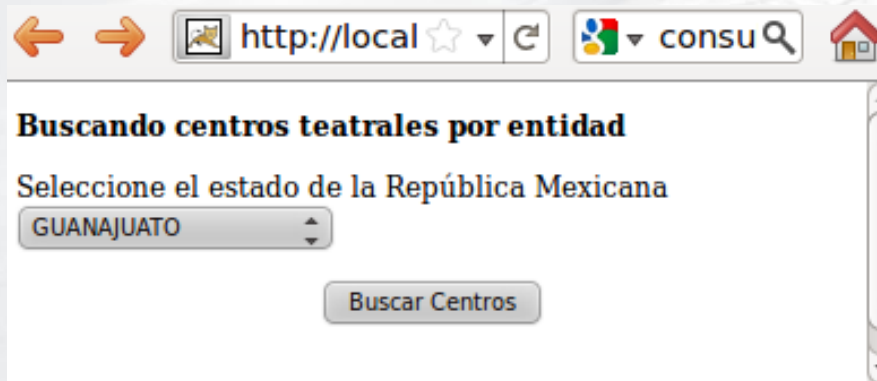
<!-- fin del else -->
<%} %>
<p align="center"><a href="/jsp/buscaCentroTeatral.jsp">Regresar</a></p>
<p>&nbsp;</p>
</body>
</html>
```

Observar el ciclo for, se itera la lista de centros teatrales obtenida anteriormente y por cada elemento de la lista se agrega un renglón a la tabla HTML.



# El resultado obtenido es el siguiente..

- URL:
  - <http://localhost:8080/ejemplos-web/jsp/buscaCentroTeatral.jsp>



Buscando centros teatrales por entidad

Seleccione el estado de la República Mexicana

GUANAJUATO

Buscar Centros



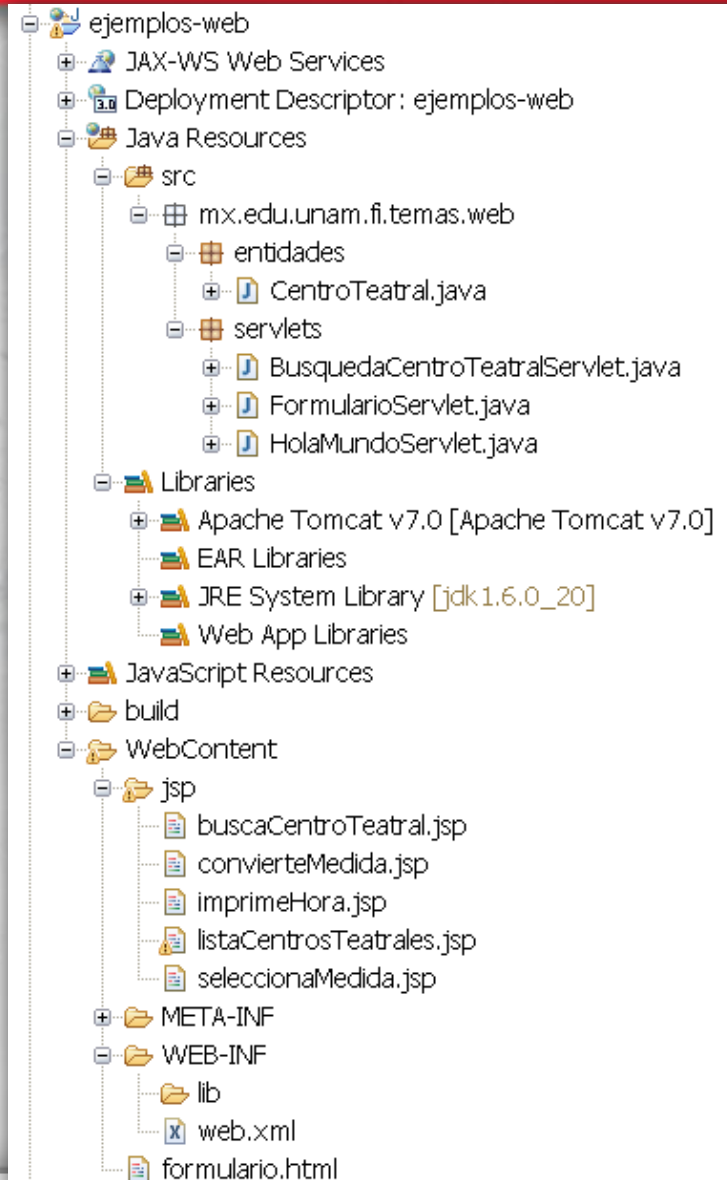
Busqueda de Centros teatrales.

id	Ciudad	Clave	Codigo Postal	Colonia	Entidad	Fecha de creación
1	Centro de la ciudad	CR-0003	01010	Colonia centro	11	Wed Dec 31 18:00:00 CST 1969
1	Sur de la ciudad	CR-0004	83298	Sur 13	11	Wed Dec 31 18:00:00 CST 1969

[Regresar](#)

# Estructura del proyecto web..

- Para mayor claridad, la estructura de la aplicación web que tiene todos ejemplos vistos anteriormente es la mostrada en la figura:





- Como se puede observar, existen muchos temas mas que revisar relacionados con el desarrollo de aplicaciones web en Java.
- En estas láminas solo se ha presentado una introducción sencilla del uso de la especificación JEE en la parte web. Se recomienda seguir con la documentación del sitio y/o con la bibliografía propuesta al inicio del semestre.



## 10. INTRODUCCIÓN AL DESARROLLO WEB CON JAVA

### **EJERCICIO 1.**



1. Mejorar el programa que realiza la búsqueda de centros teatrales, en lugar de simular la búsqueda, incorporar las clases y librerías necesarias para que se realice la conexión a una base de datos empleando JDBC o hibernate, de tal forma que los resultados mostrados son los que realmente existen en la base de datos.
2. Generar un programa que inserte un nuevo centro teatral empleando una pagina web. Los datos se deberán persistir en la base de datos.



**FIN DE CURSO.**

