

Desarrollo Web


Parte 1
Edición 2017

Taller de Programación
www.fing.edu.uy/inco/cursos/tprog
tprog@fing.edu.uy

Temario

- **Fundamentos de la World Wide Web**
 - **WWW**
 - **URI y URL**
 - **HTTP**
 - **HTML**
 - **Aplicaciones Web**
- **Desarrollo Web en Java**
 - **Java EE**
 - **Servlets**
 - **JSP**

Fundamentos de la World Wide Web

A series of horizontal lines in teal and light blue colors, with varying lengths and offsets, creating a modern, layered effect across the width of the slide.

World Wide Web (WWW)

- Es un espacio de información formado por *recursos* (objetos de interés) que se identifican mediante un identificador global denominado *Uniform Resource Identifier* (URI).
- Un recurso (u objeto) puede ser un archivo HTML, una imagen un audio, un video, etc., que tiene asociado al menos un URI.
- Una *página Web* (o documento Web) tiene un archivo HTML base y varios objetos enlazados mediante sus URIs (ej. imágenes)

URI y URL

- URI (Uniform Resource Identifier) es una cadena de caracteres que identifica un recurso físico o abstracto.
- URL (Uniform Resource Locator) es un subtipo de URI que permite localizar un recurso mediante un protocolo.

`"http://" host [":" port] [abs_path ["?" query]]`

- Host: El nombre o dirección IP del servidor
- Port: es el puerto TCP en el que el servidor Web escucha. Por general es 80
- Abs_path: ruta del archivo/pagina/recurso que se desea obtener
- Query: Permite pasar parámetros extra al servidor (...
lenguaje=es&pais=uy&ciudad=salto...)

<http://www.google.com.uy:80/search?q=taringa>

HTTP

- HyperText Transfer Protocol
- Protocolo de capa de aplicación de Internet
- Arquitectura cliente-servidor
- Los clientes envían mensajes de tipo *pedido* (request) y los servidores devuelven mensajes de tipo *respuesta* (response)

HTTP

Navegador



GET <http://example.com/index.html>

Pedido

Respuesta

Servidor Web

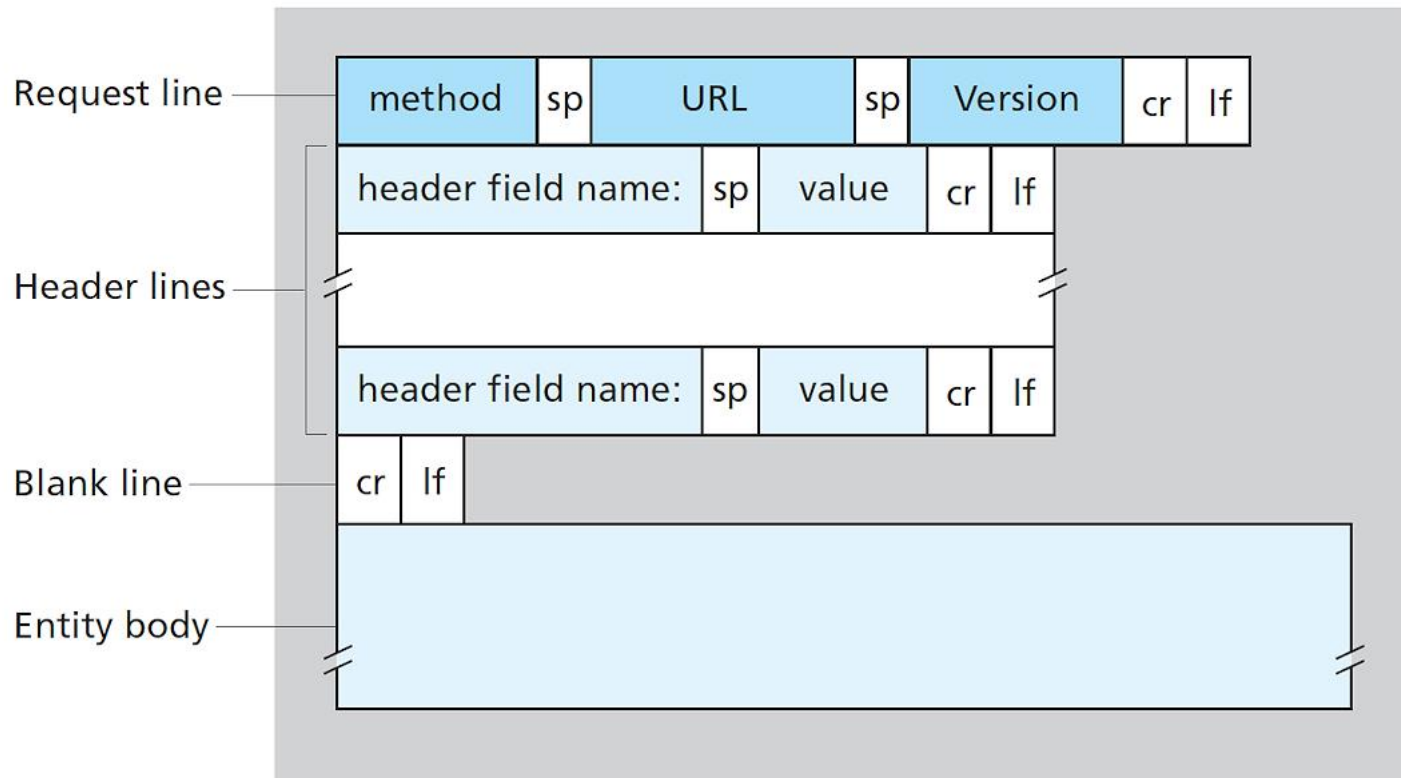


```
<HTML>
<HEAD>
  <TITLE>Example Web Page</TITLE>
</HEAD>
<BODY>
  You have reached this page.
  Congratulations
</BODY>
</HTML>
```

HTTP - Pedido

- Existen 9 métodos (GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS, PATCH, CONNECT).
- Los métodos más utilizados son *GET* y *POST*:
 - GET (método por defecto)
 - Obtiene un recurso
 - Parámetros en URL (ej. <http://www.bing.com/search?q=uruguay>)
 - Más apropiado para enviar poca información
 - POST
 - Envía información
 - Parámetros en cuerpo del pedido
 - Más apropiado para enviar mucha información (ej. archivos)

HTTP – Pedido - Formato



HTTP – GET - Ejemplo

```
GET /search?q=venecia&client=firefox-a HTTP/1.1
Host: www.google.com.uy
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: es-es;q=0.8,en-us;q=0.5,en;q=0.3
Cache-Control: max-age=0
Connection: keep-alive
Cookie: PgeoIP_country_code=UY; geoIP_country_name=Uruguay; geoIP_city=Montevideo;
```

HTTP – POST - Ejemplo

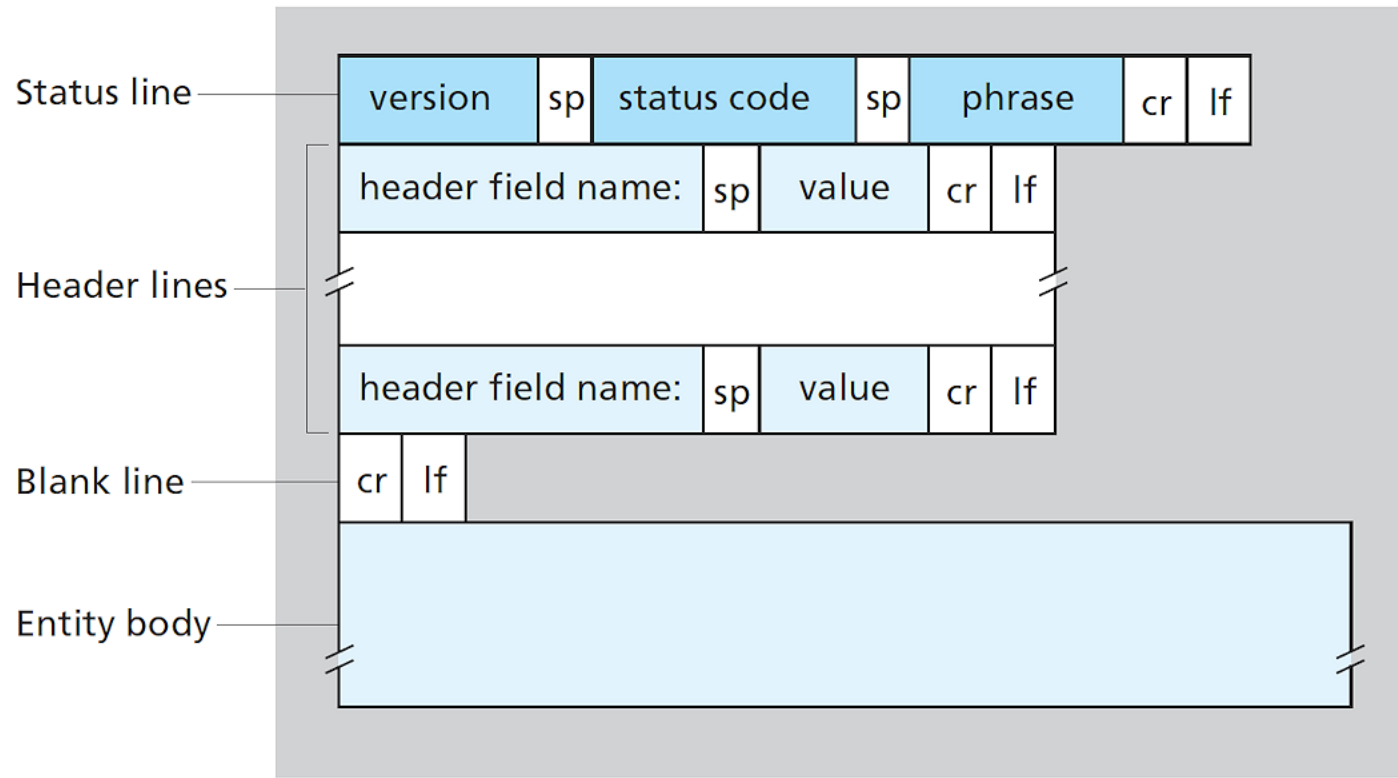
```
POST        /search HTTP/1.1
Host        www.google.com.uy
User-Agent   Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Accept      text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding gzip, deflate
Accept-Language es-es;q=0.8,en-us;q=0.5,en;q=0.3
Cache-Control max-age=0
Connection   keep-alive
Cookie       PgeolP_country_code=UY; geolP_country_name=Uruguay; geolP_city=Montevideo;

q=venecia&client=firefox-a
```

HTTP - Respuesta

- **Código de respuesta (algunos):**
 - **200: OK**
 - **404: Page Not Found (URL inválida)**
 - **500: Internal Server Error**
- **Encabezados: almacenan información extra (cookies, cache, etc.)**
- **Cuerpo: el contenido para que muestre el navegador.**

HTTP – Respuesta - Formato



HTTP – Respuesta - Ejemplo

```
HTTP/1.1 200 OK
Cache-Control      private, max-age=0
Content-Encoding   gzip
Content-Type       text/html; charset=UTF-8
Date               Sun, 26 Aug 2012 01:02:06 GMT
Expires            -1
Server             gws
Transfer-Encoding  chunked
X-Frame-Options    SAMEORIGIN
x-xss-protection   1; mode=block
```

```
<!doctype html><html itemscope="itemscope" itemtype="http://schema.org/
WebPage"><head><meta itemprop="image" content="/images/
google_favicon_128.png"><title>venecia - Buscar con Google</title>...
```

HTTP - Sesión

- Una sesión Web es un conjunto de pedidos que un navegador hace a un servidor Web
- Como HTTP no mantiene estado ni información entre pedidos, debe ser la plataforma la que provea un mecanismo para guardar la información de sesión.
- Existen dos mecanismos:
 - Reescritura de URL
 - Cookies

HTTP – Sesión - URL

- La reescritura de URL consiste en agregar datos de la sesión en cada URL.
- Esto tiene como desventajas:
 - Menor seguridad
 - URL largas y menos entendibles
- Ejemplo:
`http://www.example.com/algo.jsp;JSESSIONID=ABAD1D`

HTTP – Sesión - Cookies

- Una cookie es un conjunto de datos que el servidor envía al navegador para que lo guarde en un archivo.
- El servidor utiliza el encabezado Set-Cookie en la primera respuesta para indicar que el navegador guarde un ID de sesión

Set-Cookie: JSESSIONID=ABAD1D;path=/

- El navegador utiliza el encabezado Cookie en todas los pedidos siguientes a ese servidor con el mismo ID de sesión

Cookie: JSESSIONID=ABAD1D

HTML

- **HyperText Markup Language**
- **Lenguaje de marcas para interfaz de usuario**
- **Declarativo, específico de dominio**
- **El más utilizado para escribir páginas Web**
- **Los navegadores convierten el código HTML en páginas visuales y audibles.**
- **Existe desde 1991, última versión es HTML 5.1 (2016)**

HTML - Elementos

- Un documento HTML consiste en elementos que se delimitan mediante etiquetas.
- Cada elemento tiene una etiqueta de inicio y otra de fin (hay excepciones). Ej:
 <tagname> Texto aquí </tagname>
- Los elementos pueden tener atributos. Ej:
 <tagname attrname="atributo"> Texto aquí
 </tagname>
- Los elementos se anidan entre sí. Ej:
 <parent> <child>...</child><child>...</child></parent>

HTML - Estructura

`<!DOCTYPE html>` `<!-- instrucción para preproceso -->`

`<html>` `<!-- elemento raíz -->`

`<head>` `<!-- encabezado -->`

Acá van el título, metadatos, enlaces a otros recursos (ej. hojas de estilo, scripts, etc.), etc. Esto no se muestra en la página.

`</head>`

`<body>`

Acá va el contenido, lo que se va a mostrar.

`</body>`

`</html>`

HTML - Ejemplo

```
<!DOCTYPE html>
<html>
<head>
<title>Taller de Programación</title>
<meta charset="UTF-8">
<meta name="author" content="TProg">
<meta name="e-mail" content="tprog@fing.edu.uy">
</head>
<body>
<h1>Página Web Mínima </h1>
<p>Hello World!</p>
</body>
</html>
```

Página Web Mínima

Hello World!

HTML – Formularios <form>

- Mecanismo básico de envío de información al servidor
- Permite utilizar HTTP GET o HTTP POST
- Elementos relacionados:
 - <input>: campo de entrada de datos. Toma diferentes formas de acuerdo al atributo *type*:
 - text: entrada de texto de una línea
 - password: como el anterior pero ocultando el texto
 - radio: botón con una opción excluyente
 - submit: botón que envía el formulario al *action* del *form*(en HTML5 se agregaron muchos tipos nuevos: email, date, color, number, etc.)

HTML – Formularios <form>

search:

number:

range:

color:

tel:

url:

email:

date:

month:

week:

time:

datetime:

datetime-local:

HTML – Formularios <form>

```
<body>
<form action="submit_info" method="post">
  <label for="name">Nombre:</label>
  <input type="text" id="name" name="name"/><br/>
  <label for="pass">Clave:</label>
  <input type="password" id="pass" name="pass"/><br/>
  <input type="checkbox" name="remember" value="rememberMe"/>
  Remember Me
  <br/>
  <input type="submit" value="Enviar"/>
</form>
</body>
```

Nombre:

Clave:

☐ Remember Me

Aplicaciones Web

- Aplicaciones cliente-servidor que residen en un servidor Web y son accesibles a través de la red usando un navegador.
- Constan de recursos estáticos (páginas Web) y de componentes ejecutables escritos en algún lenguaje de programación.
- Generalmente, los servidores Web ejecutan estos componentes y generan contenido dinámicamente que envían al cliente.
- Algunos componentes pueden ser enviados al cliente y ejecutados por el navegador.
- Los componentes se pueden ejecutar únicamente en el servidor (server-side), únicamente el cliente (client-side) o en ambos.

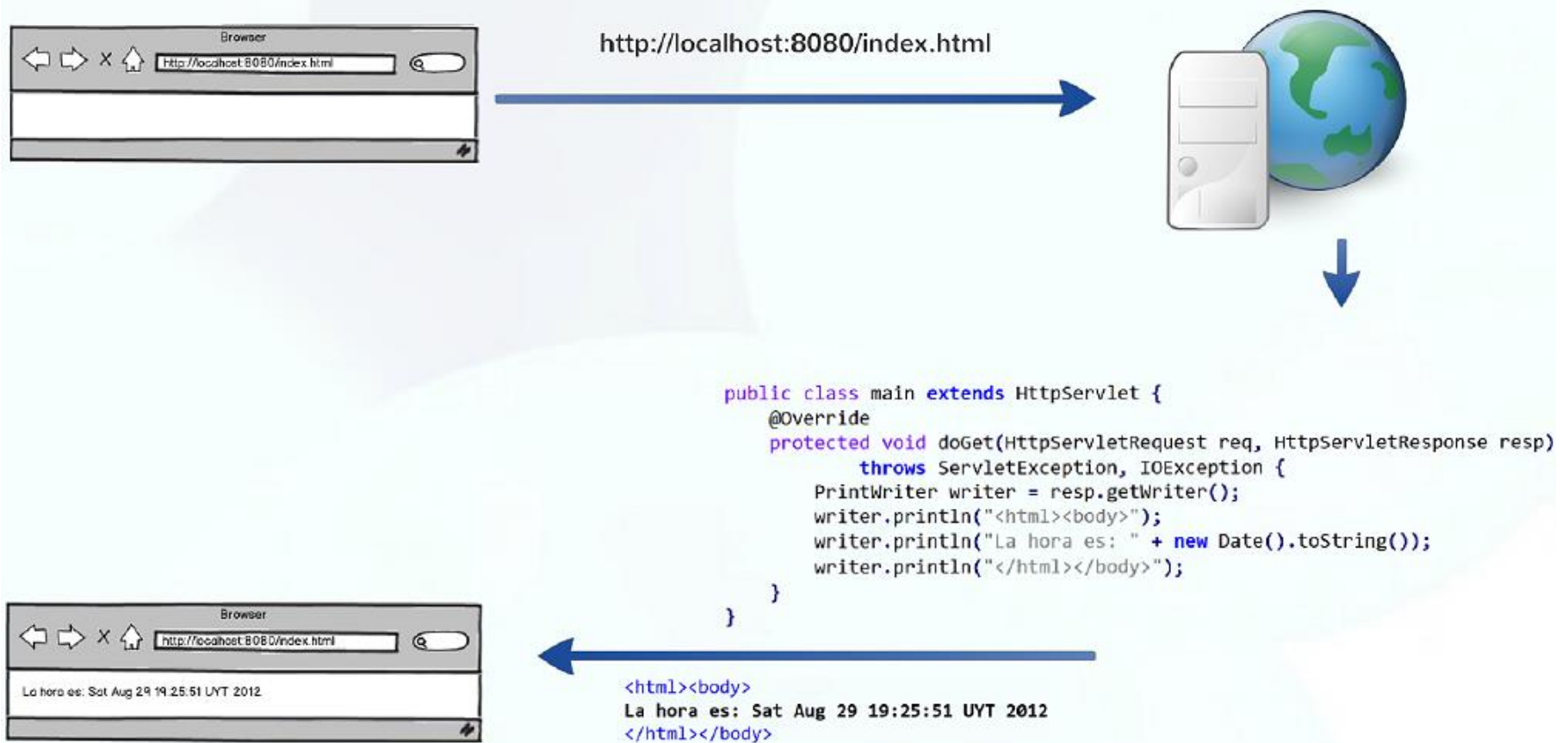
Aplicaciones Web - Ventajas

- **Fácil distribución:** no hay que instalar librerías ni ejecutables.
- **Actualización automática:** cambio en la aplicación Web se refleja para todos los usuarios instantáneamente.
- **Más escalable** a la cantidad de usuarios.
- **Accesible a los usuarios:** solo necesitan un navegador (disponibles en todas las plataformas).

Aplicaciones Web - Server-side Scripting

- Método de desarrollo Web en donde el código (script o programa) se ejecuta en el servidor para generar páginas Web dinámicas.
- Los servidores Web pueden, según el caso, ejecutar directamente el código o utilizar intérpretes, módulos y ambientes externos para ejecutarlo.
- Ejemplos de tecnologías para server-side scripting: Servlets/JSP/JSF (Java EE), ASP.NET (Microsoft), PHP, Ruby on Rails, JavaScript (Node.js), Python, Perl, etc.

Aplicaciones Web - Server-side Scripting - Ejemplo



Aplicaciones Web: Client-side Scripting

- Método de desarrollo Web en donde el código (script o programa) se envía al cliente para ser ejecutado en el navegador, lo que permite mostrar contenido dinámico y manejar las interacciones del usuario en forma similar a las aplicaciones de escritorio o nativas.
- Ejemplos de tecnologías para client-side scripting:
 - JavaScript: soporte nativo en navegadores
 - TypeScript: debe ser transpilado a JavaScript
 - Rich Internet Applications: precisan un plug-in en navegador
 - Flash/Flex
 - Silverlight
 - JavaFX

Desarrollo Web en Java

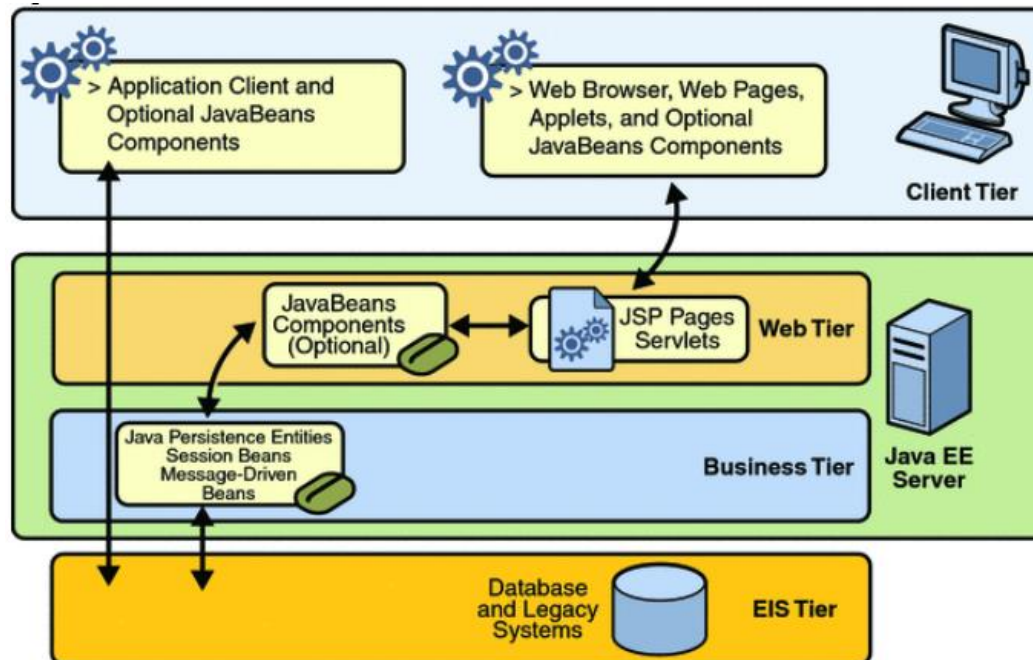
A series of horizontal lines in teal and light blue colors, with varying lengths and offsets, creating a modern, layered effect across the middle of the slide.

Java EE

- **Java Platform, Enterprise Edition (antes J2EE)**
- **Se basa en el lenguaje Java y ejecuta sobre el Java Runtime Enviroment (Java Virtual Machine y Java Class Library)**
- **Esta compuesta por:**
 - **Un modelo de aplicación**
 - **Un conjunto de especificaciones de APIs (Application Programming Interface)**
 - **Una arquitectura del ambiente de ejecución**

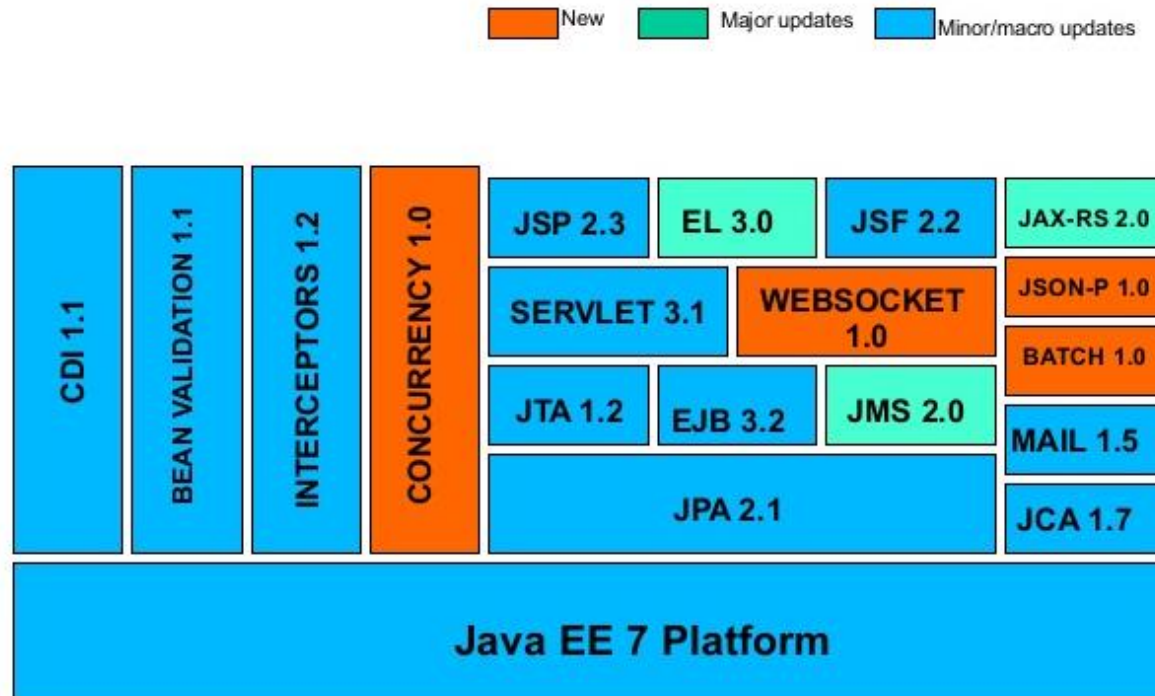
Java EE – Modelo de Aplicación

- Aplicaciones empresariales multicapa distribuidas
- Hasta 4 niveles o capas (tiers): capa de cliente, capa Web, capa de negocio y capa de sistema de información empresarial (EIS)



Java EE - API

- Cada especificación se define en un documento llamado JSR (Java Specification Request) que forma parte del Java Community Process (www.jcp.org)



Java EE – Arquitectura de Ejecución

- Se basa en el concepto de **contenedor** (container).
- Un contenedor es un ambiente de ejecución (runtime) que se encarga de
 - Manejar el ciclo de vida del componente
 - Brindar Servicios (transacciones, persistencia, inyección de dependencias, seguridad, mensajería, validación, etc.)
- Cada tipo de componente corre en un tipo de contenedor específico.
 - Ej: un servlet (Web component) corre en un Web Container.

Java EE – Arquitectura de Ejecución

- Un *servidor de aplicaciones* Java EE es un software que implementa algunos o todos los tipos de contenedores Java EE.
- Apache Tomcat es un servidor de aplicaciones que implementa JEE parcialmente.
- Los principales componentes internos de Tomcat son:
 - Catalina: contenedor Web (servlets, JSP)
 - Jasper: motor de procesamiento de JSP
 - Coyote: servidor HTTP

Java EE – Niveles de implementación de JEE

- Dentro de los proyectos de Apache Software Foundation, existen diferentes servidores de aplicaciones que implementan JEE en diferente grado.



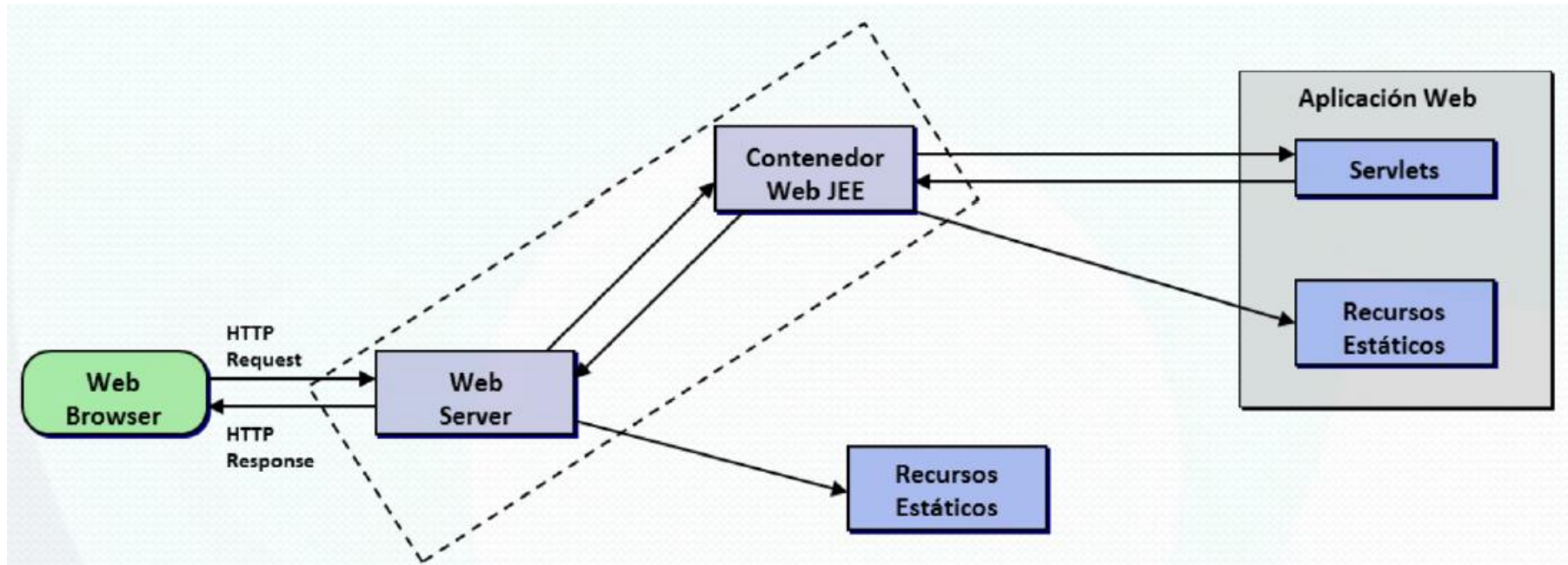
Geronimo (JEE Full Platform)

TomEE (JEE Web Profile)

Tomcat (Servlet & JSP)



Java EE - Servidor de Aplicaciones

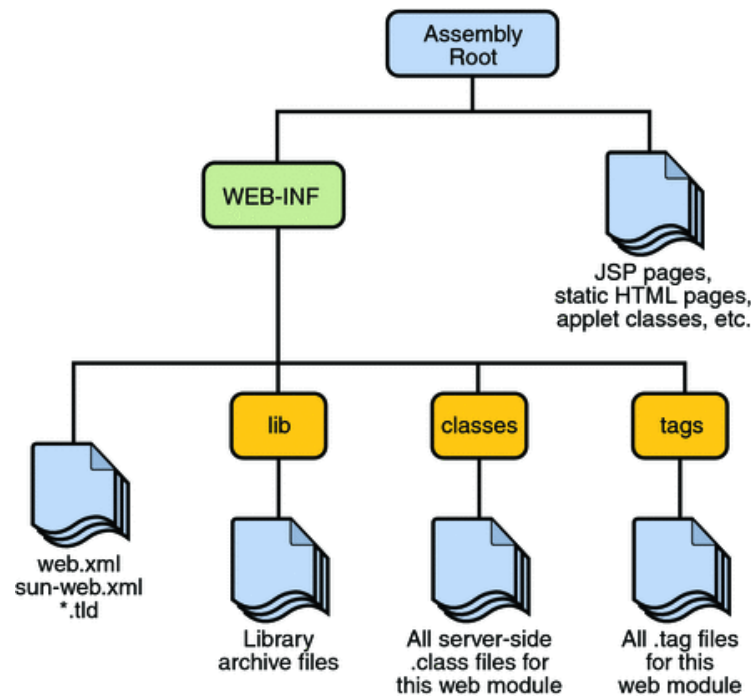


Java EE – Deployment

- Las aplicaciones JEE se empaquetan en una o más unidades de deployment que se copian en el servidor de aplicaciones.
- Las unidades de deployment se llaman **módulos**.
- Cada módulo corresponde a un tipo de contenedor (ej. un módulo Web corresponde a un contenedor Web, un módulo EJB corresponde a un contenedor EJB).
- Cada módulo empaqueta uno o más componentes del mismo tipo de contenedor y un descriptor de deployment (opcional).

Java EE – Deployment - Ejemplo

- Los módulos Web se empaquetan en archivos WAR con esta estructura.



Servlets

- Objetos Java que pueden atender pedidos HTTP realizados a un servidor Web y enviar contenido generado dinámicamente.
- Son invocados a través de contenedores Web JEE que manejan su ciclo de vida (nunca se hace un *new* de un servlet directamente)
- Utilizan el paquete `javax.servlet.*`
- Extienden la clase `HttpServlet` (que a su vez extiende `GenericServlet`).
- Utilizan la anotación `@WebServlet` (permite mapear con URL)
- Utilizan las clases `HttpServletRequest` y `HttpServletResponse` para encapsular los pedidos y respuestas HTTP.
- Escribe el código HTML de la respuesta en el código Java.

Servlets – Request/Response

Encapsula un Request realizado por un cliente

Permite acceder a los parámetros del mismo

- `getParameter()` retorna valor del parámetro
- `getParameterNames()` retorna los nombres de los parámetros



Servidor Web



Encapsula la respuesta que se le va a dar al cliente

Dos formas de devolver los datos

- Modo texto: a través `getWriter()`
- Binario: a través de `getOutputStream()`

Servlets - Ejemplo

```
@WebServlet(description = "Servlet da la hora", urlPatterns = { "/acaestamiservlet" })  
public class MiServlet extends HttpServlet {  
    public MiServlet() {}  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
        ServletException, IOException {  
        PrintWriter writer = response.getWriter();  
        writer.println("<html>");  
        writer.println("<h1>Hello world!</h1>");  
        writer.append("<p>Context Path: " + request.getContextPath()+"</p>");  
        writer.println("<p>La hora es: " + new Date().toString()+"</p>");  
        writer.println("</html>");  
    }  
}
```

Servlets - Ejemplo

```
<html>  
<h1> Hello world! </h1>  
<p>Context Path: /Servlets</p>  
<p>La hora es: Mon Sep 11 21:52:18 ART  
2017</p>  
</html>
```



Servlets - Sesión

- Los servlets pueden manejar los datos de sesión de un usuario mediante el objeto *HttpSession*.
- El servlet puede obtener la sesión mediante la invocación *getSession* en el cuerpo del *doGet* o *doPost*.
- Si se invoca *getSession(true)* se crea una nueva sesión si no existía.
- Si se invoca *getSession(false)* no se crea una nueva sesión si no existía.
- Con los métodos *setAttribute* y *getAttribute* se puede guardar y recuperar cualquier objeto serializable que quiera mantenerse durante la sesión (Ej: un carrito de compras)

Servlets - Contexto

- Los servlets pueden manejar los datos del contexto de una aplicación mediante el objeto **ServletContext**.
- El servlet puede obtener el contexto mediante la invocación **getServletContext**.
- El contexto es un concepto similar al de sesión pero que es único para todos los clientes.

Servlets – Sesión - Ejemplo

- Contar páginas navegadas

```
public class MainServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

        Integer paginasNavegadas = (Integer)
            req.getSession().getAttribute("paginas_navegadas");

        // no está el atributo: sesión nueva
        if(paginasNavegadas == null) /
            paginasNavegadas = 0;

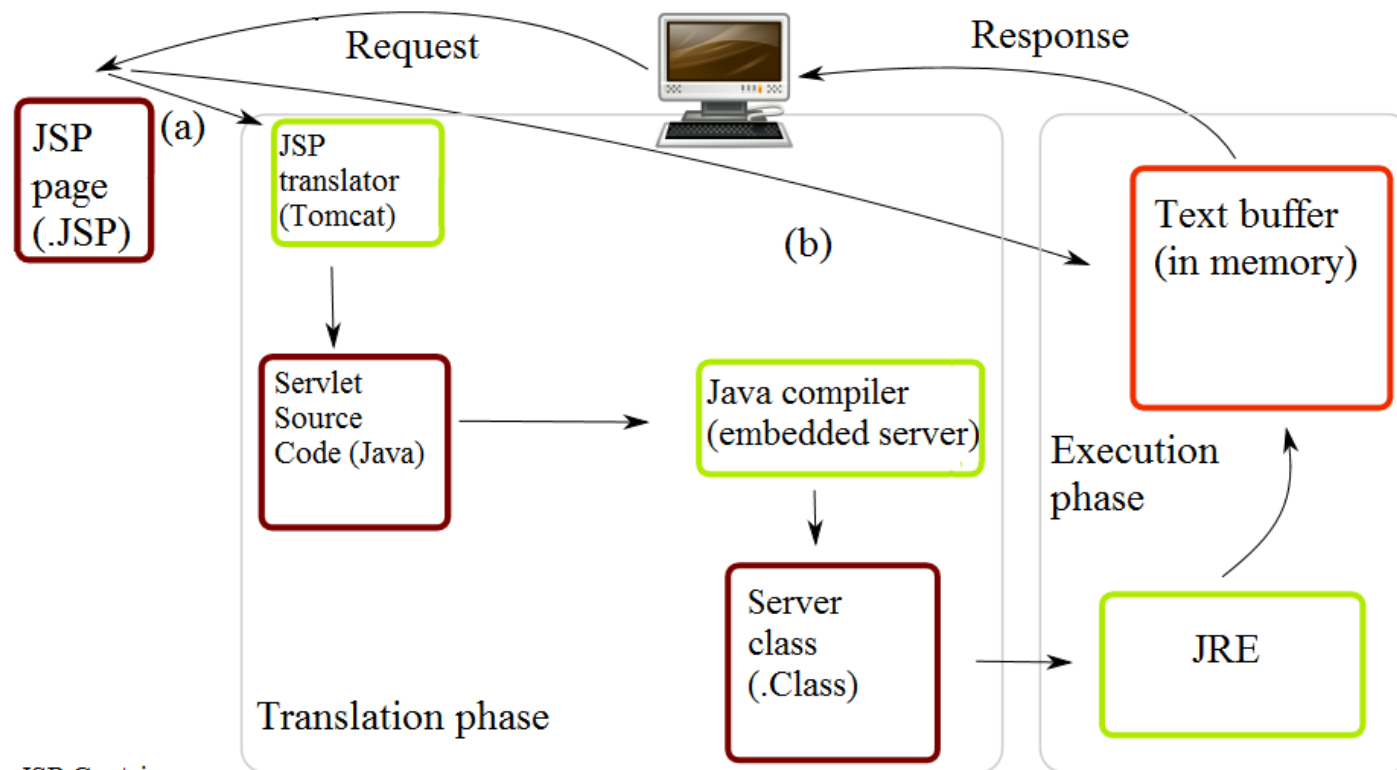
        PrintWriter writer = resp.getWriter();
        writer.printf("Has entrado en %d páginas", paginasNavegadas);

        // Asigna el nuevo valor en la sesión
        ++paginasNavegadas;
        sesion.setAttribute("paginas_navegadas", paginasNavegadas);
    }
}
```

JSP

- Siguiente paso evolutivo para permitir el *server-side scripting* en la plataforma Java.
- Se base en la tecnología de servlets: una JSP genera un servlet en forma transparente para el desarrollador.
- Apunta a separar la lógica de procesamiento de datos de la presentación de los mismos, permitiendo una mejor división del trabajo en programadores Java y diseñadores Web.
- Escribe código Java embebido dentro del contenido estático (documento HTML). Es el enfoque inverso al de servlet.

JSP - Ejecución



JSP Container

(a) Translation occurs at this point, if JSP has been changed or is new.

(b) If not, translation is skipped.

JSP - Directivas

- Comienzan con `<%@`
- Indican al contenedor las reglas que se deben aplicar a la JSP que las contiene.
- No se muestran en la página Web
- Las más utilizadas son `page`, `jsp:include`, `include` y `taglib`.
- Cada directiva tiene ciertos atributos (ej. `import` es un atributo de `page`)

JSP – Directivas - Ejemplos

- Ejemplos:

- importar un paquete de clases Java

`<%@ page import="java.io.*" %>`

- incluir código de otra JSP:

`<%@ jsp:include page="calcularDescuentos.jsp" %>`

- incluir un archivo:

`<%@ include file="cotizaciones.txt" %>`

- mostrar página de error cuando ocurre un excepción:

`<%@ page errorPage="errorPagoOnline.jsp" %>`

JSP - Declaraciones

- Comienzan con `<%!`
- Permiten declarar variables y métodos en Java.
- Las declaraciones se copian en el servlet generado.
- Ejemplos:

`<%! String nombreUsuario; %>`

`<%! private double calcularDeuda(){...} %>`

JSP - Expresiones

- Comienzan con `<%=`
- Permiten incluir expresiones de Java para su evaluación
- El valor del resultado de la evaluación (un String) se inserta en la página Web.
- Ejemplo:
 - expresión

`<%= sueldo*1.25 %>`

se compila en el servlet como

`out.println(sueldo*1.25);`

JSP - Scriptlet

- Comienzan con `<%`
- Permiten añadir código al método `_jspService()` del servlet generado (equivalente a método `service` de `javax.servlet.GenericServlet`)
- Ejemplo:

```
<%
```

```
String autor = application.getInitParameter("Autor");  
if ((autor==null) || (autor.trim().equals("")))) autor="Anónimo";  
out.println("Autor="+autor);  
%>
```

JSP - Acciones

- Utilizan sintaxis XML con etiquetas `<jsp:.../>`
- Permiten realizar acciones dinámicamente (en tiempo de ejecución)
- Ejemplos:
 - Incluir un archivo en tiempo de ejecución (diferente a `<%@ jsp:include>` y `<%@include>`)
`<jsp:include page="header.jsp" />`
 - Redirigir flujo de ejecución conservando pedido y respuesta
`<jsp:forward page="otraPagina.jsp" />`

JSP – Objetos implícitos

- Son objetos que inicializa el contenedor Web y que pueden usarse en la JSP directamente (no se declaran ni se crean explícitamente)
- Ejemplos:
 - request (HttpServletRequest)
`<% email = request.getParameter("email"); %>`
 - response (HttpServletResponse)
`<% response.setCharacterEncoding("UTF-8"); %>`
 - session (HttpSession)
`<% usr=(Usuario)sesión.getAttribute("logueado")%>`
 - application (ServletContext)
`<% usrs=(Collection<Usuario>)application.getAttribute("logueados")%>`

JSP – Objetos implícitos

- Otros ejemplos:

- out (JspWriter) – similar a HttpServletResponse.getWriter()

`<% out.println(new Date()); %>`

- config (ServletConfig) – parámetros del web.xml

`<% nombre = config.getInitParameter("nombre"); %>`

- exception (Throwable) – sólo en página de error

`<%= exception.toString() %>`

- page(Object) – instancia del servlet generado

`<%= page.getClass().getName() %>`

- pageContext(PageContext) – usado con bibliotecas de etiquetas

JSP – Ejemplo

```
<%@page import="java.text.SimpleDateFormat"%>
<%@page contentType="text/html"pageEncoding="UTF-8" %>
<%@page import="java.util.Date" %>
<html>
  <head>
    <jsp:include page="/WEB-INF/template/head.jsp"/>
    <title>Perfil :: gamebook</title>
  </head>
  <body>
    <!-- Incluye otro JSP --%>
    <jsp:include page="/WEB-INF/template/header.jsp"/>

    <% String nombre = "Karl Malone"; %>

    <h2>Información básica</h2>
    <label class="rotulo">Nombre:</label>
    <label class="valor"><%= nombre %></label>
```

Directivas

Acción

Comentario

Declaración

Expresión

JSP – Ejemplo

```
<%  
    Date nacimiento = (Date)  
        request.getAttribute("nacimiento");  
    String nacimientoStr;  
    if(nacimiento != null) {  
        nacimientoStr =  
            new SimpleDateFormat("dd/MM/yyyy").  
                format(nacimiento.getTime().toString());  
    } else {  
        nacimientoStr = "Desconocido";  
    }  
%>  
<label class="rotulo">Fecha de nacimiento:</label>  
<label class="valor"><%= nacimientoStr %></label>  
</body>  
</html>
```

Scriptlet

Expresión

JSP – Manejo de errores

- Paso 1: Utilizar la directiva *page* con el atributo *errorPage* para señalar cuál es la página de error.

```
<%@page import="com.gamebook.model.Usuario"%>
<%@page import="java.util.Collection"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page errorPage="/WEB-INF/errorPages/500.jsp" %>
<!doctype html>
<html>
  <head>
    <jsp:include page="/WEB-INF/template/head.jsp"/>
```

JSP – Manejo de errores

- Paso 2: Utilizar la directiva *page* con el atributo *isErrorPage="true"* para definir una página de error.

```
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<%@ page isErrorPage="true" %>
<html>
<head>
    <link rel="stylesheet" type="text/css"
          href="/media/styles/main.css">
    <title>Error interno en el Servidor</title>

</head>
<body id="error_page">
    <div id="body-container">
        <div class="gameover">500: GAME OVER</div>
```

JSP – Manejo de errores

- Paso 3 (Opcional): Utilizar el objeto implícito *exception* para obtener información sobre la excepción ocurrida.

```
<div class="main" style="color: #000000;">
    <%= exception.getClass().getName() %> :
    <%= exception.getMessage() %>
</div>
</div>
</body>
</html>
```

JSP – Etiquetas personalizadas

- Son una alternativa a los scriptlets que permiten escribir menos código en las JSP.
- Pasos para escribir librerías de etiquetas:
 - Implementar una clase *handler* que extienda la clase *SimpleTagSupport* o implemente la interfaz *SimpleTag*. En el método *doTag* se escribe la lógica que se ejecutará cuando se procese la etiqueta en la JSP.
 - Escribir el descriptor de la librería de etiquetas, que asocia una etiqueta personalizada con el handler anterior. Es un archivo XML con extensión *.tld* que se copia en WEB-INF
 - Agregar la directiva *taglib* y la etiqueta personalizada en la JSP.
- En JSP 2.0 cada etiqueta personalizada puede escribirse en un archivo *.tag* (en WEB-INF/tags) conteniendo la lógica (sin clase handler)

JSP – Etiquetas personalizadas

- AhoraTagHandler.java

```
package uy.edu.fing.tprog;
```

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

```
import java.util.Calendar;
```

```
import javax.servlet.jsp.JspException;
```

```
import javax.servlet.jsp.JspWriter;
```

```
import javax.servlet.jsp.tagext.SimpleTagSupport;
```

```
public class AhoraTagHandler extends SimpleTagSupport {
```

```
    public void doTag() throws JspException, IOException {
```

```
        JspWriter out = getJspContext().getOut();
```

```
        out.print(Calendar.getInstance().getTime());
```

```
    }
```

```
}
```

JSP – Etiquetas personalizadas

- mytags.tld

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib>
<tlib-version>1.0</tlib-version>
<jsp-version>1.2</jsp-version>
<uri>localhost:8080/mytags</uri>
<tag>
<name>ahora</name>
<tag-class>uy.edu.fing.tprog.AhoraTagHandler</tag-class>
<body-content>empty</body-content>
</tag>
</taglib>
```


JSP – Etiquetas personalizadas

- ahora.jsp

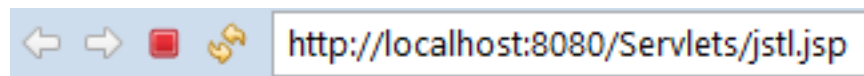
```
<html>
<head>
<%@ taglib uri="localhost:8080/mytags" prefix="tt"%>
</head>
<body>
Fecha y hora actual:<tt:ahora />
</body>
</html>
```

JSTL

- **JavaServer Pages Standard Tag Library**
- **Provee un conjunto de etiquetas que incluyen:**
 - **Un iterador (forEach)**
 - **Instrucción if**
 - **Etiquetas para procesar XML**
 - **Etiquetas para ejecutar SQL**
 - **Etiquetas para internacionalización**
 - **Funciones de uso común**
- **Se utilizan desde la JSP de la misma forma que las etiquetas personalizadas.**

JSTL - Ejemplo

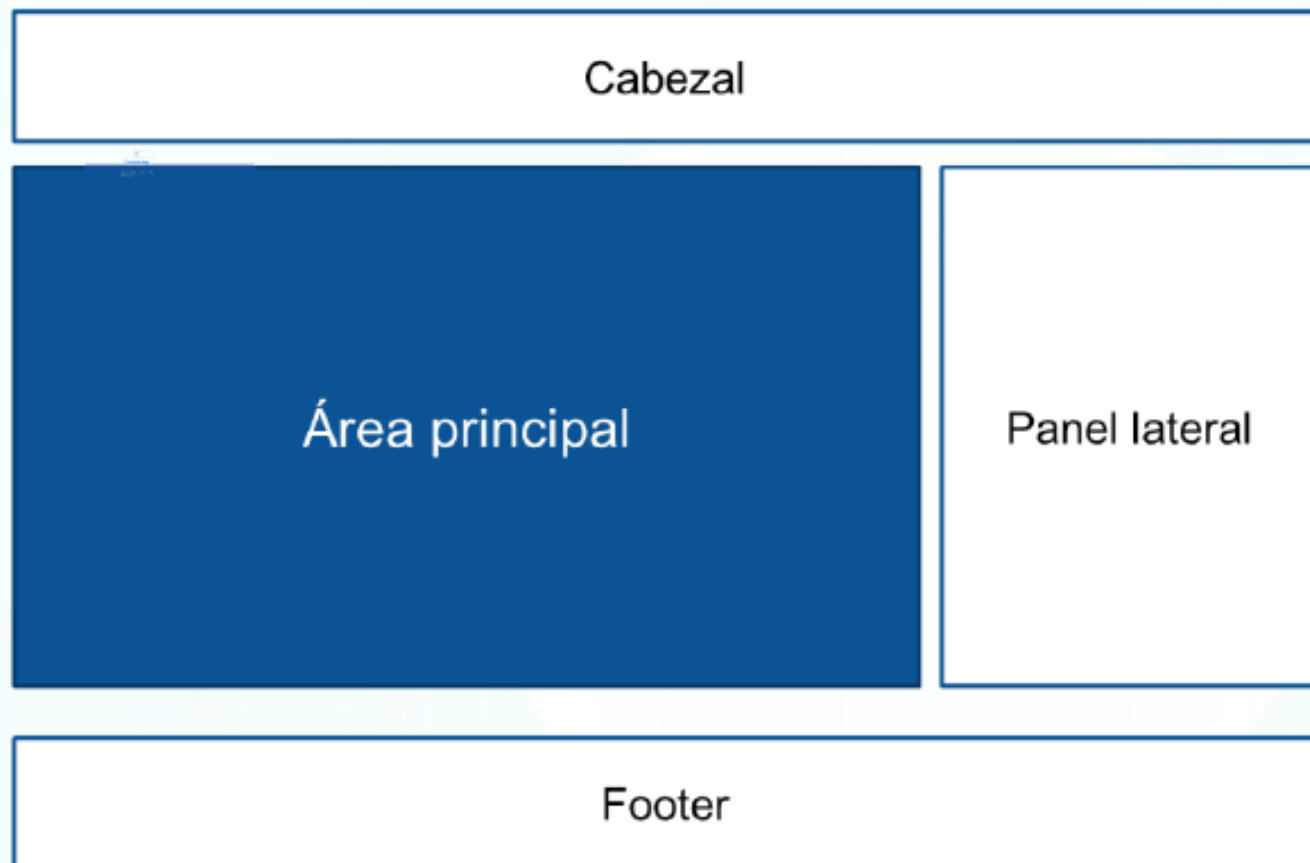
```
<%@ taglib
uri="http://java.sun.com/jsp/jstl/
core" prefix="c" %>
<html>
<head>
<title>Contar hasta 10</title>
</head>
<body>
<c:forEach var="i" begin="1"
end="10" step="1">
Vamos ${i} </br>
</c:forEach>
</body>
</html>
```



Vamos 1
Vamos 2
Vamos 3
Vamos 4
Vamos 5

JSP Templating

- Motivación: Las páginas Web usualmente tienen un esquema común.



JSP Templating

- Una solución: Utilizar templates mediante `jsp:include` para no repetir código.

```
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<html>
  <head>
    <title>feisbook</title>
  </head>
  <body>
    <jsp:include page="header.html"/>
    <jsp:include page="lateral.jsp"/>

    <div>
      <!-- ...
      { Contenido Área principal }
      ... -->
    </div>

    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

Continuará ...

A series of horizontal lines in teal and light blue colors, some solid and some dashed, extending across the bottom of the slide.