

# Sistemas Distribuidos e Internet

Tema 1

*Introducción a JEE*

# Índice General

- Servlets
- JSPs
- Encadenamiento de Servlets y JSPs



# Índice

- Introducción
- Servidor de aplicaciones
- Interfaz
- Ciclo de vida
- HttpRequest
- Gestión de la sesión
- Contexto de aplicación

# Introducción: Plataformas Java

- **Java SE** (Java Platform, Standard Edition)
  - Para aplicaciones y applets, núcleo de la especificación de Java 2, máquina virtual, herramientas y tecnologías de desarrollo, librerías, ...
- **Java EE** (Java Platform, Enterprise Edition)
  - Se apoya en Java SE; con el paso del tiempo, algunas APIs de Java EE se pasaron (y quizás se sigan pasando) a Java SE
  - Incluye las especificaciones para **Servlets, JSP, JSF, Beans, ...**  
(<http://www.oracle.com/technetwork/java/javaee/tech/index.html>)
- **Java ME** (Java Platform, Micro Edition)
  - Subconjunto de Java SE para pequeños dispositivos (móviles, PDAs, ...)
- **JavaFX** (JavaFX Script)
  - API para aplicaciones cliente. Permite incluir gráficos, contenidos multimedia, embeber páginas web, controles visuales, ...
- **JDBC** (Java SE)
  - API para acceso a bases de datos relacionales
  - El programador puede lanzar queries (consulta, actualización, inserción y borrado), agrupar queries en transacciones, ...

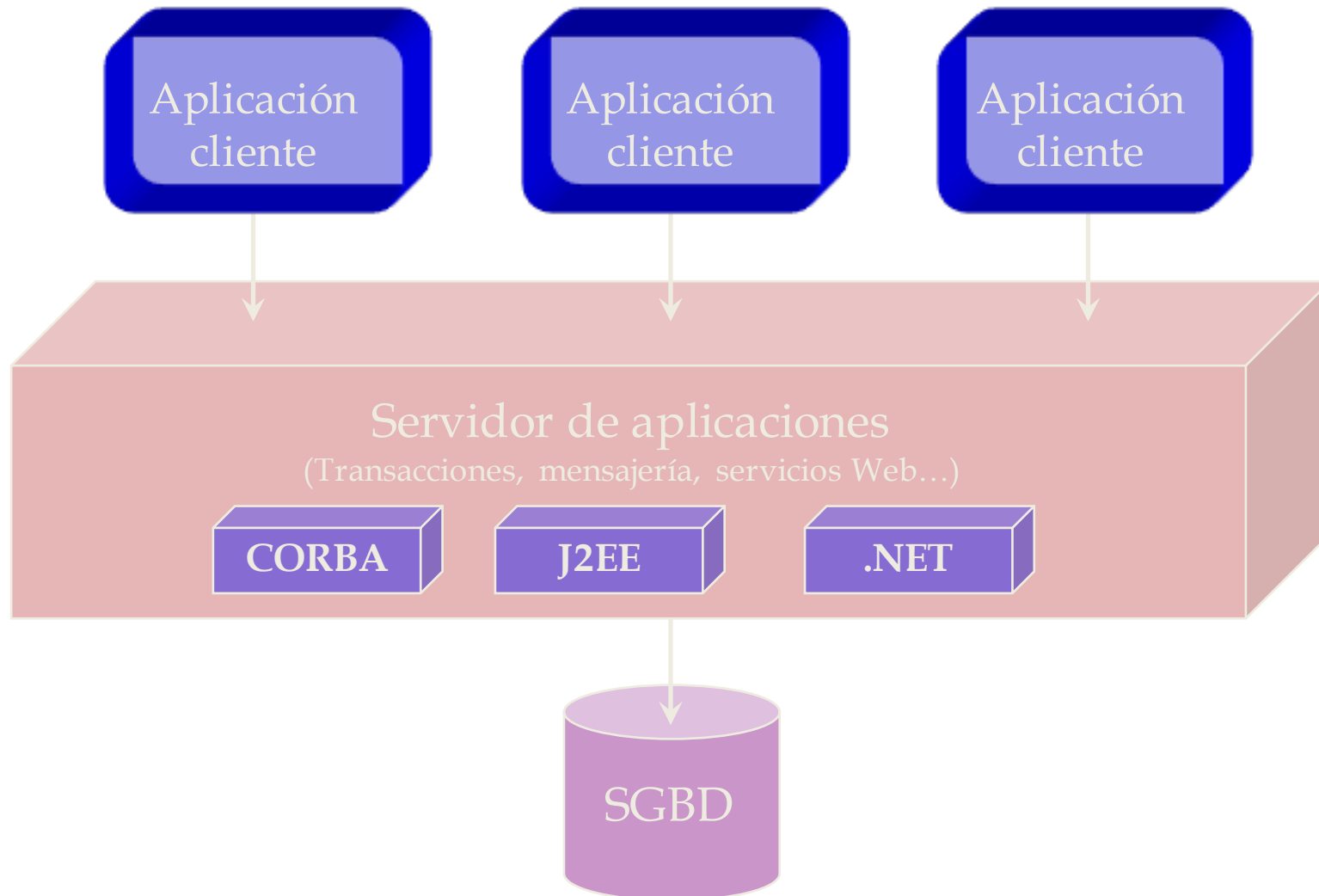
# Introducción: JEE

- Java EE = Java Enterprise Edition
- Especificación de Sun para una plataforma basada en APIs de Java 2 (Java SE) que permiten construir aplicaciones empresariales.
- Las especificaciones suelen ser interfaces y clases abstractas.
- Existen múltiples implementaciones de diversos fabricantes incluso OpenSource: IBM WebSphere, Sun Glasfish, BEA WebLogic, OraclexiAS, Netscape Iplanet, Apache Tomcat (Subproyecto de Jakarta), JBoss
- Una aplicación Java EE no depende de una implementación particular
- Sitio central: **<http://java.sun.com/javaee/index.jsp>**

# Servidores de aplicaciones

- Programa que provee la infraestructura necesaria para aplicaciones web empresariales
  - Los programadores van a poder dedicarse casi en exclusiva a implementar la lógica del dominio
  - Servicios como seguridad, persistencia, transacciones, etc. son proporcionados por el propio servidor de aplicaciones
  - Pieza clave para cualquier empresa de comercio electrónico
- Es una capa intermedia (*middleware*) que se sitúa entre el servidor web y las aplicaciones y bases de datos subyacentes

# Servidores de aplicaciones



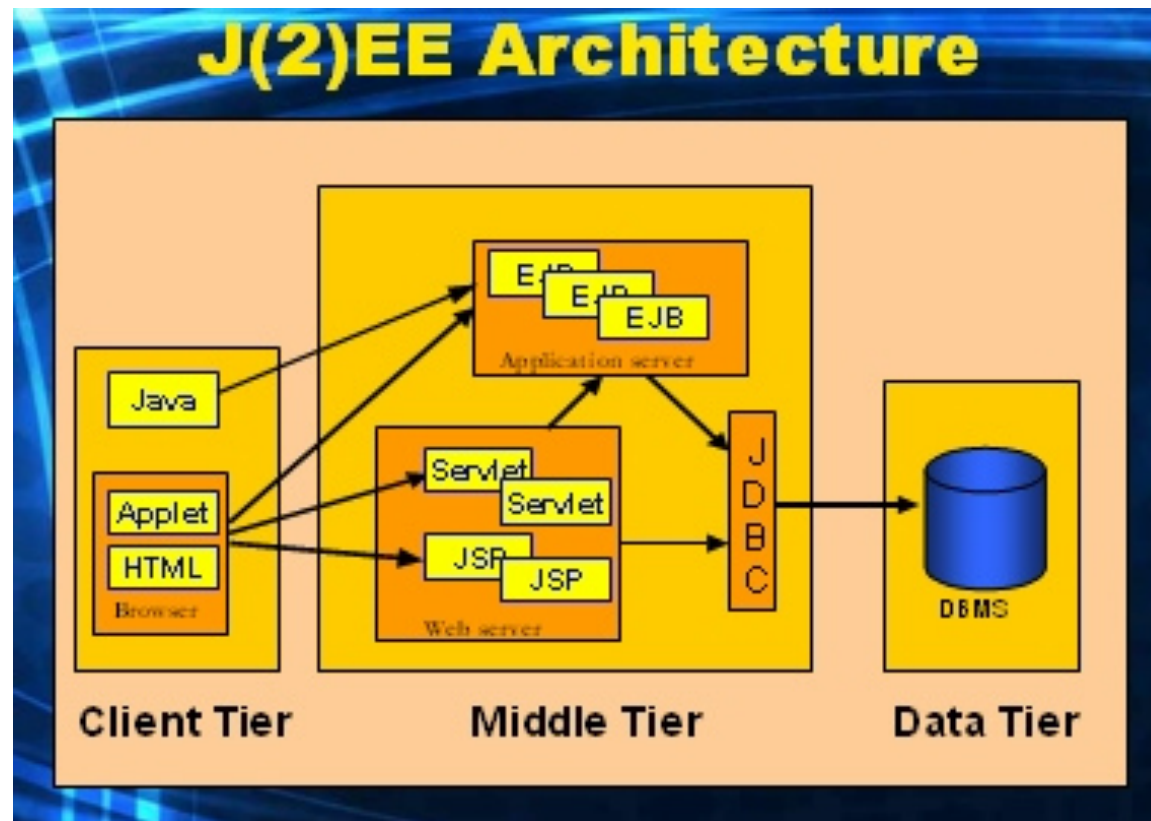


# Servidores de aplicaciones

- Motivación
  - Surgen cuando queda patente que las aplicaciones cliente/servidor no van a ser escalables a un gran número de usuarios
    - Se hace necesario mover las reglas de negocio a algún lugar intermedio entre el cliente y la bases de datos
  - Empiezan a aparecer productos que realizan esta tarea
    - Servidores de transacciones, aplicaciones, etc.
  - Diseñados para gestionar de forma centralizada el modo en que los clientes podían acceder a los servicios con los que tenían que interoperar

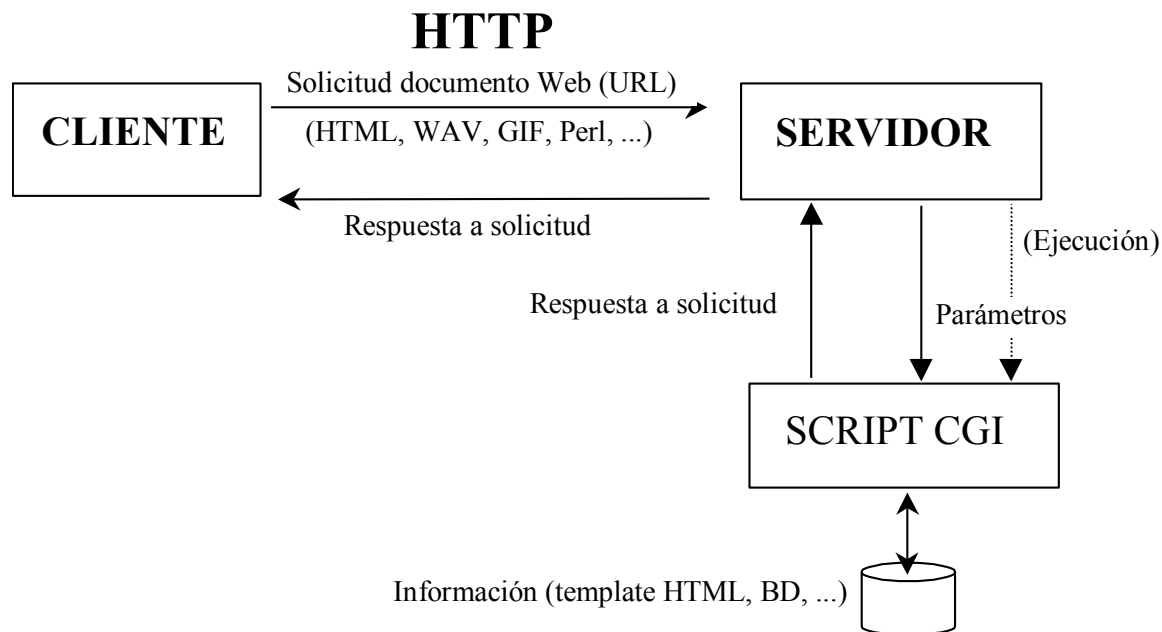
# Servidores de aplicaciones

- Tecnologías actuales
  - JEE y .NET



# CGI: un histórico

- CGI: Common Gateway Interface
- Inicio de la Web: Documentos estáticos
- CGI: Necesario para dotar de dinamismo e interactividad a las aplicaciones web
- Se apoya en los comandos HTTP para establecer la comunicación entre Servidor y aplicación externa (de servidor)



# Introducción. CGI

- Pasarela entre el servidor Web y el script CGI

```
#!/usr/bin/perl  
print "Content-Type: text/html\n\n";  
print "<H1>Este es un script CGI muy sencillo</H1>\n";  
print "¡Este es mi primer script CGI!\n";
```

- *¿Por qué usar CGI?*

- Páginas Web dinámicas

- Actualizar páginas Web que se ven en los browsers
- Fechas, horas, número de accesos...
- Documentos dinámicos: Páginas con información de BD
- Configuración de documentos en función del usuario, del browser, del dominio, devolver diferentes datos cada vez que se llame, ...

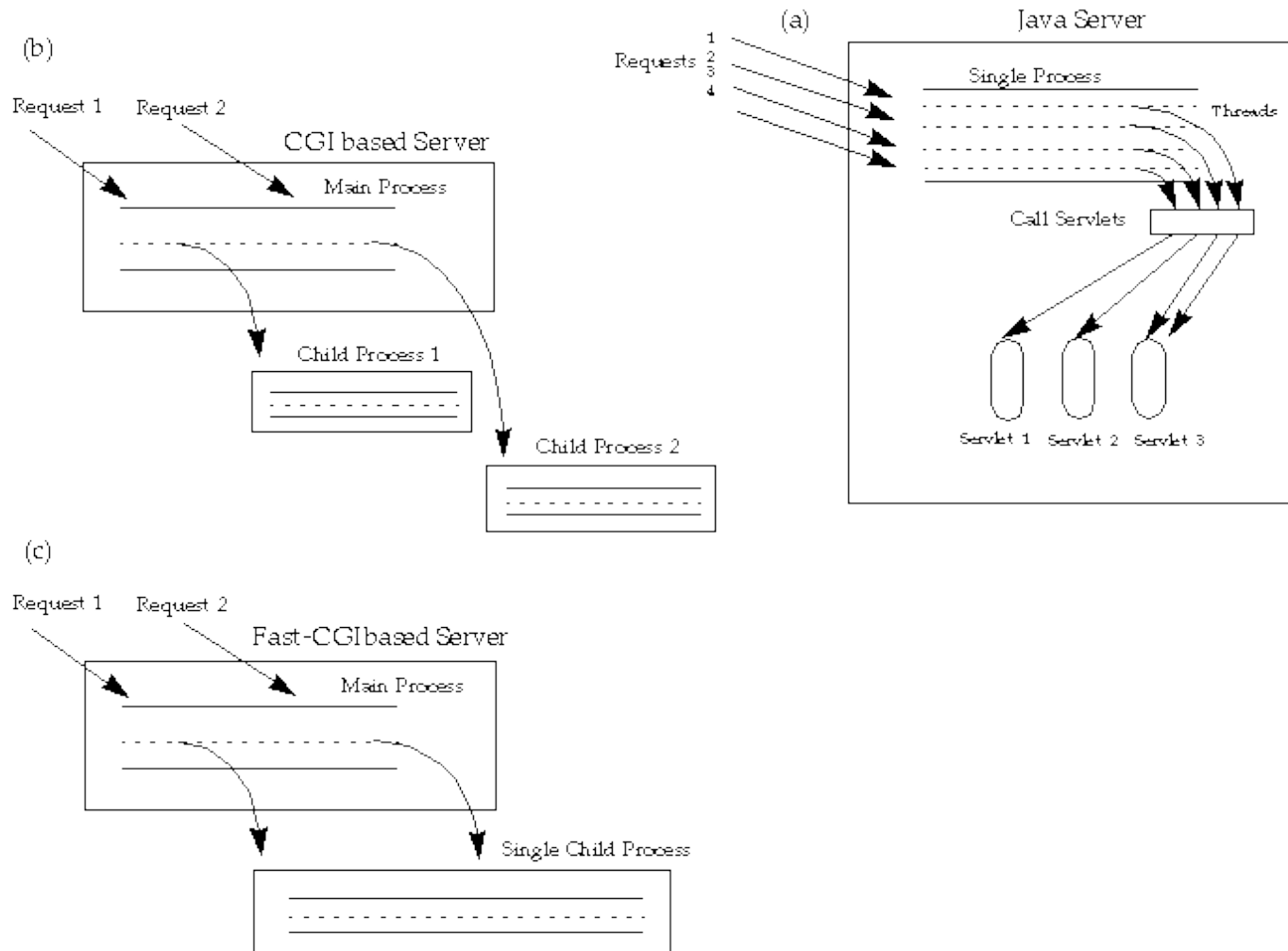
- Páginas Web interactivas

- Recepción de datos del usuario y envío a un script para su gestión
- Ejemplos: Tablones de news, leer, colocar mensajes, ...

# Qué es un servlet

- Módulos/componentes que amplían los servidores orientados a petición/respuesta
- La respuesta en el lenguaje Java a los CGIs (*Common Gateway Interface*) para construir páginas dinámicas
  - Basándose en datos del usuario
  - Utilizando la información que varía en el tiempo
  - Usando información de una base de datos

# Servlets vs. CGIs



# Servlets: Ventajas sobre los CGI

- Eficiencia
  - JVM
- Facilidad de uso y aprendizaje
- Potentes
  - Comunicación directa con el servidor
- Portables
- Las del Lenguaje Java

# Contenedor de servlets

- Un contenedor define un ambiente estandarizado de ejecución que provee servicios específicos a los servlets
  - Por ejemplo, dan servicio a las peticiones de los clientes, realizando un procesamiento y devolviendo el resultado
- Los servlets tienen que cumplir un contrato con el contenedor para obtener sus servicios
  - Los contratos son interfaces Java
    - Por ejemplo, la interfaz *Servlet*



# Servlet “HolaMundo.java”

```
import java.io.*;           // Necesario importar estos tres paquetes
import javax.servlet.*;
import javax.servlet.http.*;

public class HolaMundo extends HttpServlet { // Herencia de HttpServlet

    // sustituir métodos doGet() y/o doPost().
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        // Incluir excepciones generadas por doGet() y doPost()

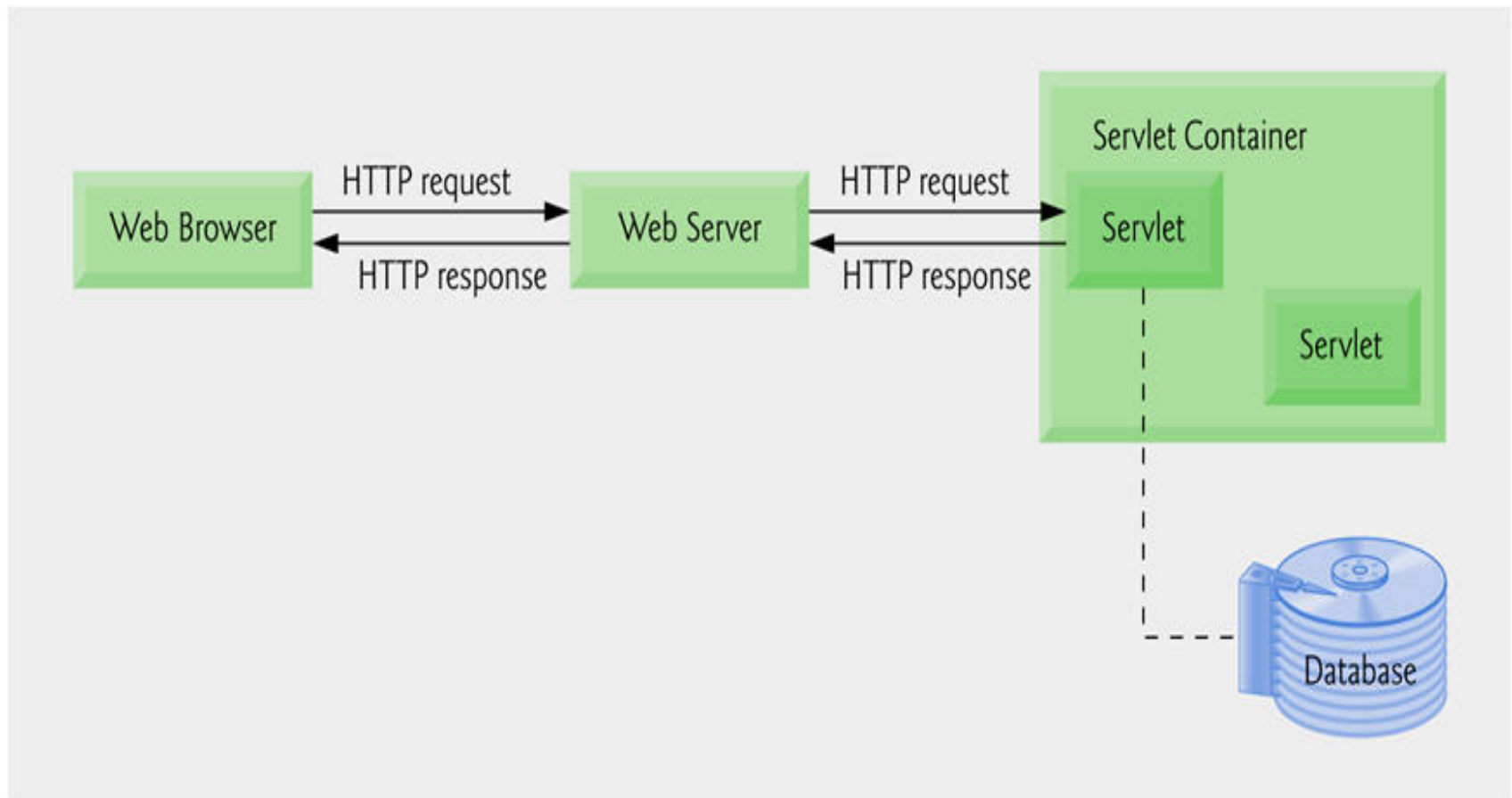
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<html>"); out.println("<head>");
        out.println("<title> Servlet Hola Mundo </title>");
        out.println("</head>");

        out.println("<body>");
        out.println("<h1> Hola Mundo!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

- HttpServletRequest (Datos de entrada)
  - Variables de Formulario
  - Cabeceras de solicitud HTTP (CGI)
  - Datos del servidor (CGI)
- HttpServletResponse (Datos de salida)
  - Códigos de estado
  - Encabezados de respuesta: Content-type, Set-Cookie...
  - Obtención de un objeto PrintWriter para la respuesta
- Derivar de la clase GenericServlet (clase base de HttpServlet) para usar p.e. en un servidor de Correo o FTP

# Aplicaciones con Servlets



# Servlet API

A diagram of the Servlet API. It shows a package icon (a circle with an 'I') next to the word 'Servlet'. To the right of 'Servlet' is a vertical list of five methods, each preceded by a green circle and a dotted line. The methods are: init(ServletConfig), getServletConfig(), service(ServletRequest, ServletResponse), getServletInfo(), and destroy().

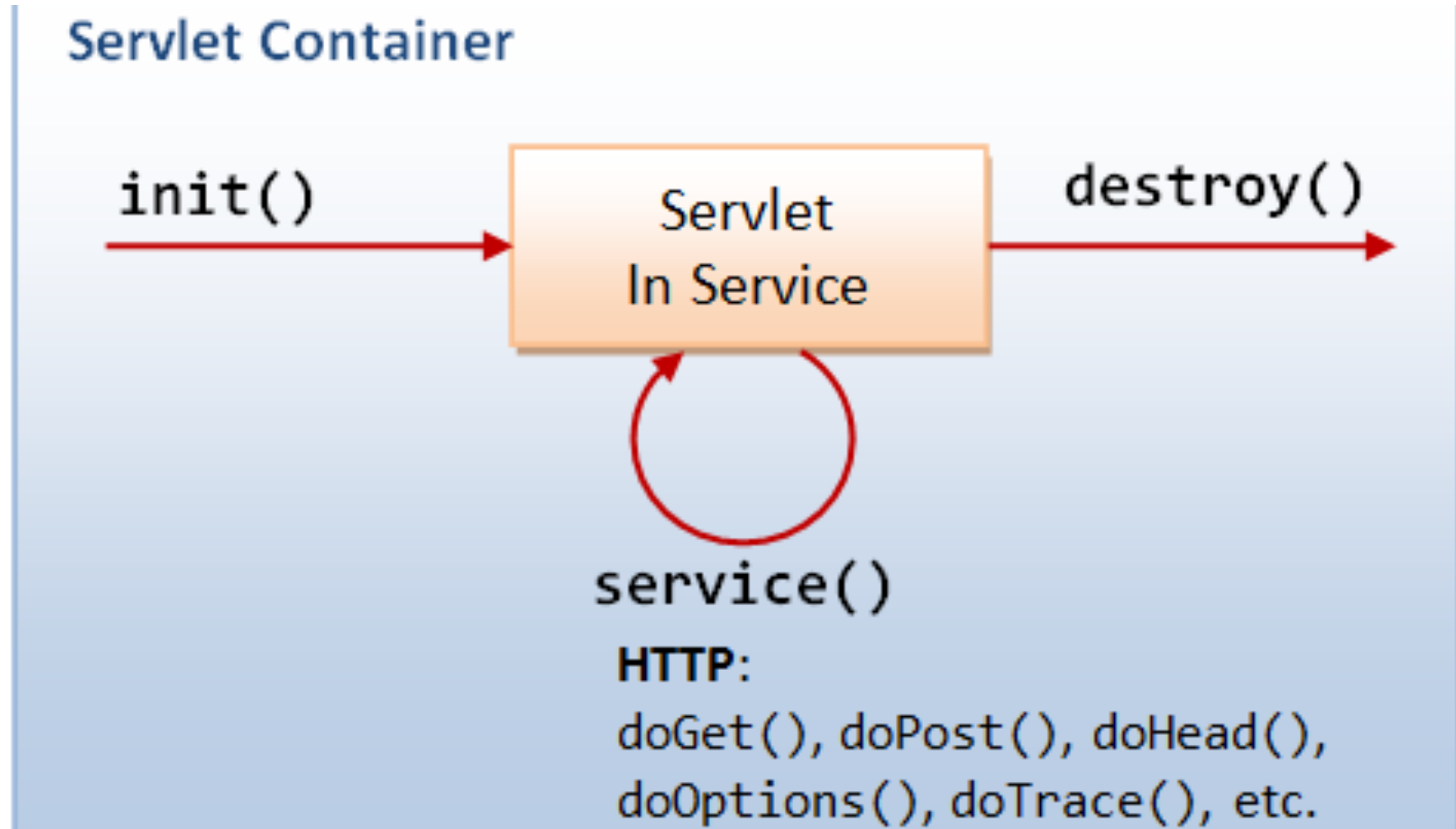
```
import java.io.IOException;
import javax.servlet.*;

public class MyServlet extends GenericServlet {

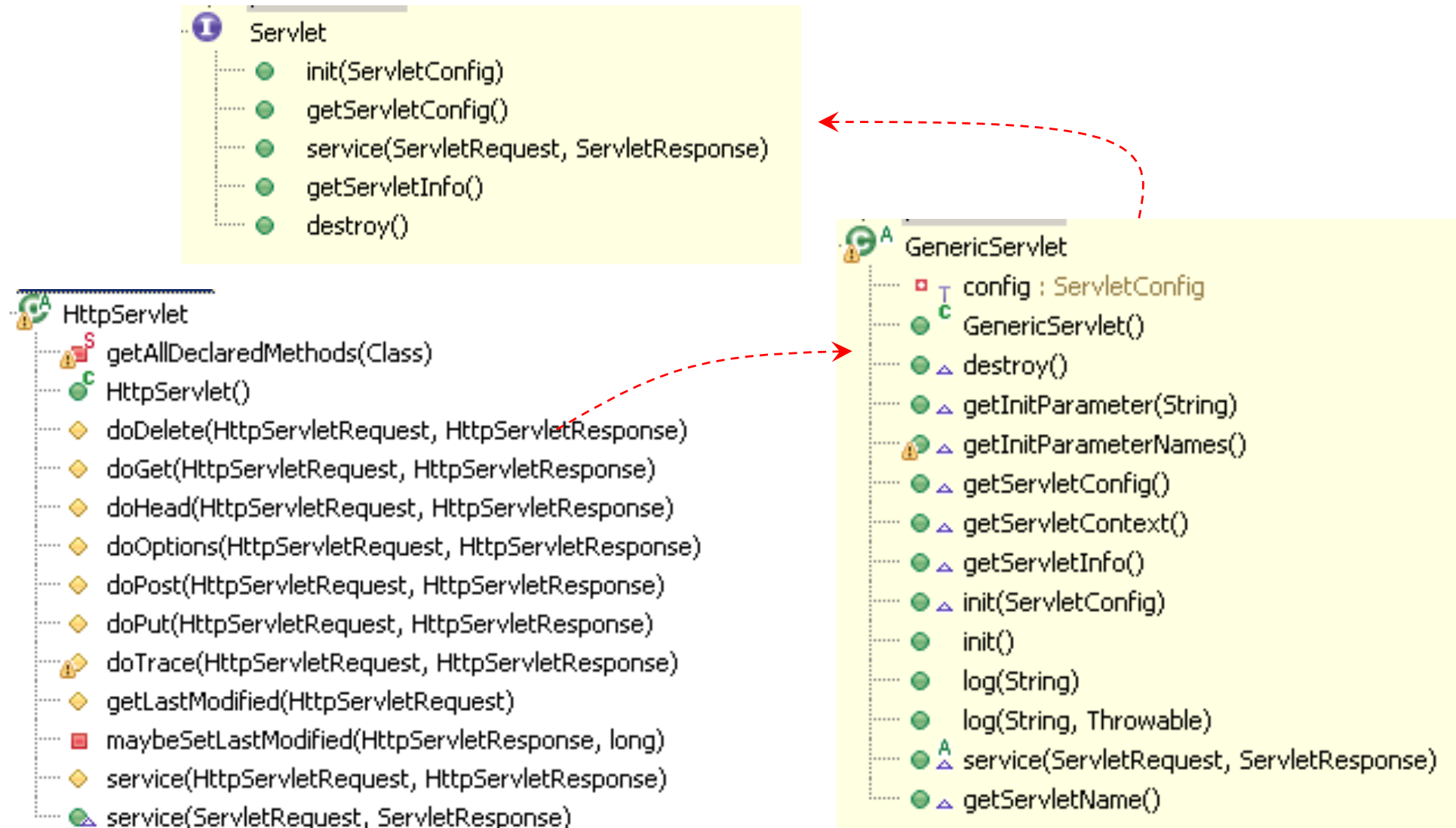
    public void service (
        ServletRequest request,
        ServletResponse response)
        throws ServletException, IOException {
        //...
    }
    //...
}
```

SDI - Introducción a JEE

# Servlet Ciclo de vida



# Métodos y herencia en Servlet



# Tipos de peticiones HTTP

Un navegador puede enviar la información al servidor de varias formas:

- **GET:** Paso de parámetros en la propia URL de acceso al servicio o recurso del servidor. Método “doGet” del servlet
- **POST:** Lo mismo que GET pero los parámetros no van en la línea de URL sino en otra línea a parte. El manejo es idéntico. Método “doPost” del servlet.
- **PUT, ...**

# Método service() en HttpServlet

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

    String method = req.getMethod();

    if (method.equals(METHOD_GET)) {
        doGet(req, resp);

    } else if (method.equals(METHOD_HEAD)) {
        doHead(req, resp);

    } else if (method.equals(METHOD_POST)) {
        doPost(req, resp);

    } else if (method.equals(METHOD_PUT)) {
        doPut(req, resp);

    } else if (method.equals(METHOD_DELETE)) {
        doDelete(req, resp);

    } else if (method.equals(METHOD_OPTIONS)) {
        doOptions(req, resp);

    } else if (method.equals(METHOD_TRACE)) {
        doTrace(req, resp);
    }
}
```

Separa la petición  
en función del  
método HTTP

# Servlets: Métodos doGet y doPost

- Son llamados desde el método `service()`
- Reciben interfaces instanciadas:
  - “`HttpServletRequest`” canal de entrada con información enviada por el usuario
  - “`HttpServletResponse`” canal de salida (contenido web)

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, java.io.IOException {
    . . .
}

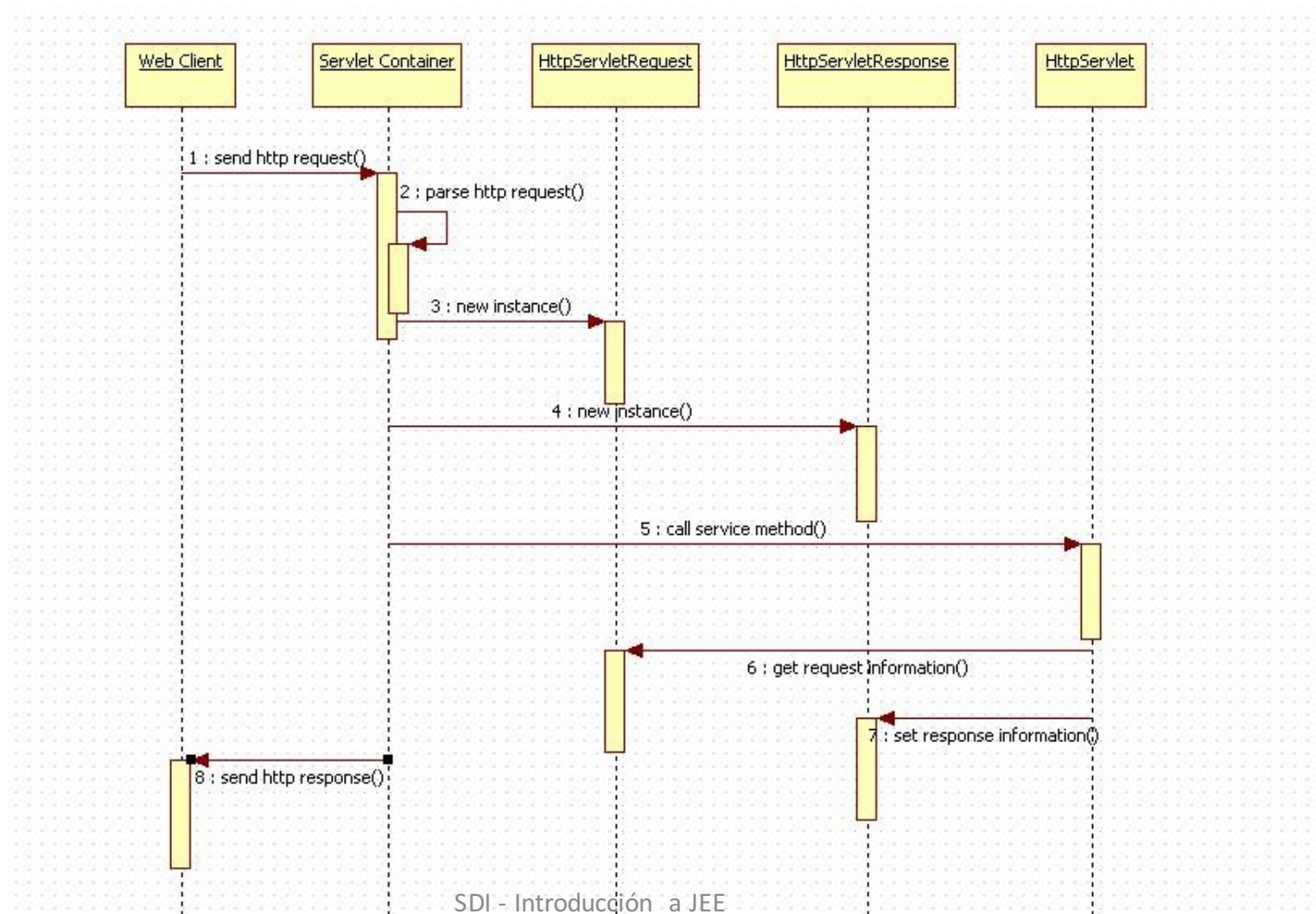
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, java.io.IOException {
    . . .
}
```

`doGet()` → consulta (no cambia estado, idempotente)

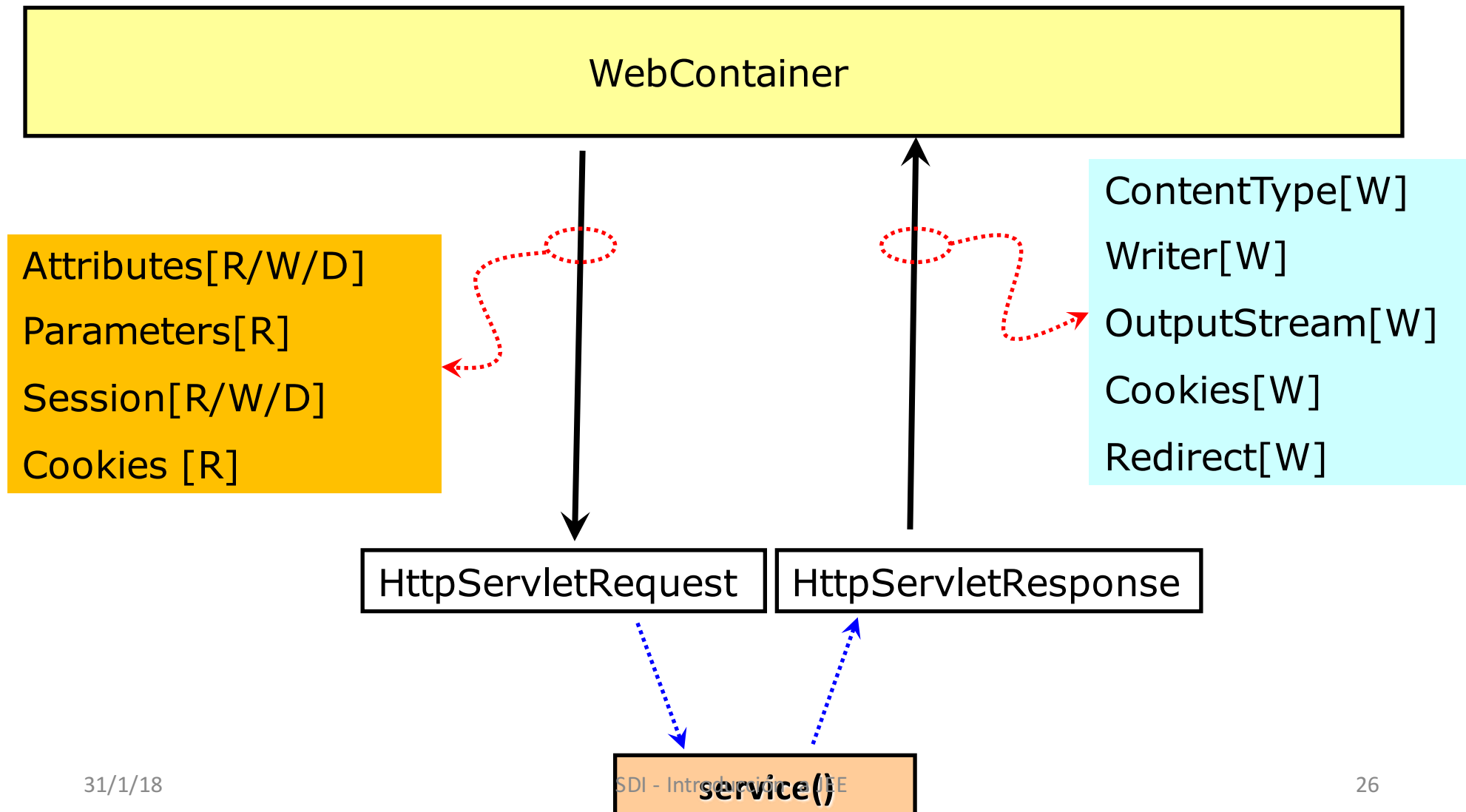
`doPost()` → modifica datos (cambia estado)



# Servlets: diagrama de secuencia



# Modelo de datos desde service()



# Servlets: Respondiendo en HTML

- La salida del servlet será, habitualmente, un documento HTML
- 2 pasos:
  - Indicar en la cabecera de la respuesta el tipo de contenido que vamos a retornar
    - El caso más habitual será devolver HTML
      - Aunque también podemos devolver otra cosa: una imagen generada, xml, etc.
  - Al ser un proceso tan común existe un método que nos lo soluciona directamente:
    - “setContentType” de
    - “HttpServletResponse”
  - Crear y enviar código HTML válido
  - Ej: **HolaMundoServlet**

# HolaMundo Servlet

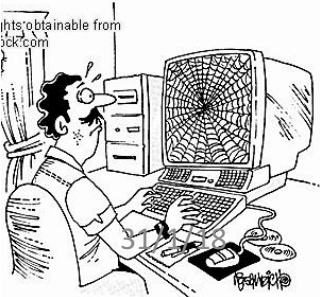
```
public class HelloWorld extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
  
        response.setContentType("text/html");  
  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD><TITLE>Hola Mundo!</TITLE></HEAD>");  
        out.println("<BODY>");  
        out.println("Bienvenido a mi primera página Güev!");  
        out.println("</BODY></HTML>");  
    }  
  
    protected void doPost(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
  
        doGet(request, response);  
    }  
}
```

# Servlets: registro en el descriptor de despliegue

- Insertamos en **web.xml** la declaración del servlet y del servlet-mapping

```
<servlet>  
  <servlet-name>HolaMundo</servlet-name>  
  <servlet-class>uo.sdi.servlet.HolaMundoServlet</servlet-class>  
</servlet>
```

```
<!-- Standard Action Servlet Mapping -->  
<servlet-mapping>  
  <servlet-name>HolaMundo</servlet-name>  
  <url-pattern>/HolaMundoCordial</url-pattern>  
</servlet-mapping>
```



<http://<server>/HolaMundoCordial>

# Servlets: registro con anotaciones

- Insertamos antes de la clase Servlet la anotación con el nombre del servlet y el url de mapeo.

```
@WebServlet(name = "HolaMundo", urlPatterns = { "/HolaMundoCordial" })
```

```
public class HolaMundoServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L;
```

```
    /**
```

```
     * @see HttpServlet#HttpServlet()
```

```
     */
```

```
    public HolaMundoServlet() {
```

```
        super();
```

```
        // TODO Auto-generated constructor stub
```

```
    }
```

```
.....
```



```
http://<server>/HolaMundoCordial
```

# HttpServletRequest: Recogiendo información de usuario

- Los parámetros nos llegan en la **request**
  - Request es tipo “*HttpServletRequest*”
  - Método *getParameter(<nombre>)*
- Son los parámetros de la QueryString o de los campos del formulario
- Siempre nos devuelve objetos de tipo **String**
  - Habrá que hacer las conversiones que proceda

**Object HttpServletRequest.getParameter(nombre)**  
devuelve:

- "" (si no hay valor)
- null (si no existe el parámetro)
- El valor en caso de haber sido establecido

# Formularios HTML y request.getParameter()

```
<form ACTION="http://server/app/HelloWorld" METHOD="POST">  
  Nombre: <INPUT TYPE="TEXT" NAME="NombreUsuario" SIZE="15">  
  <INPUT TYPE="Submit" VALUE="Aceptar">  
</FORM>
```

```
public class HelloWorld extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
        String nombre = (String) request.getParameter("NombreUsuario");  
  
        response.setContentType("text/html");  
  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD><TITLE>Hola Mundo!</TITLE></HEAD>");  
        out.println("<BODY>");  
        out.println("Bienvenido " + nombre);  
        out.println("</BODY></HTML>");  
    }  
}
```



# Ejemplo Servlet con parámetros:

## HolaMundo personalizado

- Creamos index.html, página con un formulario que nos pasa el parámetro “**Nombre**”:

```
<HTML><HEAD> <TITLE>Mi primer formulario</TITLE> </HEAD>
<BODY>
  <FORM ACTION="http://127.0.0.1:8080/sdi/HolaMundoCordial"
    METHOD="POST">
    <CENTER><H1>Saludador</H1></CENTER>
    <HR><BR>
    <TABLE ALIGN="CENTER">
      <TR>
        <TD ALIGN="RIGHT">¿C&ocirc;mo te llamas?</TD>
        <TD><INPUT TYPE="TEXT" NAME="NombreUsuario"
          ALIGN="LEFT" SIZE="15"></TD>
      </TR>
      <TR>
        <TD><INPUT TYPE="Submit" VALUE="Saluda!"> </TD>
      </TR>
    </TABLE>
  </FORM>
</BODY>
</HTML>
```

# Ejemplo Servlet con parámetros:

## HolaMundo personalizado

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HolaMundoServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse
        res) throws ServletException, IOException {
        ...
        String nombre = (String) req.getParameter("NombreUsuario");
        ...
        if ( nombre != null )
            out.println("<br>Hola "+nombre+"<br>");
        ...
    }
}
```

# Servlets: Políticas de acceso concurrente (threading)

- Los servlets están diseñados para soportar múltiples accesos simultáneos por defecto
- **¡Ojo!** El problema puede surgir cuando se hace uso de un **recurso compartido**:
  - Ejemplo: abrimos un fichero desde un servlet
  - Solución:
    - Hacer que el recurso sea el que defina la política de acceso concurrente
      - Ejemplo: las bases de datos están preparadas para ello

# Servlets: Ciclo de vida

## INICIALIZACIÓN:

- Una única llamada al metodo “init” por parte del contenedor de servlets  
`public void init(ServletConfig config) throws ServletException`
- Se pueden recoger unos parametros concretos con “getInitParameter” de “ServletConfig”. Estos parámetros se especifican en el descriptor de despliegue de la aplicación: **web.xml**

## PETICIONES

- La primera petición a init se ejecuta en un thread que invoca a service
- El resto de peticiones se invocan en un nuevo hilo mapeado sobre service

## DESTRUCCIÓN:

- Cuando todas las llamadas desde el cliente cesen o un temporizador del servidor así lo indique. Se deben liberar recursos retenidos desde init()  
`public void destroy()`

# Gestión de la Sesión. Mantenimiento del estado de la sesión.

- El protocolo HTTP no mantiene estado
- Complica la tarea de guardar las acciones (por ejemplo, en una tienda virtual) de un usuario
- Posibles soluciones:
  - Cookies
  - Añadir información en la URL
  - Usar campos ocultos de formularios (HIDDEN)
  - Empleo del objeto `HttpSession` del servlet

(Ejemplo: Carrito de la Compra)

# Servlets: Seguimiento de sesión

- Los servlets proporcionan una solución técnica
  - La API **HttpSession**
- Una interfaz de alto nivel construida sobre las cookies y la reescritura de URLs
  - Pero transparente para el desarrollador
- Permite almacenar objetos

# Servlets: Seguimiento de sesión

- Para buscar el objeto HttpSession asociado a una petición HTTP:
  - Se usa el método “getSession()” de “HttpServletRequest” que devuelve null si no hay una sesión asociada
    - En este último caso podríamos crear un objeto HttpSession
    - Pero, al ser una tarea sumamente común, se puede pasar al método un argumento con valor true y él mismo se encarga de crear un objeto HttpSession si no existe

# Interfaz HttpSession

HttpSession	
	getCreationTime()
	getId()
	getLastAccessedTime()
	getServletContext()
	setMaxInactiveInterval(int)
	getMaxInactiveInterval()
	getSessionContext()
	getAttribute(String)
	getValue(String)
	getAttributeNames()
	getValueNames()
	setAttribute(String, Object)
	putValue(String, Object)
	removeAttribute(String)
	removeValue(String)
	invalidate()
	isNew()



Deprecated methods



# Servlets: Seguimiento de sesión

- Añadir y recuperar información de una sesión
  - **getAttribute("nombre\_variable")**
    - Devuelve una instancia de `Object` en caso de que la sesión ya tenga *algo* asociado a la etiqueta **nombre\_variable**
    - `null` en caso de que no se haya asociado nada aún
  - **setAttribute("nombre\_variable", referencia)**
    - Coloca el objeto referenciado por **referencia** en la sesión del usuario bajo el nombre **nombre\_variable**. A partir de este momento, el objeto puede ser recuperado por este mismo usuario en sucesivas peticiones. Si el objeto ya existiera, **lo sobrescribe**
  - **getAttributeNames()**
    - Retorna una *Enumeration* con los nombres de todos los atributos establecidos en la sesión del usuario

# Servlets: Seguimiento de sesión

- **getId()**
  - Devuelve un identificador único generado para cada sesión
- **isNew()**
  - True si el cliente (navegador) nunca ha visto la sesión. False para sesión preexistente
- **getCreationTime()**
  - Devuelve la hora, en milisegundos desde 1970, en la que se creó la sesión
- **getLastAccessedTime()**
  - La hora en que la sesión fue enviada por última vez al cliente

# Servlets: Seguimiento de sesión

- Caducidad de la sesión
  - Peculiaridad de las aplicaciones web
    - **No sabemos cuándo se desconecta el usuario del servidor**
  - Automáticamente el servidor web invalida la sesión tras un periodo de tiempo (p.e., 30') sin peticiones o manualmente usando el método `invalidate`

**¡OJO!**

**¡SOBRECARGAR LA SESIÓN ES PELIGROSO!**

Los elementos almacenados no se liberan hasta que no salta el timeout o `session.invalidate()`

# Servlets: Contexto de la aplicación

- Se trata de un saco “**común**” a todas las sesiones de usuario activas en el servidor
- Nos permite compartir información y objetos entre los distintos usuarios
- Se accede por medio del objeto “`ServletContext`”

```
public ServletContext getServletContext()
```

(Ejemplo: Contador de Visitas)



Se hereda de `GenericServlet`

# Interfaz ServletContext()

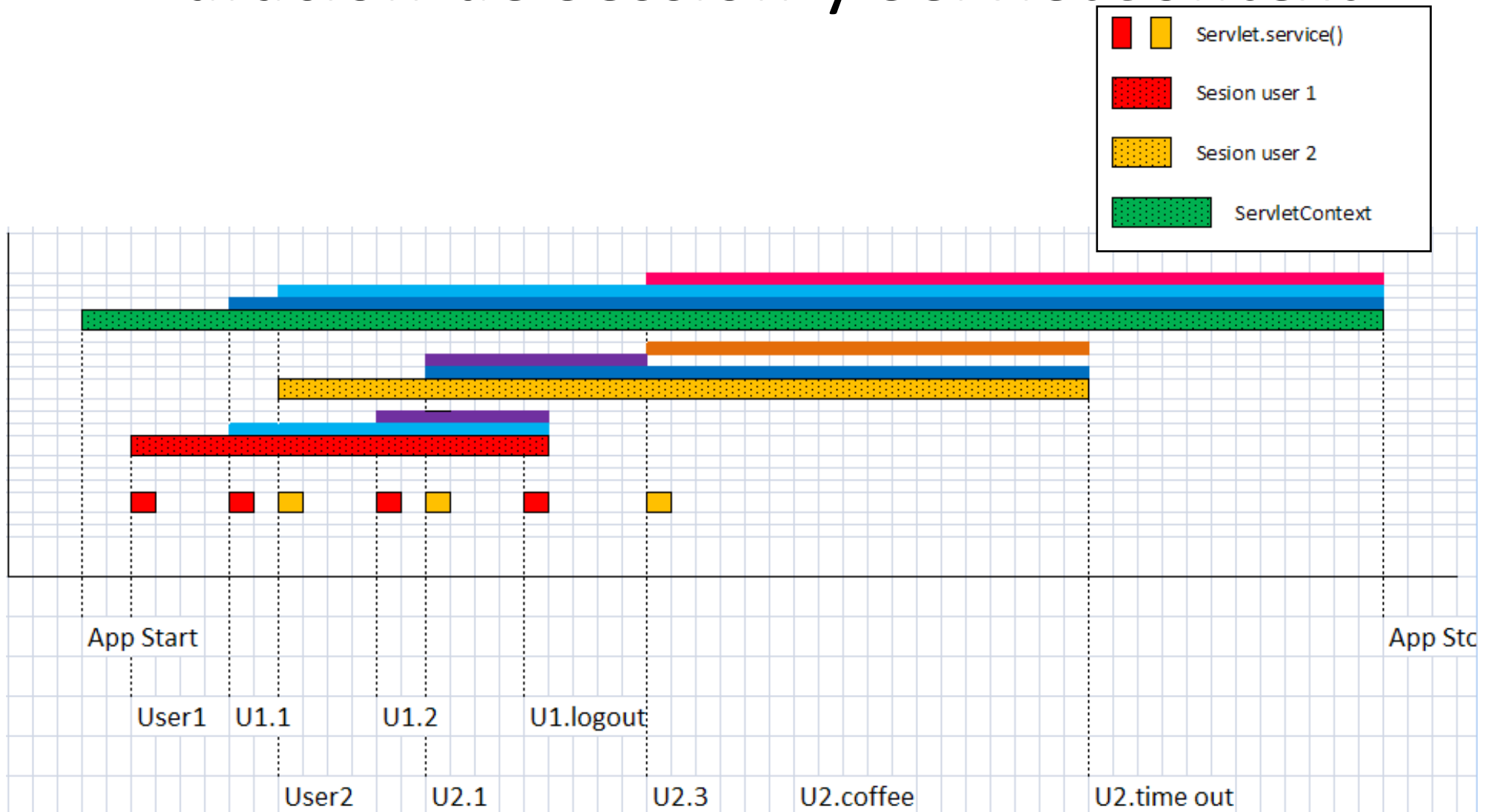
ServletContext	
●	getContext(String)
●	getContextPath()
●	getMajorVersion()
●	getMinorVersion()
●	getMimeType(String)
●	getResourcePaths(String)
●	getResource(String)
●	getResourceAsStream(String)
●	getRequestDispatcher(String)
●	getNamedDispatcher(String)
●	getServlet(String)
●	getServlets()
●	getServletNames()
●	log(String)
●	log(Exception, String)
●	log(String, Throwable)
●	getRealPath(String)
●	getServerInfo()
●	getInitParameter(String)
●	getInitParameterNames()
●	getAttribute(String)
●	getAttributeNames()
●	setAttribute(String, Object)
●	removeAttribute(String)
●	getServletContextName()

# Servlets: Contexto de la aplicación

- Para colocar o recuperar objetos del contexto...
  - **Añadir un atributo**
    - Se usa el método `setAttribute` de `ServletContext`
    - El control sobre varios servlets manipulando un mismo atributo es responsabilidad del desarrollador
  - **Recuperar un atributo**
    - Se usa el método `getAttribute` de `ServletContext`
    - Hay que convertir el objeto que devuelve al tipo requerido (retorna un `Object`)

Ejemplo: Contador de Visitas

# Duración de Session y ServletContext







# Índice

- Introducción
- Elementos de JSP
  - Elementos de secuencias/Scripting
  - Directivas
  - Acciones

# ¿Qué es JSP?

- JSP = Java Server Pages
- Una tecnología para crear páginas Web dinámicas
  - Contienen código HTML normal junto a elementos especiales de JSP
- Están construidas sobre **servlets**
  - Cuando se solicita una página JSP, la primera vez se compilará el código Java a un Servlet y se cargará en el Servidor de Servlets. El código HTML se adjuntará al código de salida del método service() del Servlet.
- Vienen a resolver el problema de aquellos (que también tenían los CGI):
  - Generar HTML directamente por código
    - Dificulta enormemente la **separación de tareas** entre **diseñadores y programadores**

# Ejemplo de página JSP

```
<html>
  <head>
    <title>Saludo personalizado con JSP</title>
  </head>
  <body>

    <% java.util.Date hora = new java.util.Date(); %>

    <% if (hora.getHours() < 12) { %>
      <h1>¡Buenos días!</h1>
    <% } else if (hora.getHours() < 21) { %>
      <h1>¡Buenas tardes!</h1>
    <% } else { %>
      <h1>¡Buenas noches!</h1>
    <% } %>

    <p>Bienvenido a nuestro sitio Web, abierto las 24 horas del día.</p>

  </body>
</html>
```

```

public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws IOException, ServletException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<html>");
    out.println("<head>");
    out.println("<title>Saludo personalizado con JSPout.println</title>");
    out.println("</head>");
    out.println("<body>");

    java.util.Date hora = new java.util.Date();
    if (hora.getHours() out.println("< 12) {
        out.println("<h1>;Buenos días!</h1>");
    }
    else if (hora.getHours() out.println("< 21) {
        out.println("<h1>;Buenas tardes!</h1>");
    }
    else {
        out.println("<h1>;Buenas noches!</h1>");
    }

    out.println("<p>Bienvenido a nuestro sitio Web, abierto las 24 horas del día.</p>");
    out.println("</body>");
    out.println("</html>");
}

```

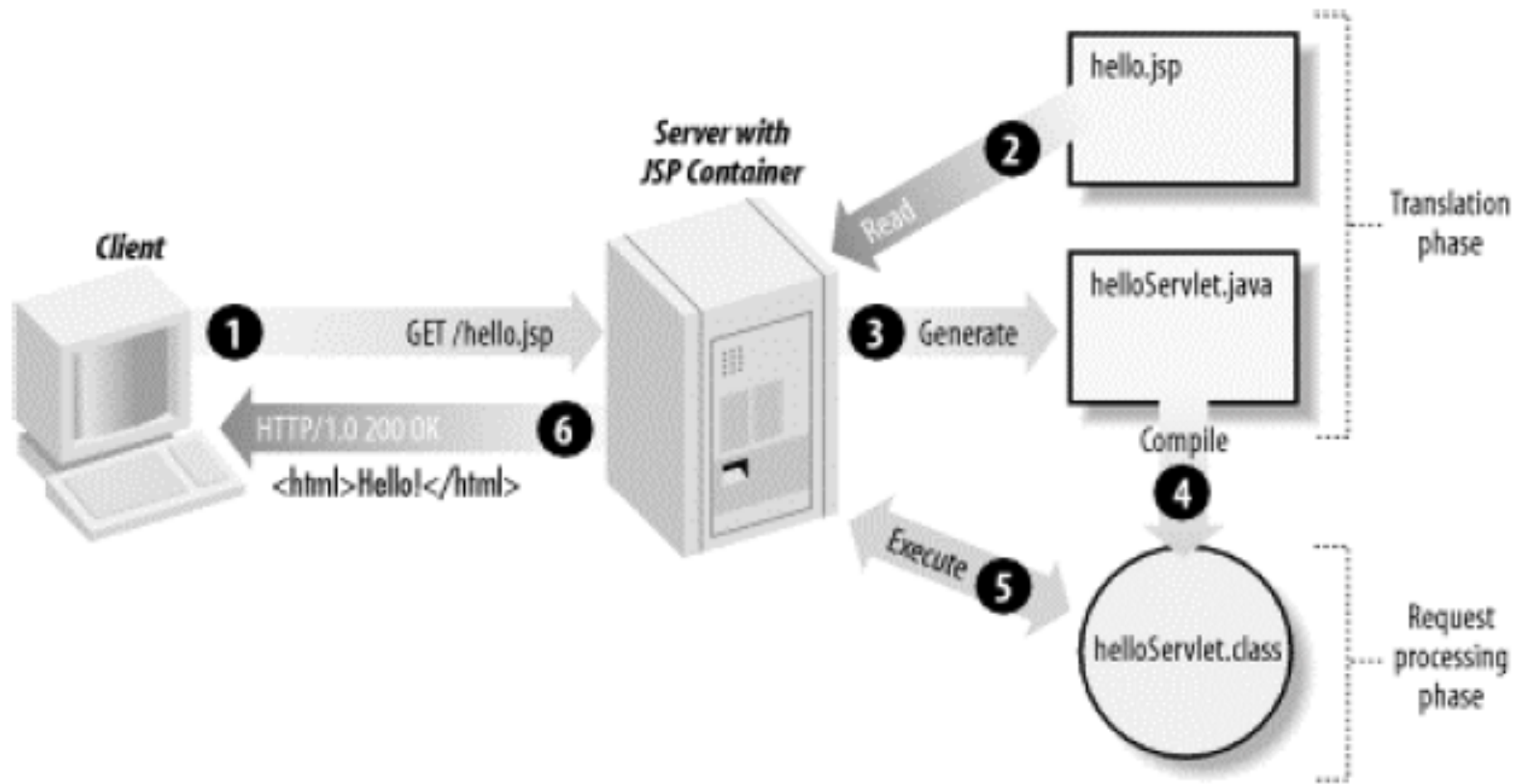
# ¿Beneficio?

- Incluir mucha lógica de programación en una página Web no es mucho mejor que generar el HTML por programa
  - Pero JSP proporciona acciones (***action elements***) que son como etiquetas HTML pero que representan código reutilizable
  - Además, se puede invocar a otras clases Java del servidor, a componentes (**Javabeans** o **EJB**)...

# Separación de presentación y lógica

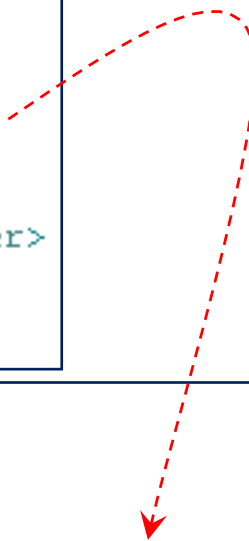
- En definitiva, lo que permite JSP (bien utilizado) es una **mayor separación entre la presentación de la página y la lógica de la aplicación**, que iría aparte
  - Desde la página JSP únicamente invocaríamos, de diferentes formas, a ese código

# JSP: Proceso de compilación



# Ejemplo: de JSP a Servlet (Tomcat)

```
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
  <center>Bienvenido a mi primera página Web!</center>
</BODY>
</HTML>
```



```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    public void _jspInit() {

    }

    public void _jspDestroy() {
    }

    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
```



# JSP a Servlet.service()

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

    try {
        response.setContentType("text/html");
        pageContext = _jspxFactory.getPageContext(this, request, response,
            null, true, 8192, true);
        _jspx_page_context = pageContext;
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();
        _jspx_out = out;

        out.write("<HTML>\r\n");
        out.write("<HEAD>\r\n");
        out.write("<TITLE>Hola Mundo!</TITLE>\r\n");
        out.write("</HEAD>\r\n");
        out.write("<BODY>\r\n");
        out.write("\t<center>Bienvenido a mi primera página Web!</center>\r\n");
        out.write("</BODY>\r\n");
        out.write("</HTML>\r\n");
        out.write("\r\n");
    } catch (Throwable t) {
        if (!(t instanceof SkipPageException)){
            out = _jspx_out;
            if (out != null && out.getBufferSize() != 0)
                try { out.clearBuffer(); } catch (java.io.IOException e) {}
            if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
        }
    } finally {
        _jspxFactory.releasePageContext(_jspx_page_context);
    }
}
```

# Elementos JSP

- Tres tipos de elementos en JSP:
  - Scripting
    - Permiten insertar código java que será ejecutado **en el momento de la petición**
  - Directivas
    - Permiten especificar información acerca de la página que permanece constante para todas las peticiones
      - Requisitos de buffering
      - Página de error para redirección, etc.
  - Acciones
    - Permiten ejecutar determinadas acciones sobre información que se requiere en el momento de la petición de la JSP
      - Acciones estándar
      - Acciones propietarias (Tag libs)

# Elementos de “scripting”

Elemento	Descripción
<code>&lt;% ... %&gt;</code>	<b>Scriptlet.</b> Encierra código Java
<code>&lt;%= ... %&gt;</code>	<b>Expresión.</b> Permite acceder al valor devuelto por una expresión en Java e imprimirlo en OUT
<code>&lt;%! ... %&gt;</code>	<b>Declaración.</b> Usada para declarar variables y métodos en la clase correspondiente a la página
<code>&lt;%-- ... --%&gt;</code>	<b>Comentario.</b> Comentario ignorado cuando se traduce la página JSP en un servlet. (comentario en el HTML <code>&lt;!-- comment →</code> )

# Un ejemplo más “dinámico”

- Hagamos una página JSP que, en función de la hora, muestre un saludo diferente (buenos días, buenas tardes o buenas noches)
  - Y que muestre también la hora actual

# Ejemplo: expresión, saludo.jsp

```
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
  <center>Bienvenido a mi primera página Web!</center>
  <p>Son las <%= new java.util.Date() %></p>
</BODY>
</HTML>
```

<% =

expresión

Nota: Si se necesita usar los caracteres "<%" dentro de un scriptlet, hay que usar "<%\>" y "<%\<%"

# Ejemplo: scriptlet, saludo.jsp

```
<%  
    String saludo;  
    java.util.Date hora = new java.util.Date();  
  
    if (hora.getHours() < 13){  
        saludo = "Buenos días";  
    }  
    else if (hora.getHours() < 20){  
        saludo = "Buenas tardes";  
    }  
    else {  
        saludo = "Buenas noches";  
    }  
%>  
<HTML>  
<HEAD>  
<TITLE>Hola Mundo!</TITLE>  
</HEAD>  
<BODY>  
    <center>Bienvenido a mi primera página Web!</center>  
    <p>Son las <%= hora %>, <%= saludo %></p>  
</BODY>  
</HTML>
```

<%

# Ejemplo: declaración, saludo.jsp

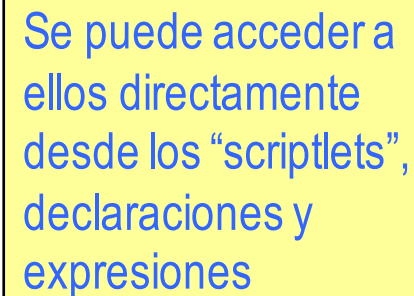
```
<%!  
private java.util.Date hora = new java.util.Date();  
  
java.util.Date getHora(){  
    return hora;  
}  
  
String getSaludo() {  
    String saludo;  
    if (hora.getHours() < 13) {  
        saludo = "Buenos días";  
    } else if (hora.getHours() < 20) {  
        saludo = "Buenas tardes";  
    } else {  
        saludo = "Buenas noches";  
    }  
    return saludo;  
}  
%>
```

<%!

```
<%!  
private java.util.Date hora = new java.util.Date();  
  
java.util.Date getHora(){  
    return hora;  
}  
  
String getSaludo() {  
    String saludo;  
    if (hora.getHours() < 13) {  
        saludo = "Buenos días";  
    } else if (hora.getHours() < 20) {  
        saludo = "Buenas tardes";  
    } else {  
        saludo = "Buenas noches";  
    }  
    return saludo;  
}  
%>  
<HTML>  
<HEAD>  
<TITLE>Hola Mundo!</TITLE>  
</HEAD>  
<BODY>  
<center>Bienvenido a mi primera página Web!</center>  
<p>Son las <%=getHora() %>, <%=getSaludo() %></p>  
</BODY>  
</HTML>
```

# JSP: objetos predefinidos

- El código java incrustado en JSP tiene acceso a los mismos objetos predefinidos que tenían los servlets
- Aquí se denominan:
  - request
  - response
  - pageContext
  - session
  - application
  - out
  - config
  - page



Se puede acceder a ellos directamente desde los “scriptlets”, declaraciones y expresiones



# JSP: Objetos predefinidos

- **request:**
  - `HttpServletRequest` asociado con la petición
  - Permite acceder a:
    - Los parámetros de la petición (mediante `getParameter`)
    - El tipo de petición (GET, POST, HEAD, etc.)
    - Las cabeceras HTTP entrantes (cookies, etc.)
  - Estrictamente hablando, se permite que la petición sea una subclase de `ServletRequest` distinta de `HttpServletRequest`, si el protocolo de la petición es distinto del HTTP
    - Esto casi nunca se lleva a la práctica

# JSP: Objetos predefinidos

- **response:**

- Objeto de clase `HttpServletResponse` asociado con la respuesta al cliente
- Como el stream de salida tiene un buffer, **es** legal seleccionar los códigos de estado y cabeceras de respuesta, aunque no está permitido en los servlets normales una vez que la salida ha sido enviada al cliente

# JSP: Objetos predefinidos

- **out:**
  - PrintWriter usado para enviar la salida al cliente
  - Esta es una versión con buffer de PrintWriter llamada JspWriter
    - Podemos ajustar el tamaño del buffer, o incluso desactivarlo, usando el atributo buffer de la directiva page
  - Se usa casi exclusivamente en scriptlets ya que las expresiones JSP obtienen un lugar en el stream de salida, y por eso raramente se refieren explícitamente a *out*

# JSP: Objetos predefinidos

- **session:**
  - HttpSession asociado con la petición
  - Las sesiones se crean automáticamente, por esto esta variable se une incluso si no hubiera una sesión de referencia entrante
  - La única excepción es usar el atributo *session* de la directiva *page* para desactivar las sesiones, en cuyo caso los intentos de referenciar la variable *session* causarán un error en el momento de traducir la página JSP a un servlet

# JSP: Objetos predefinidos

- **application:**

- El `ServletContext` obtenido mediante `getServletConfig().getContext()`

- **config:**

- El objeto `ServletConfig`

- **pageContext:**

- `PageContext` es un nuevo contexto, además de `Session` y `ServletContext`
- Permite acceder a writer específico para JSP y establecer atributos en ámbito de página

# Índice

- Introducción
- Elementos de JSP
  - Elementos de secuencias/Scripting
  - Directivas
  - Acciones

# Directivas

- Las directivas son mensajes para el contenedor de JSP
- Ejemplos:

Elemento	Descripción
<code>&lt;%@ page ... %&gt;</code>	Permite importar clases Java, especificar el tipo de la respuesta ("text/html" por omisión), etc.
<code>&lt;%@ include ... %&gt;</code>	Permite incluir otros ficheros antes de que la página sea traducida a un servlet
<code>&lt;%@ taglib ... %&gt;</code>	Declara una biblioteca de etiquetas con <b>acciones</b> personalizadas para ser utilizadas en la página

Forma general: `<%@ directive { attr="value" }* %>`

# JSP: Directiva “page”

- **import**="package.class" o  
import="packg.class1,...,packg.classN".
  - Permite especificar los paquetes que deberían ser importados. El atributo import es el único que puede aparecer múltiples veces
- **ContentType** = "MIME-Type" o contentType =  
"MIME-Type; charset = Character-Set"
  - Especifica el tipo MIME de la salida. El valor por defecto es “text/html”. Tiene el mismo valor que el scriptlet usando “response.setContentType”
- **isThreadSafe** = "true|false".
  - True (por defecto) indica procesamiento del servlet normal, múltiples peticiones pueden procesarse simultáneamente con un sólo ejemplar del servlet (el autor sincroniza los recursos compartidos). Un valor de false indica que el servlet debería implementar `SingleThreadModel`.



# JSP: Directiva “page”

- **session** = "true|false"
  - True (por defecto) indica que la variable predefinida session (del tipo HttpSession) debería unirse a la sesión existente si existe una; si no existe se debería crear una nueva sesión para unirla. False indica que no se usarán sesiones, y los intentos de acceder a la variable session resultarán en errores en el momento en que la página JSP sea traducida a un servlet
- **buffer** = "sizekb|none"
  - Esto especifica el tamaño del buffer para el JspWriter out. El valor por defecto es específico del servidor y debería ser de al menos 8kb.
- **autoflush, extends, info, errorPage, isErrorPage, language**

# Ejemplo directiva “page”

- Modificar la página de saludo para que en lugar de hacer referencia directamente a java.util.Date importe el paquete java.util

```
<%@ page import="java.util.Date"
    contentType="text/html"
    pageEncoding="iso-8859-1" %>
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
<center>Bienvenido a mi primera página Web!</center>
<p>Son las <%= new Date() %></p>
</BODY>
</HTML>
```

Juego de  
Caracteres  
UNICODE latin-  
1

# JSP: Directiva “include”

- Permite incluir ficheros en el momento en que la página JSP es traducida a un servlet

```
<%@ include file="url relativa" %>
```

- Los contenidos del fichero incluido son analizados como texto normal JSP y así pueden incluir HTML estático, elementos de script, directivas y acciones
- Uso
  - Barras de navegación, copyright, etc.


```
<%@ include file="copyright.html" %>
```

# Ejemplo: directiva include

```
<%@ page import="java.util.Date"
      contentType="text/html"
      pageEncoding="iso-8859-1" %>

<p>Son las <%= new Date() %></p>
```

```
<%@ page contentType="text/html"
      pageEncoding="iso-8859-1" %>
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
<center>Bienvenido a mi primera página Web!</center>
<%@ include file="date.jsp" %>
</BODY>
</HTML>
```



# Directiva “taglib”

- Declara que la página usa una librería de tags (acciones)

```
<%@ taglib uri="http://www.mycorp/supertags"  
prefix="super" %>
```

...

```
<super:doMagic> ... </super:doMagic>
```

- Identifica la librería por su URI
- Le asocia un prefijo para usar en el código de la JSP

## Syntax

```
<%@ taglib ( uri="tagLibraryURI" | tagdir="tagDir" ) prefix="tagPrefix" %>
```

# Directiva “taglib”

- **uri** = “taglib.tld”
  - Absoluto o relativo. Permite especificar el fichero de descripción de la librería (.tld)
- **tagdir** = “myTags”
  - Especifica el directorio bajo /WEB-INF/tags/ en el cual hay ficheros de tags (.tag ó tagx) que no están empaquetados en bibliotecas de tags
- **prefix**= “c” (“fmt”, etc.)
  - Prefijo que permite distinguir acciones cuando se importan varias bibliotecas

# JSP: Acciones

- Usan construcciones de sintaxis XML para controlar el comportamiento del motor de servlets. Podemos insertar un fichero dinámicamente, reutilizar componentes JavaBeans, reenviar al usuario a otra página, etc.
- Tipos de acciones
  - Estándar
  - A medida
  - JSTL

Nota: Los elementos XML, al contrario que los HTML, **son sensibles a las mayúsculas**

# JSP: Acciones ejemplo

```
<HTML>
<HEAD>
<TITLE>Validación de usuario</TITLE>
</HEAD>
<BODY>

..... Body contents .....

<sig:signature name="Enrique DLC"/>
</BODY>
</HTML>
```

JSP

HTML generado

Bienvenido a la aplicación

-----  
Enrique A. de la Cal Marin  
Profesor T.U.  
Universidad de Oviedo  
Edificio Departamental zona Oeste, Edificio 1  
Desp. 1.2.16  
Departamento de Informatica  
Tlfo. +34 985182520  
Fax. +34 985181986  
-----



# Acciones

- Elementos XML que realizan determinadas acciones
  - JSP define las siguientes (estándar):

Elemento	Descripción
<code>&lt;jsp:useBean&gt;</code>	Permite usar un <i>JavaBean</i> desde la página
<code>&lt;jsp:getProperty&gt;</code>	Obtiene el valor de una propiedad de un componente <i>JavaBean</i> y lo añade a la respuesta
<code>&lt;jsp:setProperty&gt;</code>	Establece el valor de una propiedad de un <i>JavaBean</i>
<code>&lt;jsp:include&gt;</code>	Incluye la respuesta de un servlet o una página JSP
<code>&lt;jsp:forward&gt;</code>	Redirige a otro servlet o página JSP
<code>&lt;jsp:param&gt;</code>	Envía un parámetro a la petición redirigida a otro servlet o JSP mediante <code>&lt;jsp:include&gt;</code> o <code>&lt;jsp:forward&gt;</code>
<code>&lt;jsp:plugin&gt;</code>	Genera el HTML necesario para ejecutar un <i>applet</i>

# JSP: Acción useBean

- Permite cargar y utilizar un JavaBean en la página JSP y así utilizar la reusabilidad de las clases Java

```
<jsp:useBean id="name" class="package.class" />
```

- Esto significa: "usa un objeto de la clase especificada por class, y únelo a una variable con el nombre especificado por id"
- Ahora podemos modificar sus propiedades mediante `<jsp:setProperty>`, o usando un scriptlet y llamando a un método del objeto referenciado por id
  - Para recoger una propiedad se usa `<jsp:getProperty>`

# JSP: Acción useBean

- **id**
  - Da un nombre a la variable que referenciará el bean. Se usará un objeto bean anterior en lugar de instanciar uno nuevo si se puede encontrar uno con el mismo id y ámbito (scope)
- **class**
  - Designa el nombre cualificado completo del bean
- **scope**
  - Indica el contexto en el que el bean debería estar disponible. Hay cuatro posibles valores: `page`, `request`, `session`, y `application`
- **type**
  - Especifica el tipo de la variable a la que se referirá el objeto
- **beanName**
  - Da el nombre del bean, como lo suministraríamos en el método `instantiate` de Beans. Está permitido suministrar un `type` y un `beanName`, y omitir el atributo `class`

# JSP: Acciones: setProperty

- Para asignar valores a propiedades de los beans que se han referenciado anteriormente
- 2 usos:

- Despues de un useBean

```
<jsp:useBean id="myName" ... />
```

```
...
```

```
<jsp:setProperty name="myName"  
                property="someProperty" ... />
```

Se ejecuta siempre que haya una solicitud

# JSP: Acciones: setProperty

## – Dentro de un useBean

```
<jsp:useBean id="myName" ... >
    ...
    <jsp:setProperty name="myName"
        property="someProperty" ... />
</jsp:useBean>
```

Solo se ejecuta cuando haya que instanciar un bean

# JSP: Acciones: setProperty

- **name:**
  - Este atributo requerido designa el bean cuya propiedad va a ser seleccionada. El elemento `jsp:useBean` debe aparecer antes del elemento `jsp:setProperty`.
- **property:**
  - Este atributo requerido indica la propiedad que queremos seleccionar
  - Hay un caso especial: un valor de "\*" significa que todos los parámetros de la petición cuyos nombres correspondan con nombres de propiedades del Bean serán pasados a los métodos de selección apropiados
- **value:**
  - Este atributo opcional especifica el valor para la propiedad. Los valores string son convertidos automáticamente a lo que corresponda mediante el método estándar `valueOf`. No se pueden usar `value` y `param` juntos, pero si está permitido no usar ninguna

# JSP: Acciones: setProperty

- **param**

- Este parámetro opcional designa el parámetro de la petición del que se debería derivar la propiedad
- Si la petición actual no tiene dicho parámetro, no se hace nada: el sistema no pasa null al método seleccionador de la propiedad. Así, podemos dejar que el bean suministre los valores por defecto, sobrescribiéndolos sólo cuando el parámetro dice que lo haga

```
<jsp:setProperty          name="orderBean"          property="numberOfItems"  
    param="numItems" />
```

- Si no indicamos nada, el servidor revisa todos los parametros de la petición e intenta encontrar alguno que concuerde con la propiedad indicada

# Ejemplo de uso <jsp:xxx

```
<%@ page contentType="text/html" pageEncoding="iso-8859-1"
    isELIgnored="false"%>

<jsp:useBean id="date" class="uo.dasdi.beans.DateBean" />

<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
<center>Bienvenido a mi primera página Web!</center>

<%
    if (date.getHour() > 12) {
        Son mas de las 12, son ya las las <%=date.getHour() %>
    }
%>

</BODY>
</HTML>
```

A red dashed line connects the `id="date"` attribute of the `<jsp:useBean>` tag to the `date` object used in the `if (date.getHour() > 12)` statement. Another red dashed line connects the `date.getHour()` expression inside the `if` statement to the `<%=date.getHour() %>` output expression in the HTML body.



# Ejemplo de uso <jsp:xxx código del bean

```
package uo.dasdi.beans;

import java.util.Calendar;

public class DateBean {
    private Calendar calendar = Calendar.getInstance();

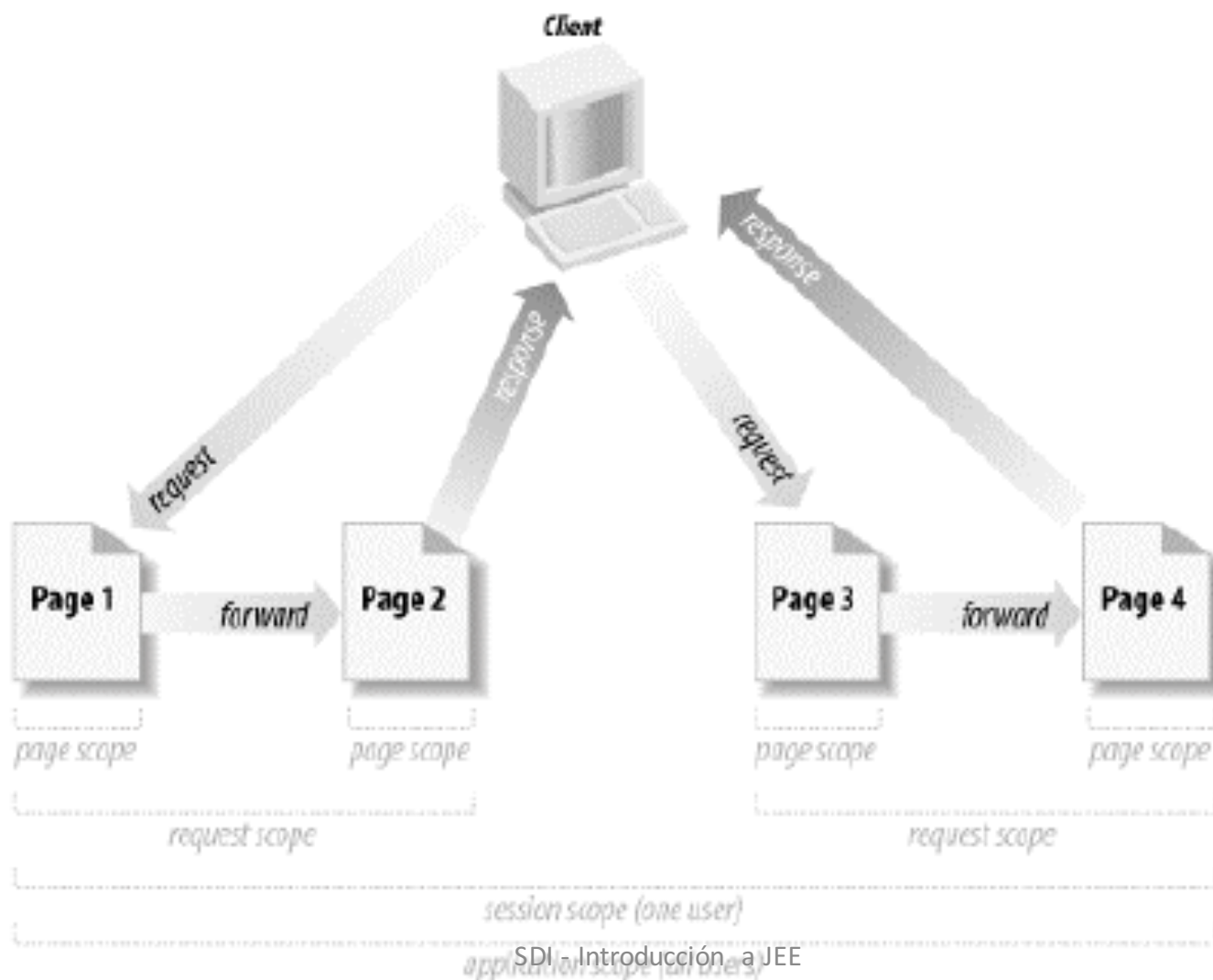
    public Date getDate(){
        return calendar.getTime();
    }

    public int getHour(){
        return calendar.get(Calendar.HOUR_OF_DAY);
    }

    public String getGreeting(){
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        String greeting;
        if (hour < 13) {
            greeting = "Buenos días";
        } else if (hour < 20) {
            greeting = "Buenas tardes";
        } else {
            greeting = "Buenas noches";
        }
        return greeting;
    }
}
```

# Comunicación entre jsp

## Posibles ámbitos



# Acciones JSTL, ejemplo

```
<%@ page contentType="text/html" pageEncoding="iso-8859-1" isELIgnored="false"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<jsp:useBean id="date" class="uo.dasdi.beans.DateBean" />

<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
<center>Bienvenido a mi primera página Web!</center>
<c:if test="${date.hour > 12}">
    <c:out value="Son mas de las 12, son las ${date.hour}" />
</c:if>
</BODY>
</HTML>
```

Con tag-libs no hay scriptlet

# Encadenamiento de Servlets y JSPs

# Encadenamiento de Servlets/JSPs

- Desde un Servlet concatenar otro Servlet:
  - `RequestDispatcher/forward/include`
- Desde un JSP
  - `<jsp:forward>`
  - `<jsp:include>`

# Cómo reenviar peticiones

- Para reenviar peticiones o incluir un contenido externo se emplea:

```
RequestDispatcher  
    getServletContext().getRequestDispatcher(URL)
```

- Ejemplo:

```
String url = “/presentaciones/presentacion1.jsp”;  
RequestDispatcher  
    despachador=getServletContext().getRequestDispatcher(url);  
//Para pasar el control total usar forward  
despachador.forward(request, response);  
//forward genera las excepciones ServletException y IOException
```

# Ejemplo de reenvío de peticiones

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String operation = request.getParameter("operation");
    if (operation == null) {
        operation = "unknown"; }

    if (operation.equals("operation1")) {
        gotoPage("/operations/presentation1.jsp", request, response); }
    else if (operation.equals("operation2")) {
        gotoPage("/operations/presentation2.jsp", request, response); }
    else { gotoPage("/operations/unknownRequestHandler.jsp", request, response); }
}

private void gotoPage(String address, HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
```

# Paso de información procesada a Servlet/JSP

- Criterio: El servlet procesará los datos y se los pasará a la/s página/s JSP cuando:
  - Los datos son complejos de procesar (el servlet es más adecuado para ello).
  - Son varias las páginas JSP las que pueden recibir los mismos datos (el servlet los procesa de forma más eficiente).
- Formas de pasar datos a un JSP desde un Servlet:
  - Almacenándolos en `HttpServletRequest`.
  - En el URL de la página objetivo.
  - Pasándolos con un Bean.
- Paso de datos en **HttpServletRequest**
  - En el servlet: `request.setAttribute("clave1", valor1);`
  - En JSP: `request.getAttribute("clave1");`
- Pasar datos en el **URL**
  - En el servlet:  
`address = "/ruta/recurso.jsp?clave1=valor1"`  
`RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);`  
`dispatcher.forward(request, response);`
  - En JSP: el nuevo parámetro se agrega al principio de los datos consultados.  
`request.getParameter("clave1");`



# Cómo interpretar los URLs relativos en la página objetivo

- Un servlet puede reenviar peticiones a lugares arbitrarios en mismo servidor mediante `forward()`. Diferencias con `sendRedirect()`:
  - `sendRedirect ()`
    - Necesita que el cliente se vuelva a conectar al nuevo recurso
    - No conserva todos los datos de la conexión
    - Trae consigo un URL final distinto
  - `forward()`
    - Se maneja internamente en servidor
    - Conserva los datos de la conexión
    - Conserva el URL relativo del servlet
- Si la página objetivo trae URLs relativos a la raíz del servidor y no a su propio directorio como el ejemplo siguiente:

`<LINK REL=STYLESHEET HREF="mis_estilos.css" TYPE=text/css>`

- En el caso de una redirección a esta página, se considerará `mis_estilos.css` relativo a la dirección del servlet que genera la etiqueta `LINK` y no a la página JSP generando un error. **Solución:**

`<LINK REL=STYLESHEET HREF="/ruta/mis_estilos.css" TYPE=text/css>`

- Lo mismo para : `<IMG SRC=...`, `<A HREF=...`, ...

# Cómo incluir contenido estático o dinámico

- Si un servlet usa el método `RequestDispatcher.forward()` no podrá enviar ningún resultado al cliente, tendrá que dejarlo todo a la página objetivo.
- Para mezclar resultados del propio servlet con la página JSP o HTML se deberá emplear **`RequestDispatcher.include()`**:

```
out.println("...");
requestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);
dispatcher.include(request, response);
out.println("...");
```

- Las diferencias con una redirección consiste en:
  - Se pueden enviar datos al navegador antes de hacer la llamada. (`out.println("...");`)
  - El control se devuelve al servlet cuando finaliza `include`.
  - Las páginas JSP, servlets, HTML incluidas por el servlet no deben establecer encabezados HTTP de respuesta.
- `include()` se comporta igual que `forward` propagando toda la información de la petición original (GET o POST, parámetros de formulario, ...) y además establece
  - **`javax.servlet.include.request_uri`, `context_path`, `servlet_path`, `path_info` y `query_string`**

# Ejemplo de inclusión de datos desde el servlet principal

```
public class ShowPage extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html"); PrintWriter out = response.getWriter();
        String url = request.getParameter("url");
        out.println(ServletUtilities.headWithTitle(url) + "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + url + "</H1>\n" + "<FORM><CENTER>\n" +
            "<TEXTAREA ROWS=30 COLS=70>");
        if ((url == null) || (url.length() == 0)) { out.println("No URL specified."); }
        else {
            String data = request.getParameter("data");
            if ((data != null) && (data.length() > 0)) {
                url = url + "?" + data;
            }
            RequestDispatcher dispatcher=getServletContext().getRequestDispatcher(url);
            dispatcher.include(request, response);
        }
        out.println("</TEXTAREA>\n" + "</CENTER></FORM>\n" + "</BODY></HTML>");
    }
};
```

# Bibliografía

- **Un Best sheller en JEE – Marty Hall**
  - <http://www.coreservlets.com>
- **JEE 6**
  - <https://docs.oracle.com/javaee/7/api/>
- **Java 7 API**
  - <http://docs.oracle.com/javase/7/docs/api/>