Escuela Politécnica de Ingeniería
Grado de Ingeniería Informática en Tecnologías de la Información

Tecnologías Web

Tema 3

Tecnologías web de servidor: JEE (Servlets)

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

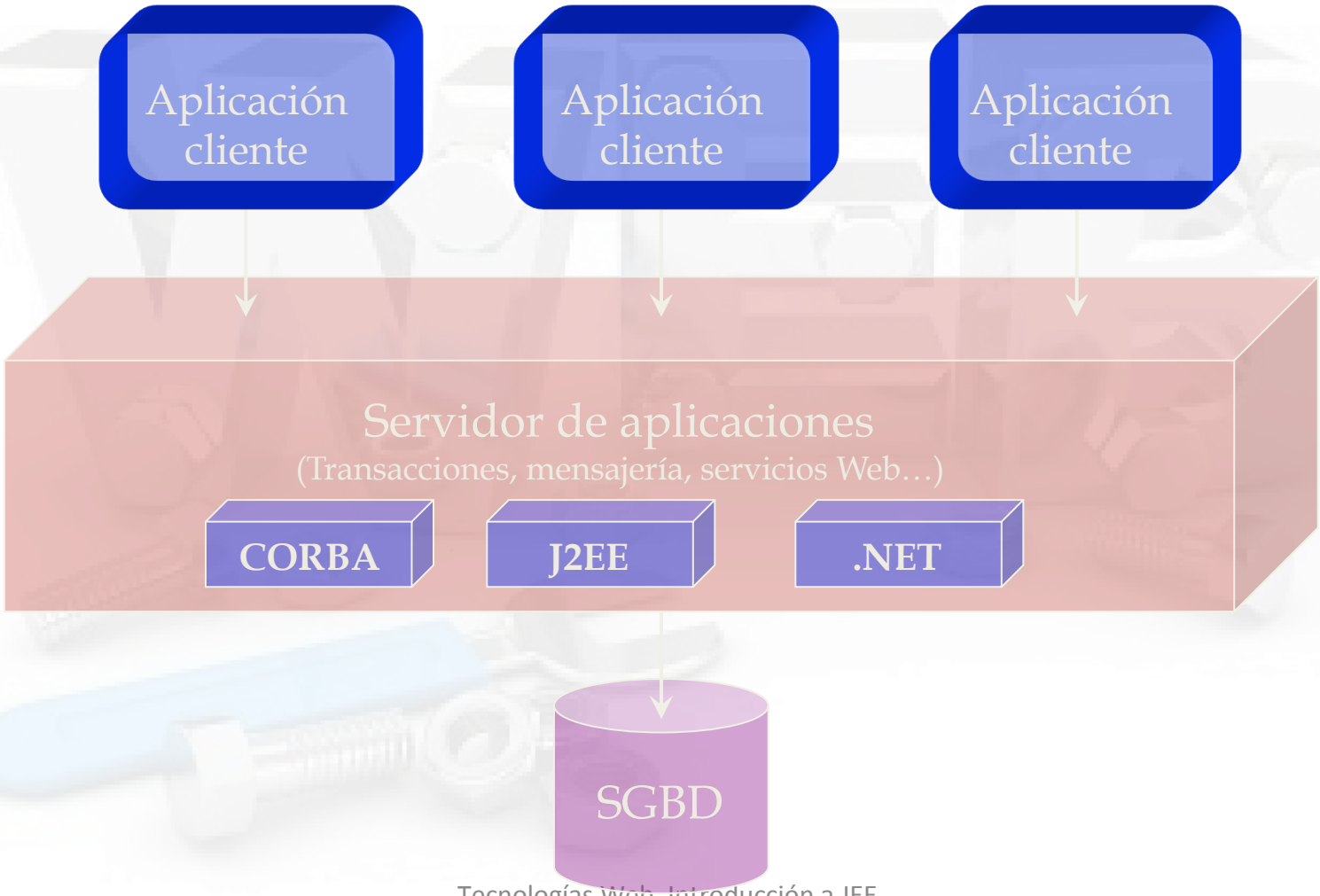
- Java EE = Java Enterprise Edition

- 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

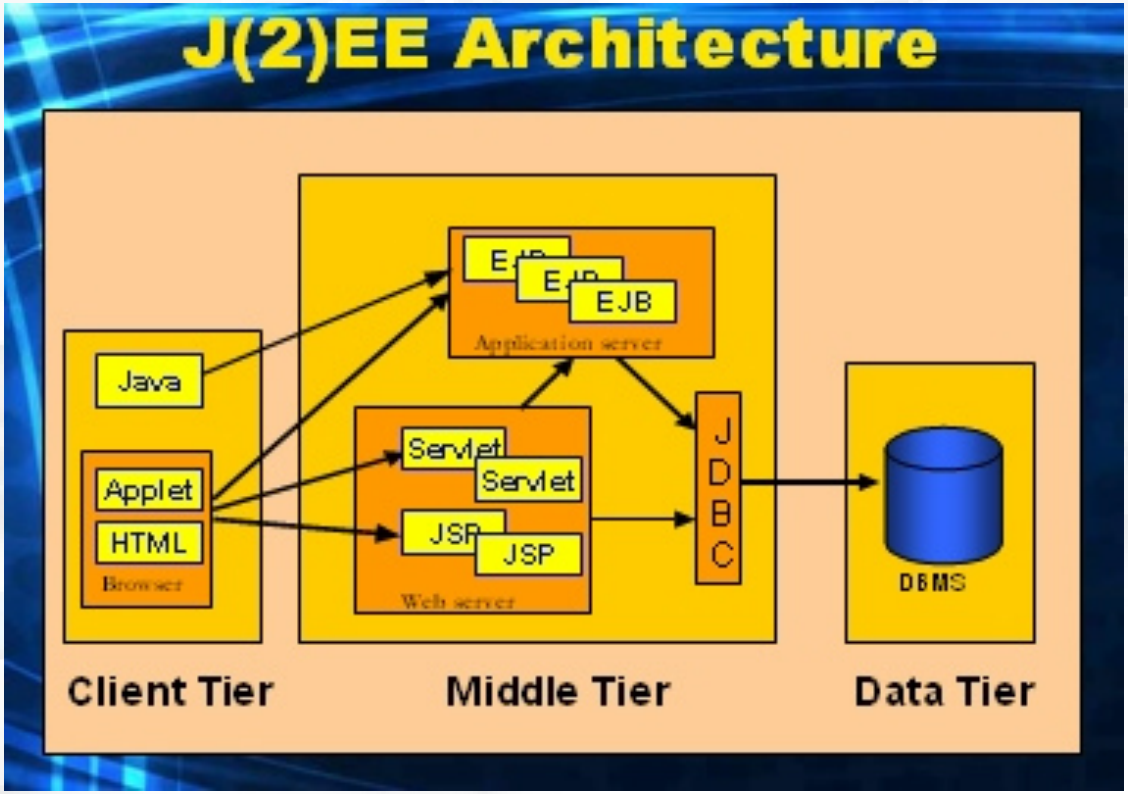


Servidores de aplicaciones

- 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

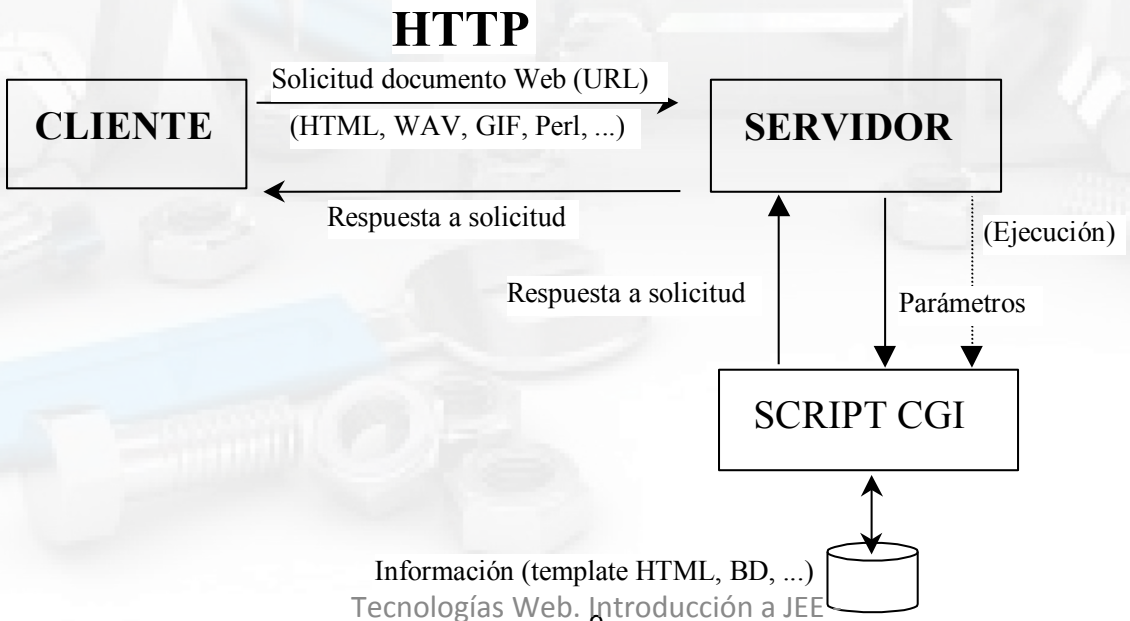
Servidores de aplicaciones

- 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

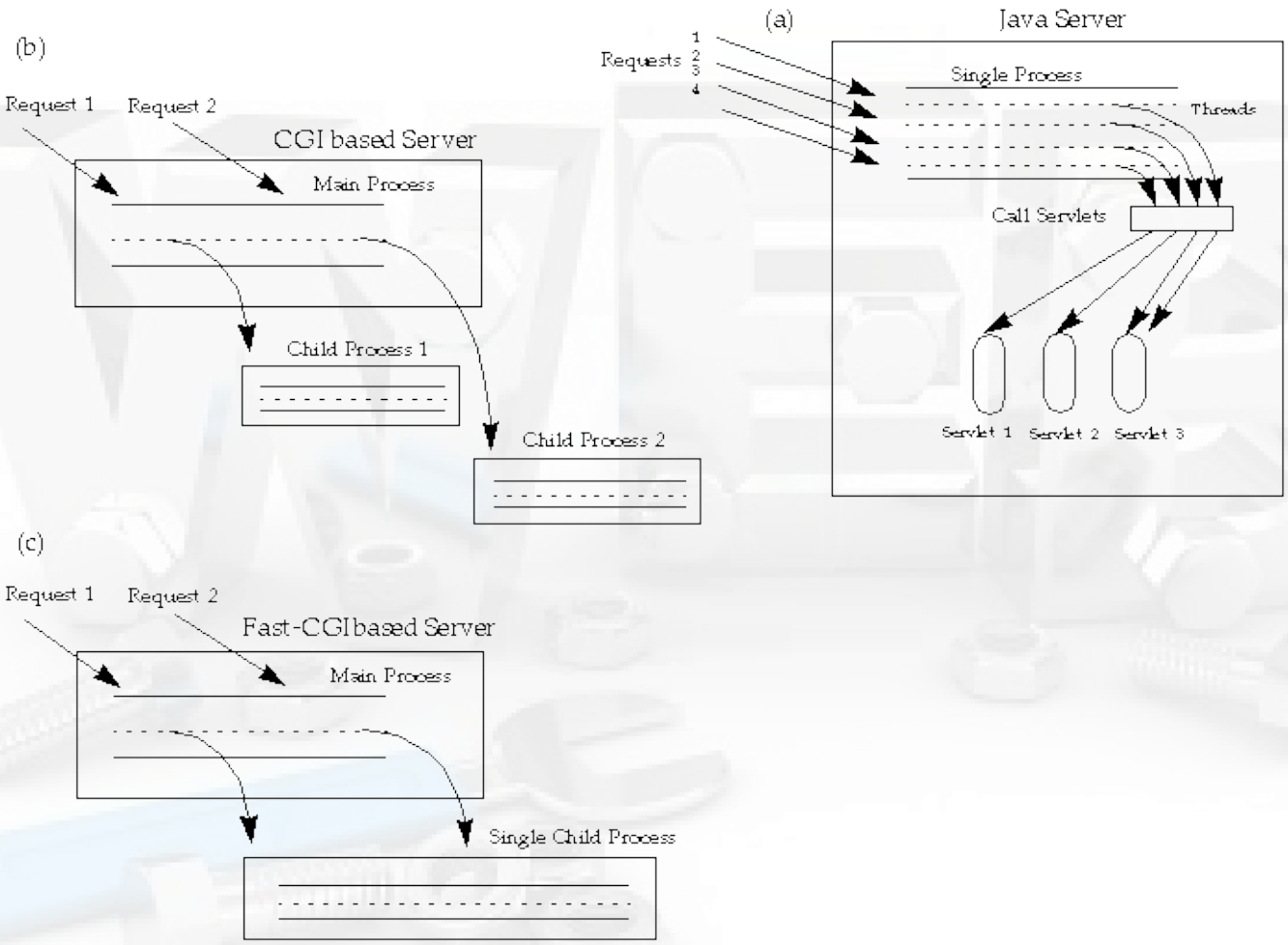
```
#!/usr/bin/perl
print "Content-Type: text/html\n\n";
print "<H1>Este es un script CGI muy sencillo</H1>\n";
print "¡Este es mi primer script CGI!\n";
```

- ***¿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 CGIs

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

Contenedor de servlets

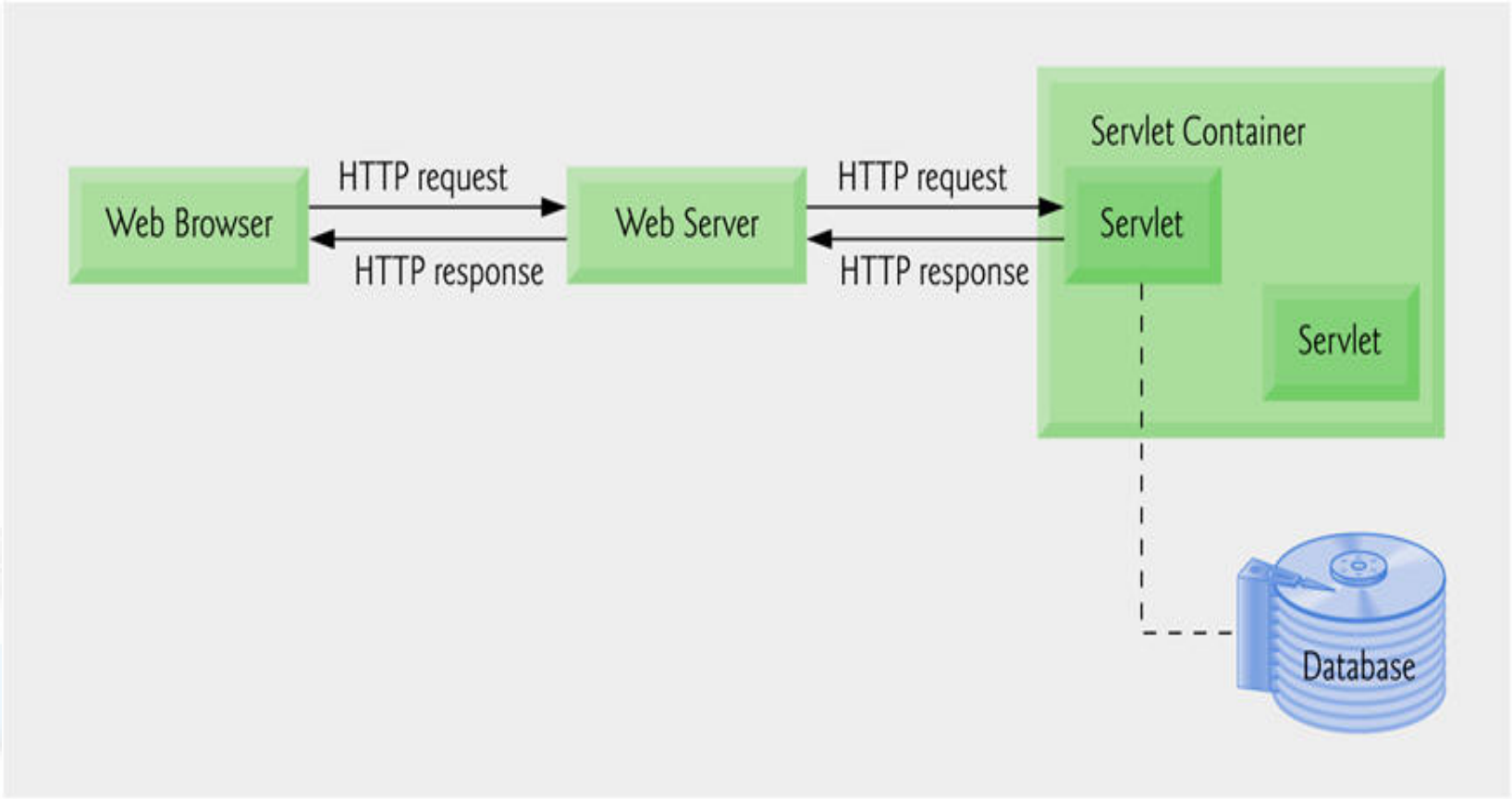
- Un contenedor define un ambiente estandarizado de ejecución que provee servicios específicos a los servlets
  - Por ejemplo, dan servicio a las peticiones de los clientes, realizando un procesamiento y devolviendo el resultado
- Los servlets tienen que cumplir un contrato con el contenedor para obtener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

Servlet "HolaMundo.java"

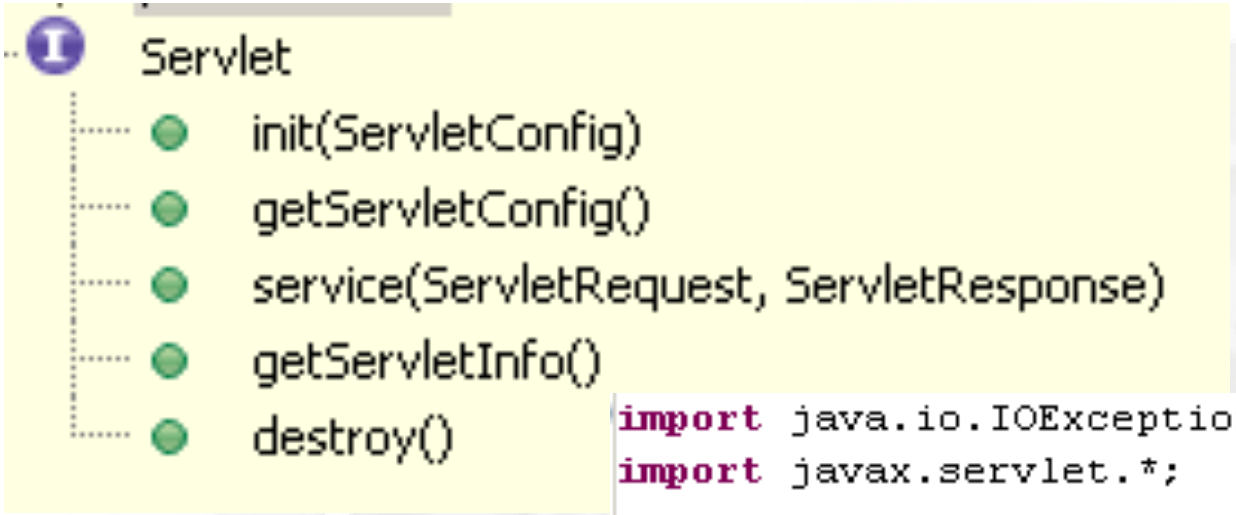
```
import java.io.*;          // Necesario importar estos tres paquetes
import javax.servlet.*;
import javax.servlet.http.*;

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

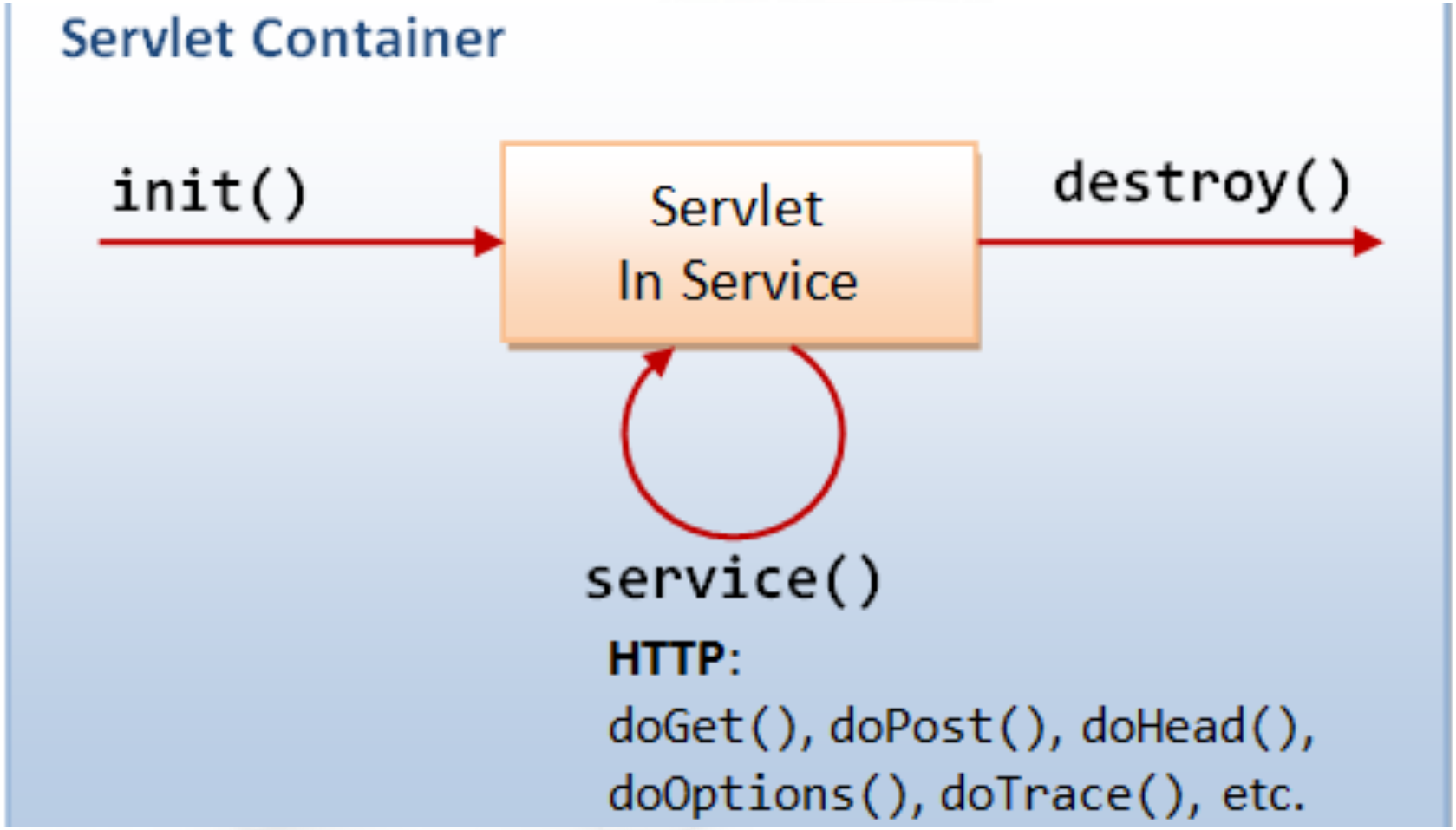


Servlet API

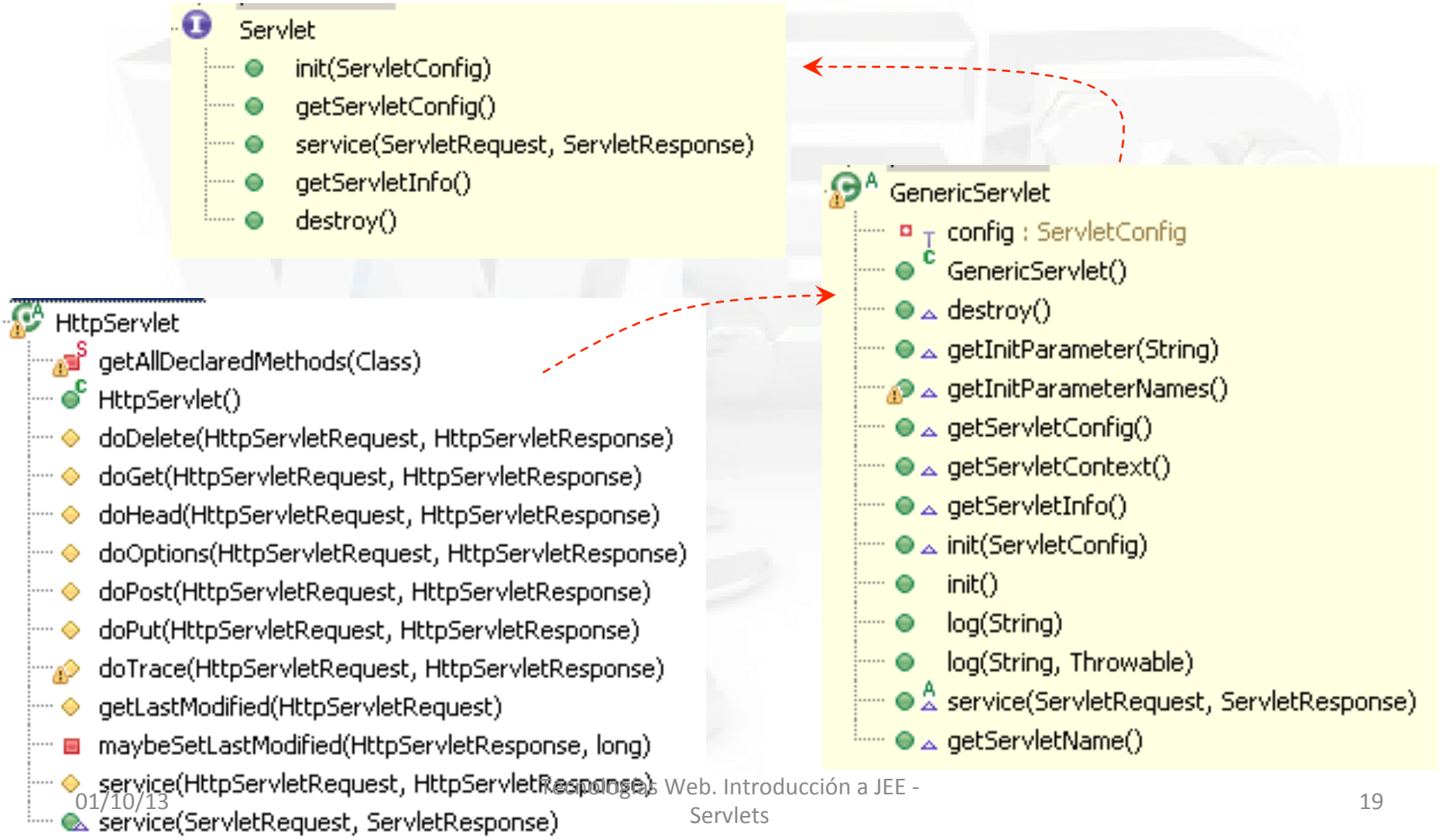



```
import java.io.IOException;
import javax.servlet.*;

public class MyServlet extends GenericServlet {

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

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

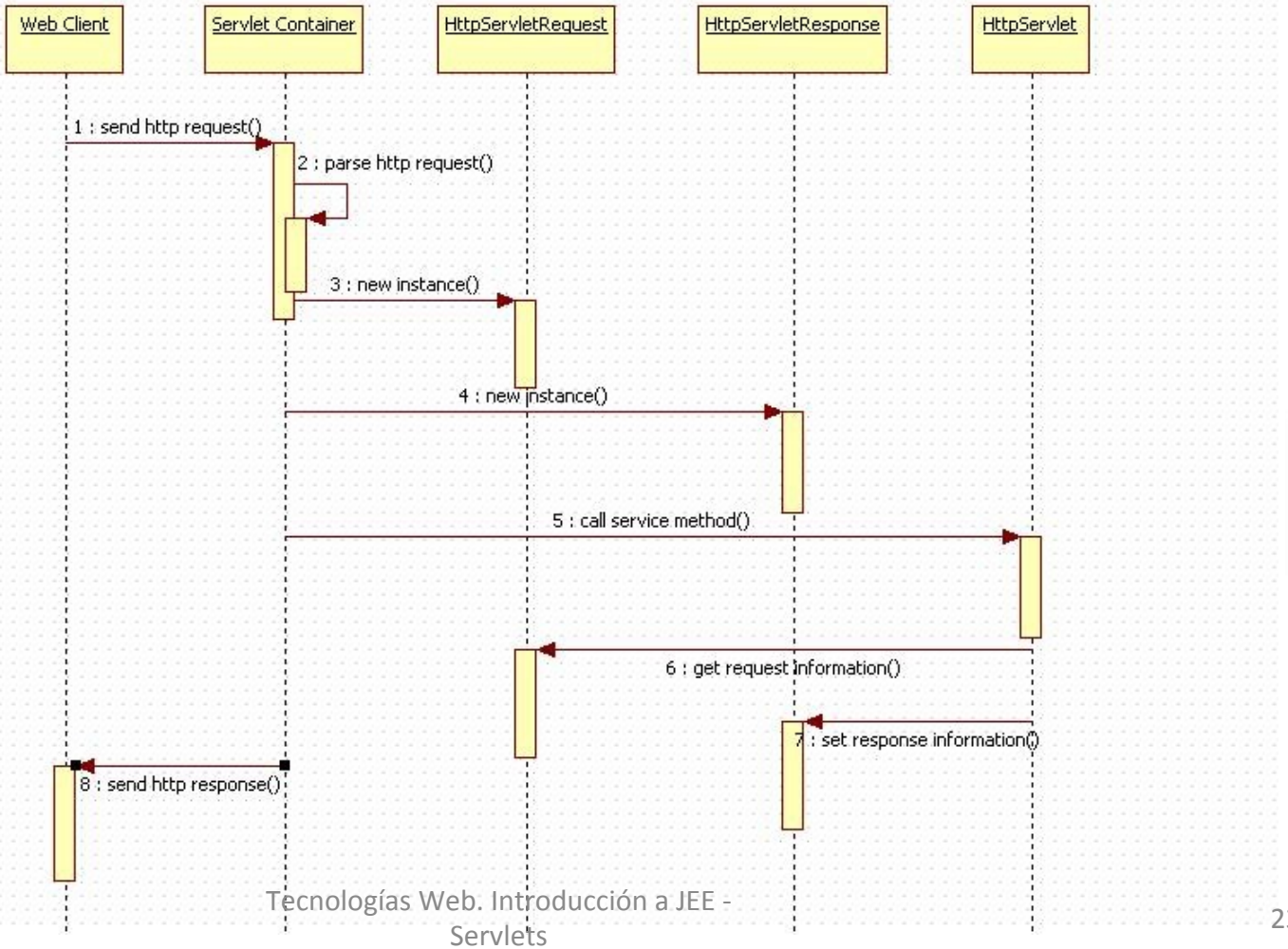
Servlets: Métodos doGet y doPost

- 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)

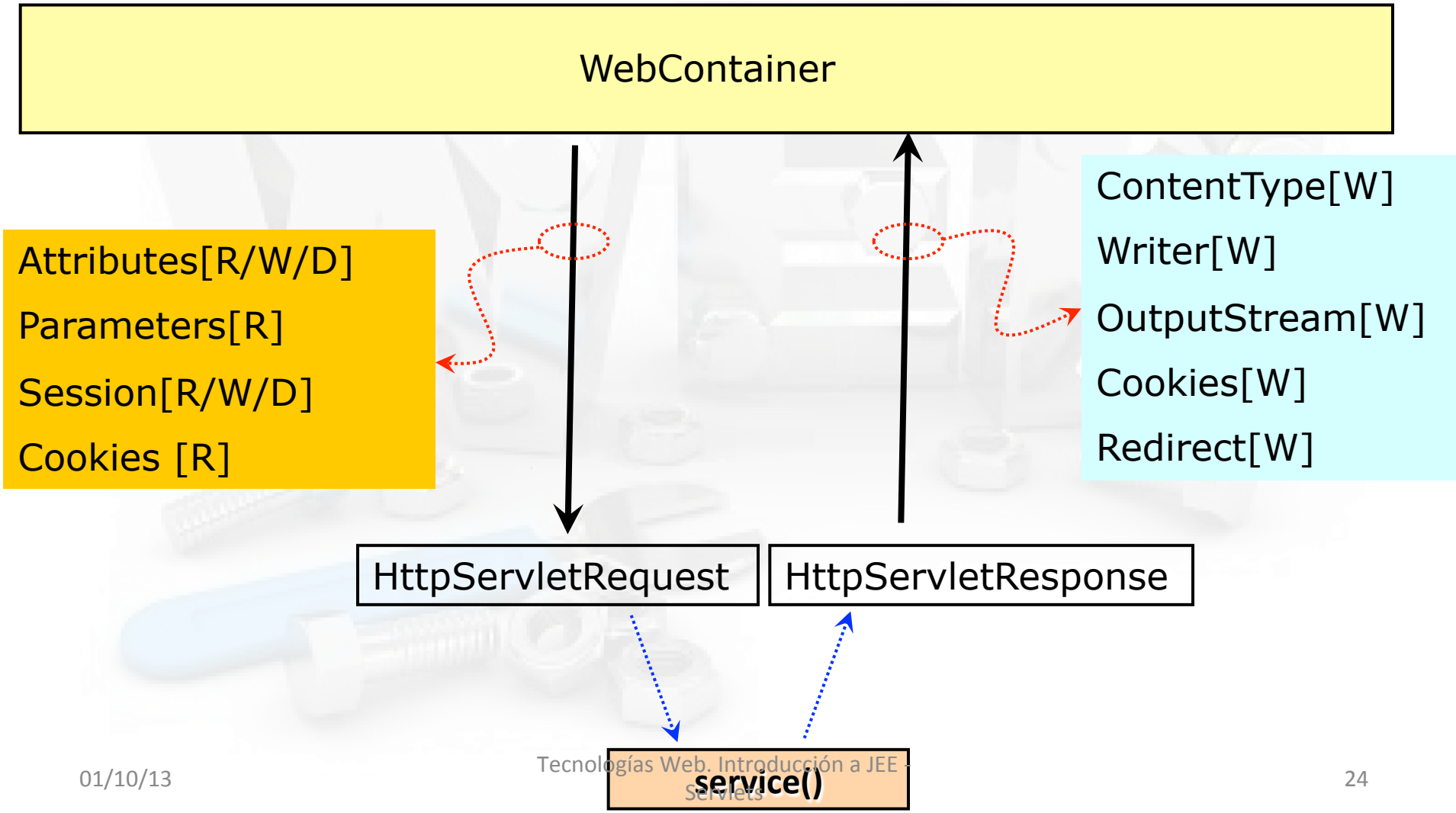
```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, java.io.IOException {
    . . .
}

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

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 HMTL
      - 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

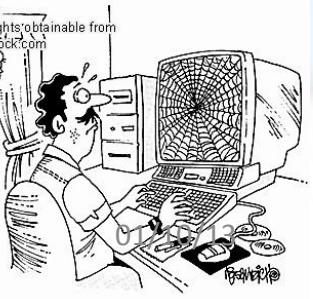
HolaMundo Servlet

```
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

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&oacute;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)

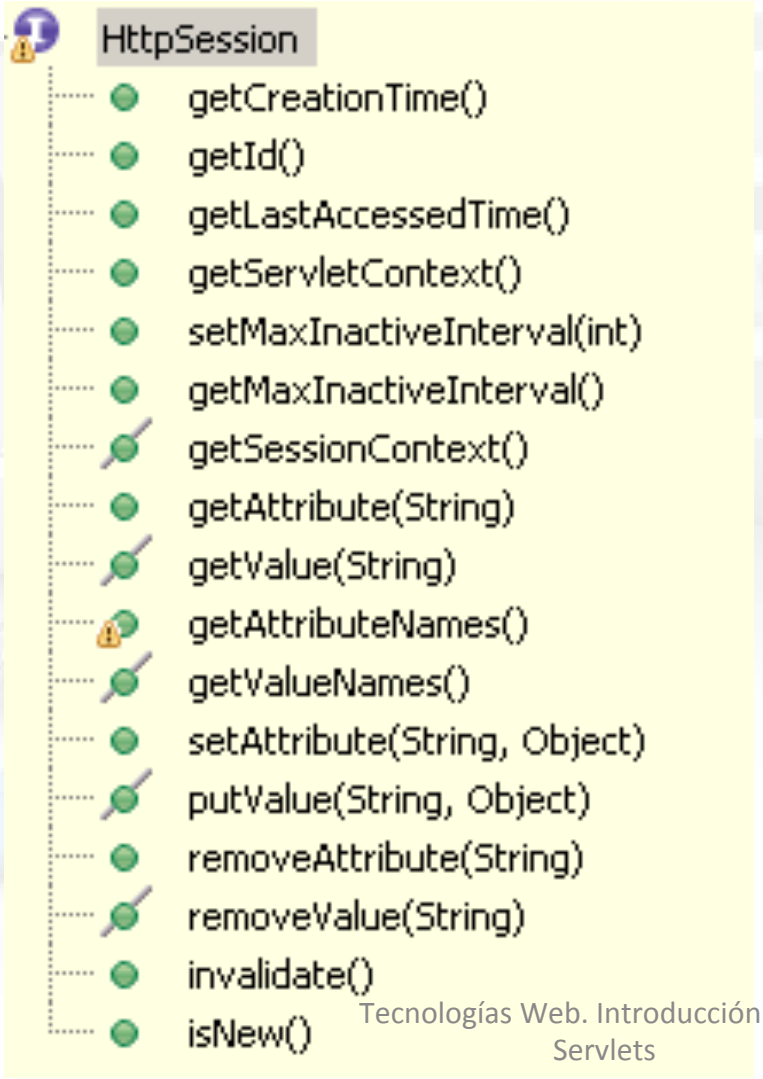
Servlets: Seguimiento de sesión

- 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



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**
  - **getAttributesNames()**
    - 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 creo 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

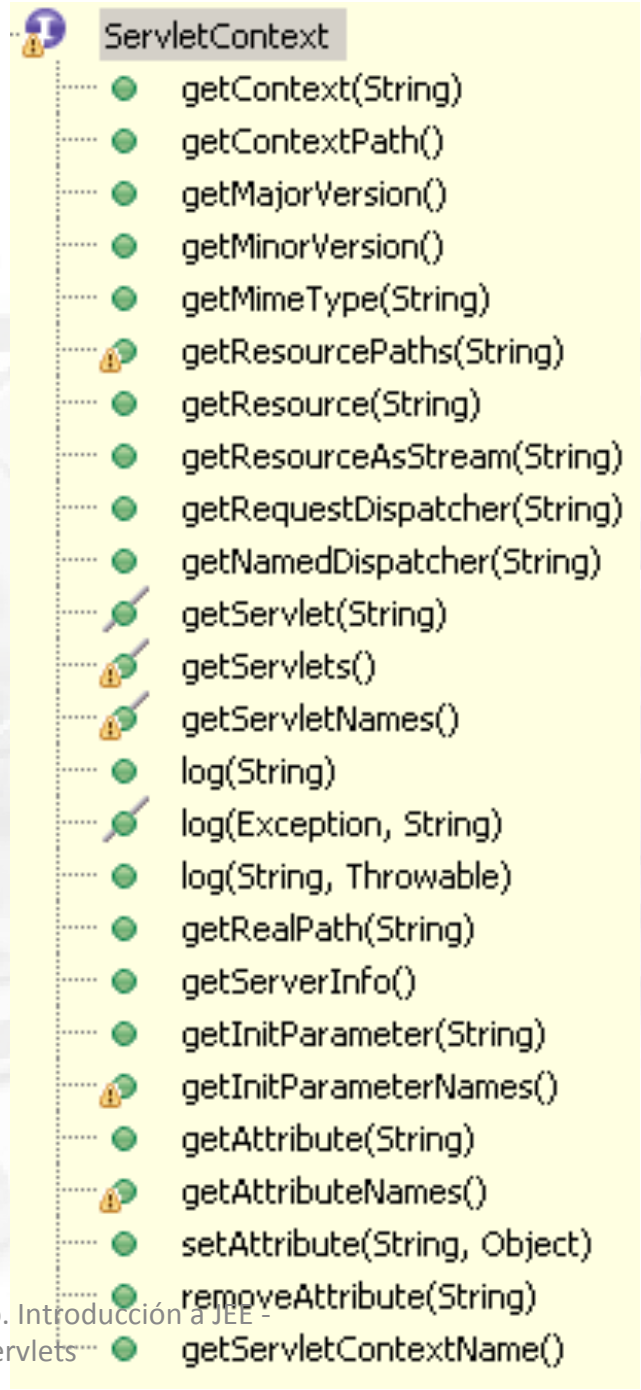
- Se trata de un saco "**común**" a todas las sesiones de usuario activas en el servidor
- Nos permite compartir información y objetos entre los distintos usuarios
- Se accede por medio del objeto "`ServletContext`"

```
public ServletContext getServletContext()
```

(Ejemplo: Contador de Visitas)

Se hereda de GenericServlet

Interfaz ServletContext()



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