Bases de Programación Java - ISII

Aplicaciones Web Dinámicas

Programación Web del lado del servidor

VNIVERSITAT DE VALÈNCIA
Escola Tècnica Superior d'Enginyeria

1. Servlets
    1. Introducción a Servlets
    2. Objeto Request
    3. Objeto Response
    4. Cookies
    5. Sesiones
    6. Reescritura de URLs

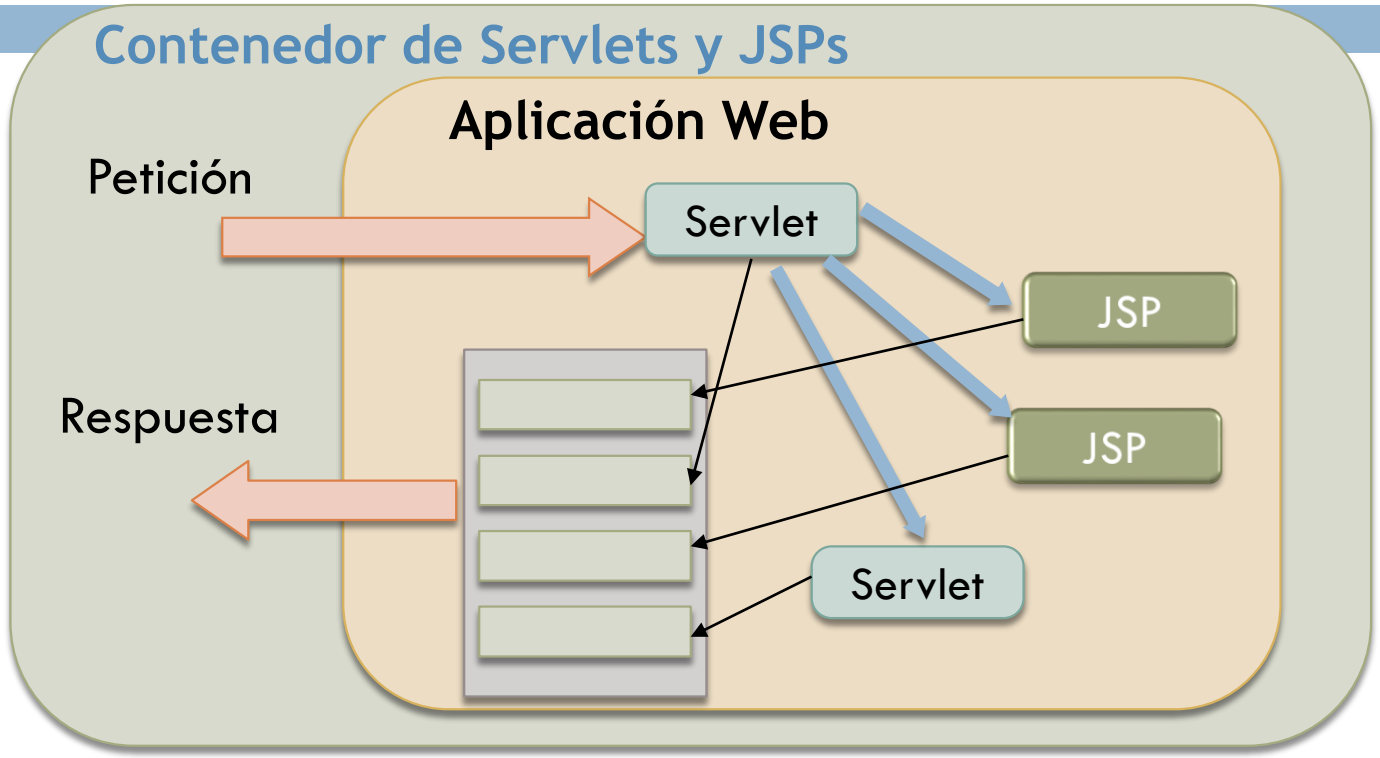
Aplicaciones Web dinámicas

Escola Tècnica Superior d'Enginyeria

- Los ***Servlets son módulos que permiten sustituir o utilizar el lenguaje Java en lugar de los*** programas ***CGI escritos en otros lenguajes como C/C++ o Perl*** *(un programa CGI es una aplicación que corre en el servidor y da respuesta a acciones remotas arrancando un proceso por petición).*
- Los servlets permiten generar páginas HTML dinámicas, es decir, páginas HTML cuyo contenido puede variar (según las acciones del cliente) y que por lo tanto no pueden almacenarse en un fichero en el servidor.
- ***No tienen entorno gráfico ya que se ejecutan en el servidor. Reciben unos datos y*** su salida o respuesta son principalmente ficheros de texto HTML.

Interacción entre componentes Web

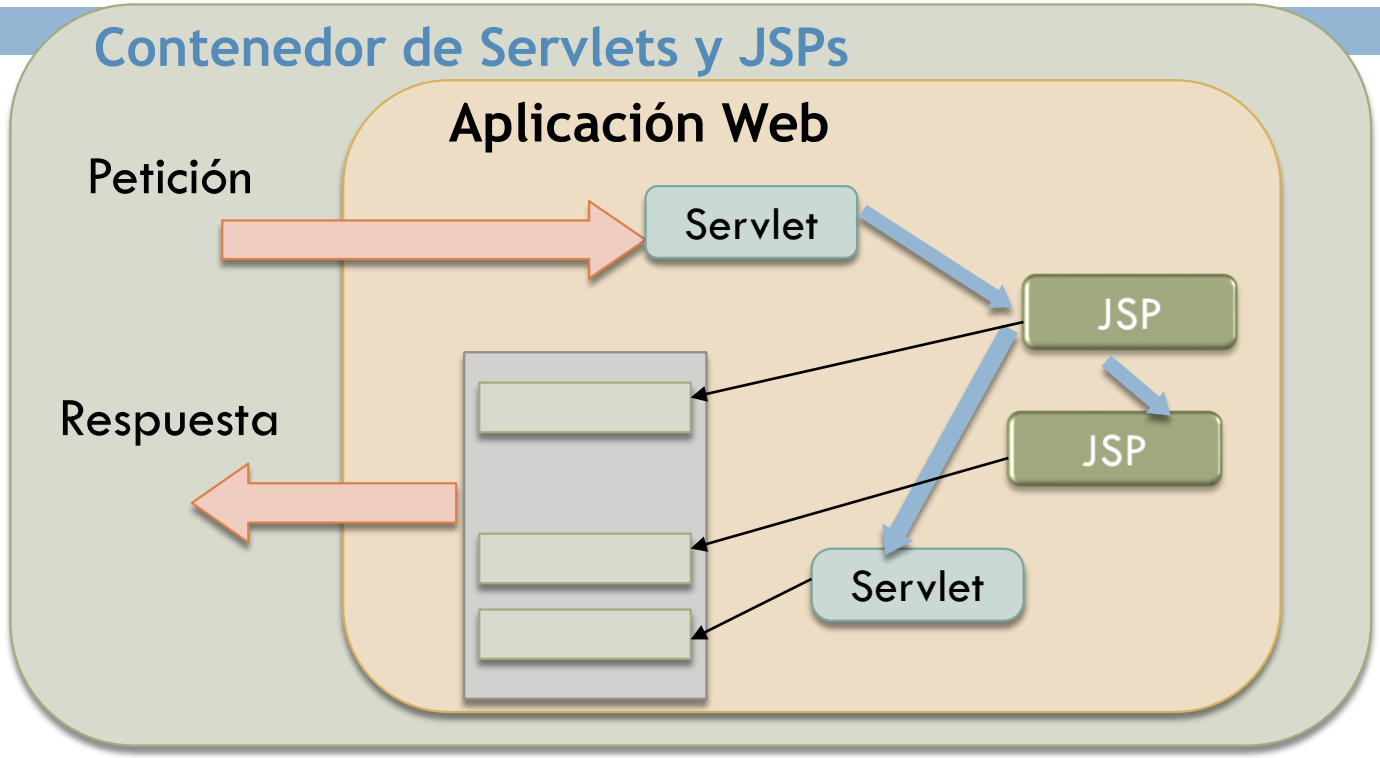
Escola Tècnica Superior d'Enginyeria



- Ejemplo en el que un servlet articula una serie de componentes para generar de forma colaborativa una respuesta.

Interacción entre componentes Web II

Escola Tècnica Superior d'Enginyeria



- Ejemplo en el que un servlet delega en otro componente la generación de la respuesta

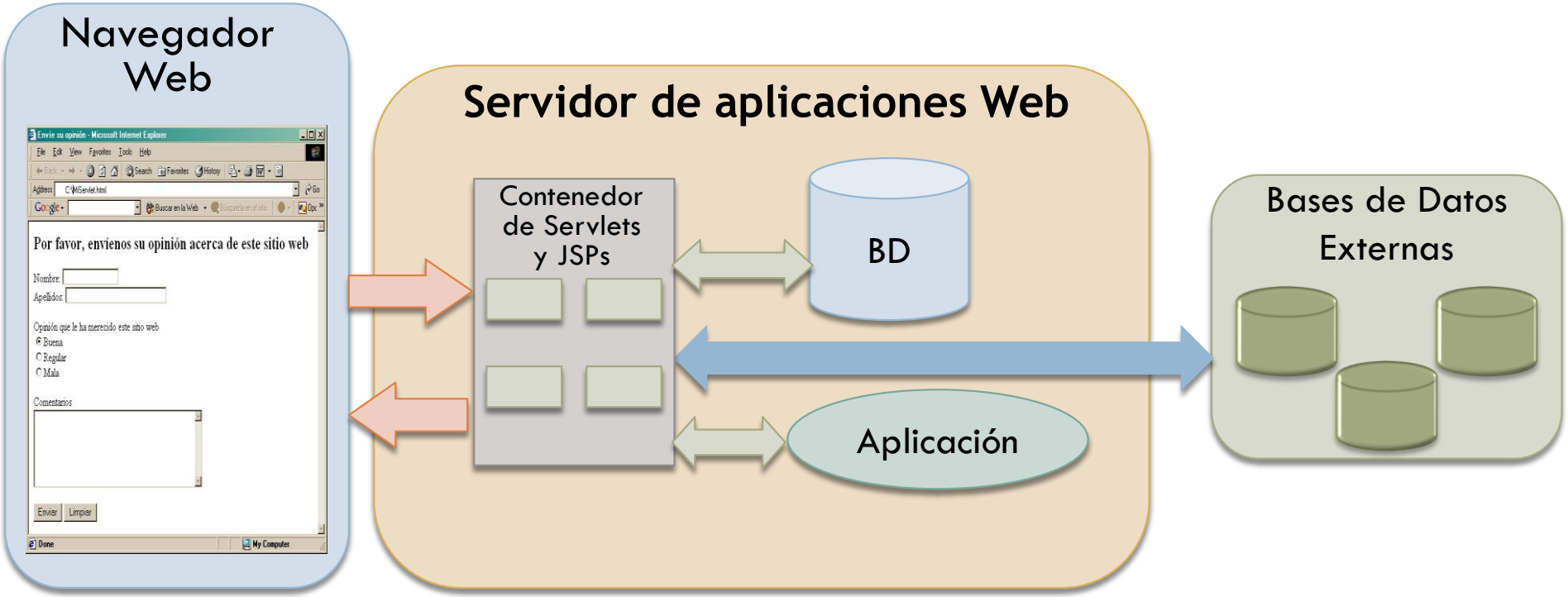
Introducción a los Servlets

Escola Tècnica Superior d'Enginyeria

- Los Servlets son la alternativa Java a los CGIs.
- Actúan como **capa intermedia** entre:
  - Petición proveniente de un Navegador Web u otro cliente HTTP
  - Bases de Datos o Aplicaciones en el servidor HTTP
- Son aplicaciones Java especiales, que extienden la funcionalidad del servidor HTTP. Se dedican a:
  - Leer los datos enviados por el cliente.
  - Extraer cualquier información útil incluida en la cabecera HTTP o en el cuerpo del mensaje de petición enviado por el cliente.
  - Generar dinámicamente resultados.
  - Formatear los resultados en un documento HTML.
  - Establecer los parámetros HTTP adecuados incluidos en la cabecera de la respuesta (por ejemplo: el tipo de documento, cookies, etc.)
  - Enviar el documento final al cliente.

Capa Intermedia

Escola Tècnica Superior d'Enginyeria



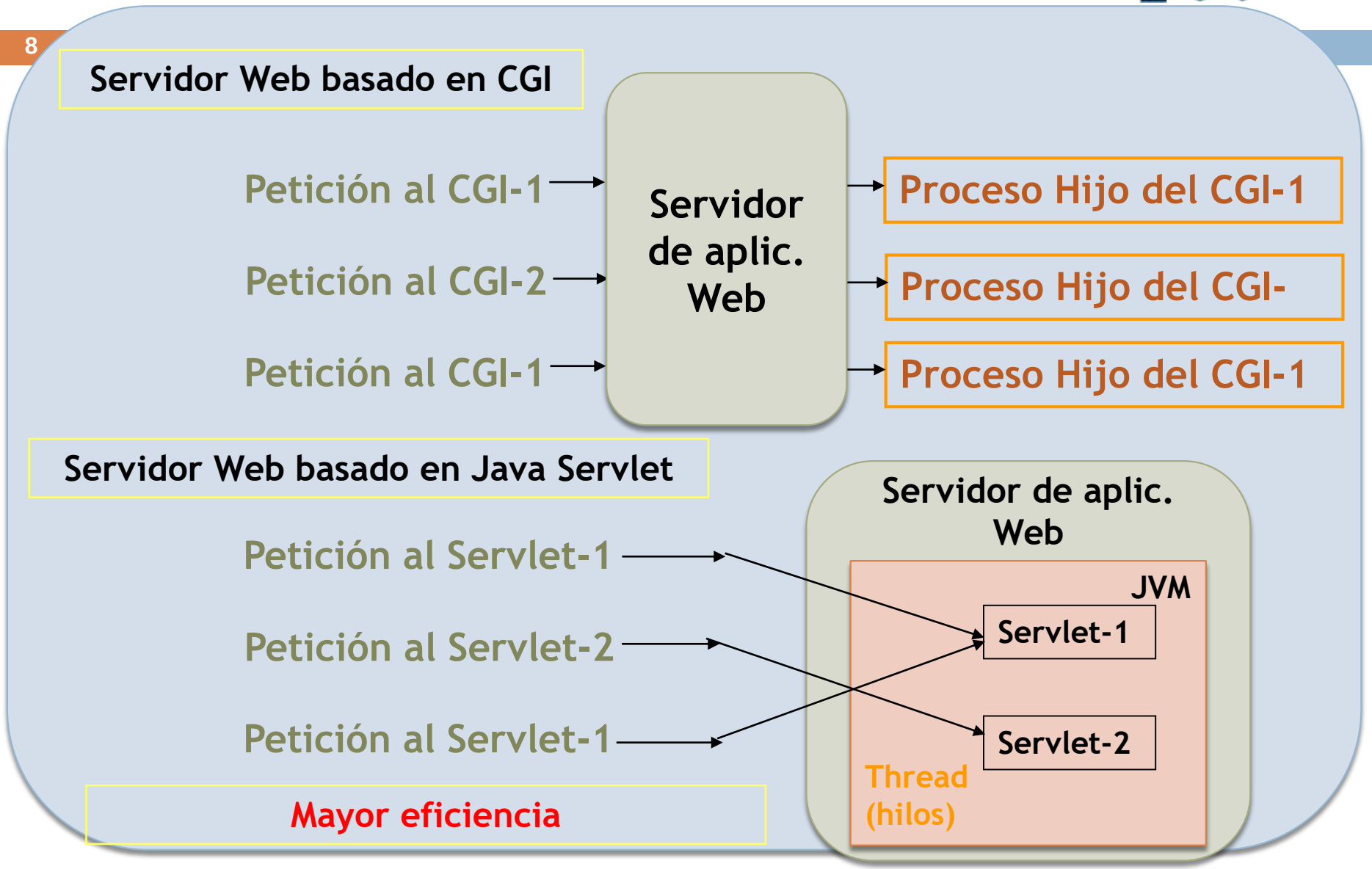
Escola Tècnica Superior d'Enginyeria

Ventajas de los Servlets

- Eficiencia.
  - Cada petición por parte de un cliente crea un hilo, y no un nuevo proceso como ocurría con los CGIs tradicionales.
- Potencia.
  - Son programados en Java, por lo que se puede emplear todas las clases y herramientas disponibles para esta plataforma.
- Seguridad.
  - Controlada por la máquina virtual de Java.
  - La mayoría de problemas de seguridad encontrados en los CGIs no aparecen en los Servlets.
- Portabilidad.
  - Ejecutado sobre cualquier SO y en la mayoría de servidores Web.
  - Son independientes de la plataforma y del servidor.
- Precio.
  - Normalmente todo el software necesario es gratis.

Escola Tècnica Superior d'Enginyeria

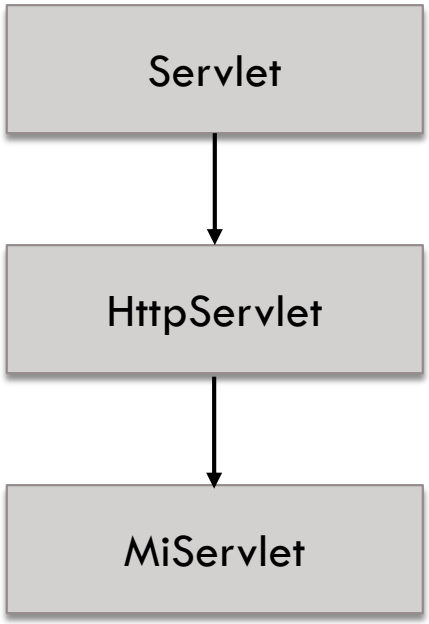
Comparación CGI / Servlets



Arquitectura de los Servlets

- Para implementar los servlets se hace uso de las clases:
  - javax.servlet: entorno básico
  - javax.servlet.http: extensión para servlets http.

Hay que programar una clase que extiende a javax.servlet.http.HttpServlet



Tomcat / Jboss

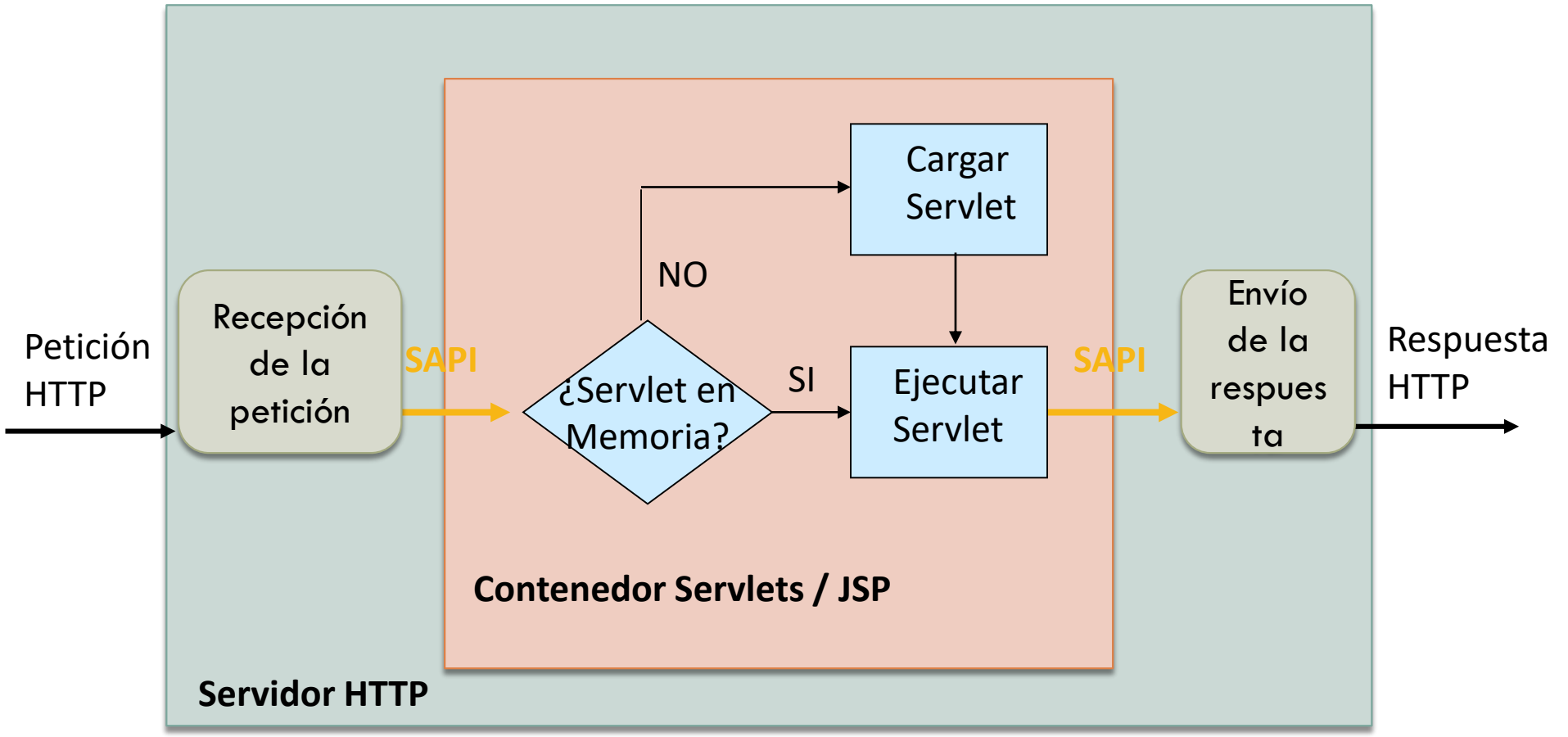
- El principal servidor (gratuito) de servlets y JSP es "Tomcat" de Apache http://jakarta.apache.org/tomcat
- **Tomcat** es un servidor web con soporte de servlets y JSPs.
- **JBoss** es un servidor de aplicaciones. Tiene un contenedor web para ejecutar aplicaciones con servlets y JSPs y todo lo que pueda montarse encima de éstos, y también un contenedor EJB para montar Enterprise Java Beans.
- El contenedor provee el entorno de ejecución para todos los servlets.
- Apache tomcat es tan sólo un web container, y no tiene el EJB container.

Ciclo de vida de un servlet

- El servidor recibe una petición HTTP que ha de ser manejada por un servlet.
- El servidor comprueba si existe una **instancia** creada en memoria de la clase servlet correspondiente. Si no, la crea.
- Las peticiones posteriores de otros usuarios utilizarán la misma instancia.
- El objeto servlet permanece en memoria mientras el servidor siga en funcionamiento y no sean desactivados por el programa que controla el servidor.

Escola Tècnica Superior d'Enginyeria

Diagrama del ciclo de vida



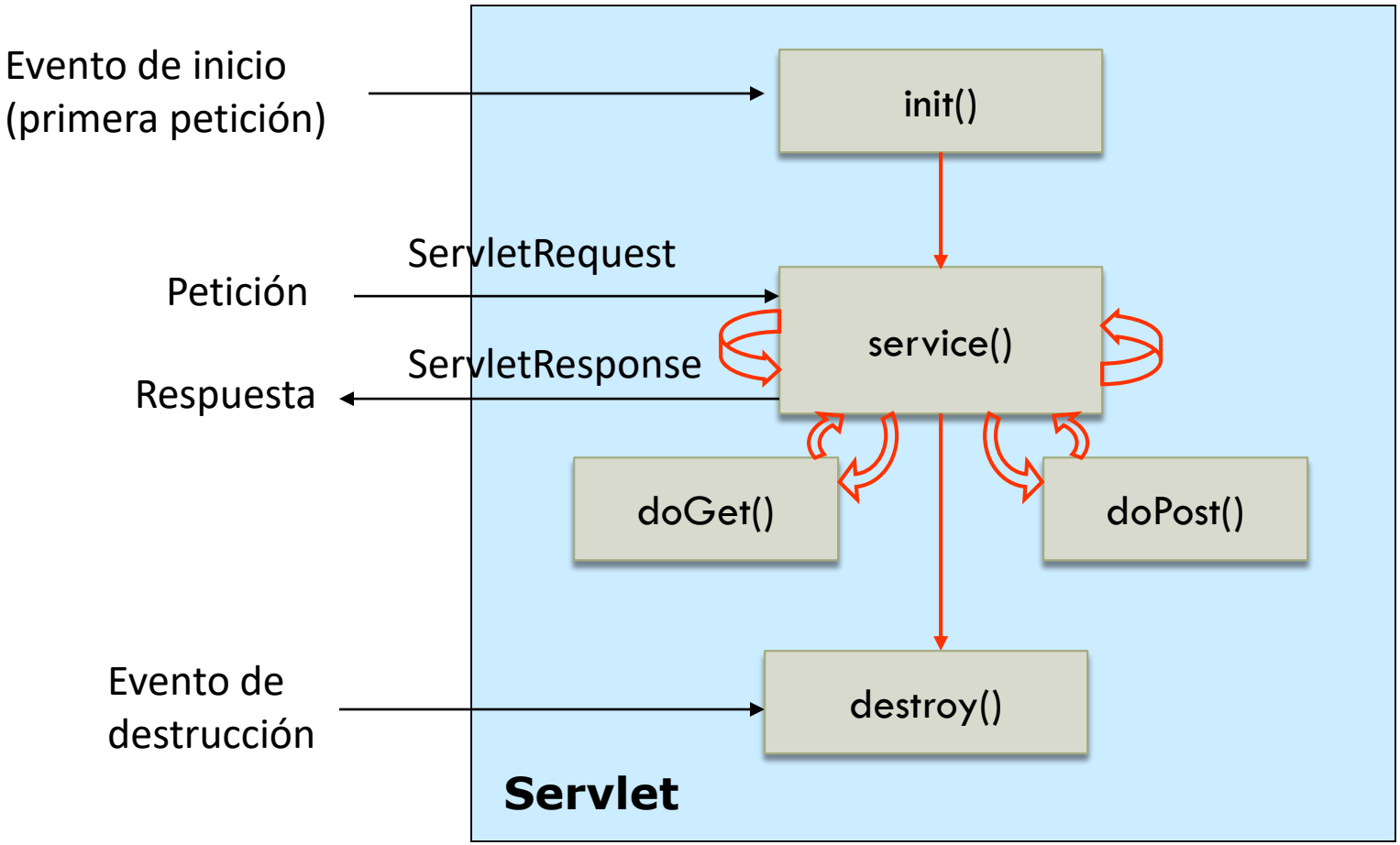
Ya hemos comentado antes que la interfaz SAPI define una manera estándar para que las peticiones HTTP y respuestas sean procesadas por esta clase Java (independiente del servidor).

Escola Tècnica Superior d'Enginyeria

Métodos implícitos (ciclo de vida)

- **init**
  - Se ejecuta una vez (el servlet se carga en memoria y se ejecuta sólo la primera vez que es invocado. El resto de peticiones generan un hilo).
- **service** (no debe sobreescribirse)
  - Se ejecuta cada vez que se produce una nueva petición.
  - Dentro de esta función se invoca a doGet o a doPost.
- **doGet** y **doPost**
  - Manejan las peticiones GET y POST.
    - Incluyen el código principal del servlet
  - La ejecución del servlet finalizará cuando termine la ejecución de estos métodos.
- **destroy**
  - Se invoca cuando el servidor decide eliminar el servlet de la memoria
  - (NO después de cada petición).

Ciclo de ejecución de los métodos



Escola Tècnica Superior d'Enginyeria

Pasos habitual en un Servlet

1. Obtener campos de la cabecera (usando métodos de HttpServletRequest)
2. Obtener los parámetros pasados desde el cliente (usando métodos de HttpServletRequest)
3. Realizar la tarea en función de los parámetros y/o de los campos de cabecera (acceder a una base de datos, actualizar un objeto de sesión, …)
4. Establecer campos de cabecera (usando métodos de HttpServletResponse)
5. Escribir la página en el flujo de salida (obtenido de HttpServletResponse)

Escola Tècnica Superior d'Enginyeria

Objetos implícitos (I)

- Existen una serie de objetos implícitos, disponibles dentro de nuestros servlets (instanciados por el propio contenedor de servlets y JSP)
- Objeto **request**
  - Es una instancia de HttpServletRequest (javax.servlet.http.HttpServletRequest)
  - Recoge la información enviada desde el cliente
- Objeto **response**
  - Es una instancia de HttpServletResponse (javax.servlet.http.HttpServletRequest)
  - Organiza los datos enviados al cliente
- Objeto **session**
  - Es una instancia de HttpSession (javax.servlet.http.HttpSession)
  - Almacena información con ámbito de sesión

Escola Tècnica Superior d'Enginyeria

Objetos implícitos (II)

- Objeto **application**
  - Es una instancia de ServletContext (javax.servlet.ServletContext)
  - Almacena información con ámbito de aplicación

- Objeto **out**

La referencia al objetp Out se obtiene con "response.getWriter()"

  - Es una instancia de PrintWriter (java.io.PrintWriter)
  - Escribe contenido dentro de la página HTML

- Objeto **config**

La referencia al objeto Config se obtiene con "getServletConfig()"

  - Es una instancia de ServletConfig (javax.servlet.ServletConfig)
  - Contiene información relacionada con la configuración del servlet

Métodos más importantes de la clase HttpServlet


```
// Metodo llamado cuando el servlet  se carga en el contenedor
public void init ( ) throws ServletException

// Para tratar una peticion GET
// El primer argumento sirve para obtener informacion sobre la peticion
// El segundo argumento sirve para establecer  la respuesta
protected void doGet ( HttpServletRequest req , HttpServletResponse resp )
            throws ServletException , IOException

// Para tratar una peticion POST
protected void doPost ( HttpServletRequest req , HttpServletResponse resp )
            throws ServletException , IOException

// Metodo llamado cuando el servidor borra el objeto servlet
public void destroy ( )
```

Escola Tècnica Superior d'Enginyeria

Estructura básica


```
import java.io.*;                // Para PrintWriter
import javax.servlet.*;          // Para ServletException
import javax.servlet.http.*;     // Para HttpServlet*

public class PlantillaServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request,
                        HttpServletResponse response)
         throws ServletException, IOException          {
   // El objeto "request" (parámetro de entrada) se utiliza para leer
   // la cabecera HTTP, cookies, datos enviados (GET o POST)
   // El objeto "response" (parámetro de salida) para fijar la respuesta

      PrintWriter out = response.getWriter();
      // out Se utiliza para enviar el contenido al cliente
   }
   //Método que se llama cuando hay una petición POST
   protected void doPost(HttpServletRequest request,
                         HttpServletResponse response)
                  throws ServletException, IOException  {}
}
```

Escola Tècnica Superior d'Enginyeria

Ejemplo: generar una página estática


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

public class HolaMundo extends HttpServlet {
protected void doGet(HttpServletRequest request,
                              HttpServletResponse response)
                              throws ServletException, IOException  {

// Informamos al navegador del contenido de la respuesta
response.setContentType("text/html");

// Obtención del flujo de salida
PrintWriter out = response.getWriter();

// Escritura de la página html
out.println( "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD " + "HTML 4.0 Transitional//EN\">" +
          "<html> <head><title>Hola Mundo</title></head>" +
          "<body> <h1>Hola Mundo</h1> </body></html>");}

// Método que se llama cuando hay una petición POST
protected void doPost(HttpServletRequest request,
                              HttpServletResponse response)
                    throws ServletException, IOException  {}  }
```

Ejemplo (I)


```
public class MuestraMensaje extends HttpServlet {
    private String mensaje;
    private String mensaje_por_defecto = "No hay mensaje";
    private int repeticiones = 1;

// Inicialización del servlet (sólo se ejecuta en la primera petición)
public void init() throws ServletException {

// La interfaz ServletConfig proporciona la información que necesita el servlet
    para inicializarse (ejemplo con varios parámetros de inicialización)

          ServletConfig config = getServletConfig();
          mensaje = config.getInitParameter("mensaje");
          if (mensaje == null) {
              mensaje = mensaje_por_defecto;
          }
          try
          {
              // Inicializa la variable global repeticiones en la primera ejec.
              String repetir_cad =  config.getInitParameter("repeticiones");
              repeticiones = Integer.parseInt(repetir_cad);
          } catch(NumberFormatException nfe) {}
    }
```

Escola Tècnica Superior d'Enginyeria

Ejemplo (II)

(Continuación)

```
...
    // En doGet se crea el código de la página Html
    public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                    throws ServletException, IOException {

          // Informa del tipo de fichero de respuesta
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          String titulo = "MuestraMensaje";
          out.println("HTML 4.0 Transitional//EN\">" +
          "<html><head><title>" +titulo + "</title></head>" +
          "<body bgcolor=\"#FDF5E6\">\n" +
          "<h1 align=\"center\">" + titulo + "</h1>");

          // Se muestra en pantalla el mensaje tantas veces como el valor de la
          // variable global repeticiones (inicializada en init())
          for(int i=0; i < repeticiones; i++)
                    out.println(mensaje + "<br>");
          out.println("</body></html>");
    }
}
```

Datos enviados desde el cliente

- El objeto **request** contiene todos los datos enviados desde el cliente al servidor.
- Todos los servlets implementan la interfaz ServletRequest, que define métodos para acceder a:
  - Los parámetros enviados por el cliente dentro de la URL o dentro del cuerpo del mensaje (p.e. a partir de un formulario)
  - Los valores de la cabeceras HTTP del mensaje
    - Cookies
    - Información sobre el protocolo
    - Content-Type
    - Si la petición fue realizada sobre un canal seguro SSL
    - etc.
  - Los datos de otras entradas.

Escola Tècnica Superior d'Enginyeria

Datos de un formulario

- La forma de leer los datos enviados desde un formulario es independiente del método de envío (GET o POST).
- String request.**getParameter**("nom_var")
  - Devuelve el valor (decodificado URL-encoded) encontrado en la primera ocurrencia de la variable dentro de los datos enviados por el cliente.
  - Devuelve null si la variable no ha sido enviada.
- String request.**getParameterValues**("nom_var")
  - Devuelve un array de valores (decodificados URL-encoded) con todos los valores asociados a la variable (SELECT multiple). Si sólo aparece un vez, devuelve un array de un elemento.
  - Devuelve null si la variable no ha sido enviada.
- Enumeration request.**getParameterNames**()
  - Devuelve una enumeración con los nombres de las variables enviadas.

Escola Tècnica Superior d'Enginyeria

Ejemplo: Recogida de datos de un formulario


```
// Formulario que carga el cliente: donde mete los datos
<html>
<head><title>Formulario</title>
</head>
<body>
// Asociamos el formulario a la clase Ejemplo (servlet)
<form action="Ejemplo" method="POST">
   // Se crea en el formulario una caja de texto
   Nombre:<input type="text" name="nombre"><br>
   // Se crea en el formulario una lista desplegable a elegir
   Favoritos:<select name="favoritos" multiple>
        <option value="cgi">CGI</option>
        <option value="php">PHP</option>
        <option value="servlet">Servlet</option>
        <option value="jsp">JSP</option>
        <option value="asp">ASP</option>
   </select>
</form>
</html>
```

Ejemplo: Recogida de datos de un formulario (Uno a uno)


```
public class Ejemplo extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                     throws ServletException, IOException  {
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.println("<html><head><title>Datos recibidos de form"
        + "</title></head><body>\n" +
        "<h1>Nombre:</h1>" + request.getParameter("nombre") +
        "<h1>Lenguajes favoritos:</h1>");
     String[] lengj= request.getParameterValues("favoritos");
     for (int i = 0; i < lengj.length; i++ )
        out.println( lengj[i] + "<br>" );
     out.println("</table></body></html>");
   } // Fin doGet
} // Fin clase
```

Recogida de datos de la cabecera

- Existen un conjunto de funciones para extraer los valores de algunos parámetros particulares:
  - Cookie[] **getCookies** ()
    - Extrae las cookies enviadas por el cliente.
  - String **getMethod** ()
    - Método utilizado en la petición (su valor es GET o POST).
  - String **getContentLength** ()
    - Longitud de los datos enviados por el cliente (utilizando el método POST) tras la cabecera HTTP.
  - String **getContentType** ()
    - Devuelve el tipo MIME de los datos enviados tras la cabecera.
  - String **getProtocol** ()
    - Devuelve la versión del protocolo HTTP (HTTP/1.0 o HTTP/1.1) utilizado por el cliente en la petición.

Escola Tècnica Superior d'Enginyeria

Recogida de datos de la cabecera


```
// Ejemplo de recogida de datos de la cabecera
public class ContenidoCabeceraHTTP extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Muestra el contenido de la cab.";
      out.println("<html><head><title>" + titulo +
         "</title></head><body>\n" +
         "<h1 align=\"center\">" + title + "</h1>\n" +
         "<b>Método de envío: </b>" +
         request.getMethod() + "<br>\n" +
         "<b>Protocolo: </b>" +
         request.getProtocol() + "<br><br>\n" +
         "<table border=\"1\" align=\"center\">\n" +
         "<tr bgcolor=\"#FFAD00\">\n" +
```

Recogida de datos de la cabecera


```
        "<TH>Nombre del parámetro<th>Valor");
        Enumeration nombres_par = request.getHeaderNames();
        while(nombres_par.hasMoreElements()) {
           String nom_cab = nombres_par.nextElement();
           out.println("<tr><td>" + nom_cab);
           out.println("<td>" + request.getHeader(nom_cab));
        }
        out.println("</table>\n</body></html>");
    }
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Datos enviados al cliente (respuesta)

- El objeto **response** representa los datos enviados desde el servidor al cliente
  - Se emplea, no sólo para escribir el contenido de la página enviada al cliente, sino también para organizar la cabecera HTTP, enviar cookies, etc.
- Todos los servlets implementan el interfaz de ServletResponse, que permite:
  - Acceder al canal de salida
  - Indicar el tipo de contenido del mensaje de respuesta
  - Indicar si existe buffer de salida
  - Establecer la localización de la información
- HttpServletResponse extiende a ServletResponse:
  - Código de estado del mensaje de respuesta
  - Cookies

Escola Tècnica Superior d'Enginyeria

Estado de la respuesta

Al generar la respuesta en el método doGet o doPost se pueden realizar llamadas a los siguientes métodos:

- **sendError** (int *codigo*, String *mensaje*) // Envío de error
  - Manda un código de estado de error (4XX), y escribe el contenido de *mensaje* en el cuerpo del documento HTML.
- **sendRedirect** (String *url*) // Envío de redirección
  - Redirecciona el navegador del cliente hacia la dirección *url*
  - Manda un código de estado SC_MOVED_TEMPORALY, y asigna al parámetro "Location" de la cabecera HTTP la dirección *url*

Parámetros de la cabecera HTTP

- Para fijar cualquier parámetro de la cabecera:
  - **setHeader** (String nombre_param, String valor_param)

```
response.setHeader("Cache-Control", "no-cache");
```

- Para ciertos parámetros, existen funciones especiales:
  - **setContentType** (String *codigo_MIME*)

    Fija el código MIME de la respuesta (Content-Type)

```
response.setContentType("text/html");
```

  - **addCookie** (Cookie *la_cookie*)
    - Envía una cookie al cliente.

Cuerpo del mensaje

- El cuerpo del mensaje es generado a partir de los objetos:
  - **PrintWriter**
    - La referencia se extrae con **response.getWriter()**
    - Cuando el código generado es texto HTML (o XML, o plano)

```
PrintWriter out = response.getWriter();
out.println("..."); out.flush(); out.close();
```

  - **ServletOutputStream**
    - La referencia se extrae con **response.getOutputStream()**
    - Cuando el código generado es binario (p.e. una imagen)

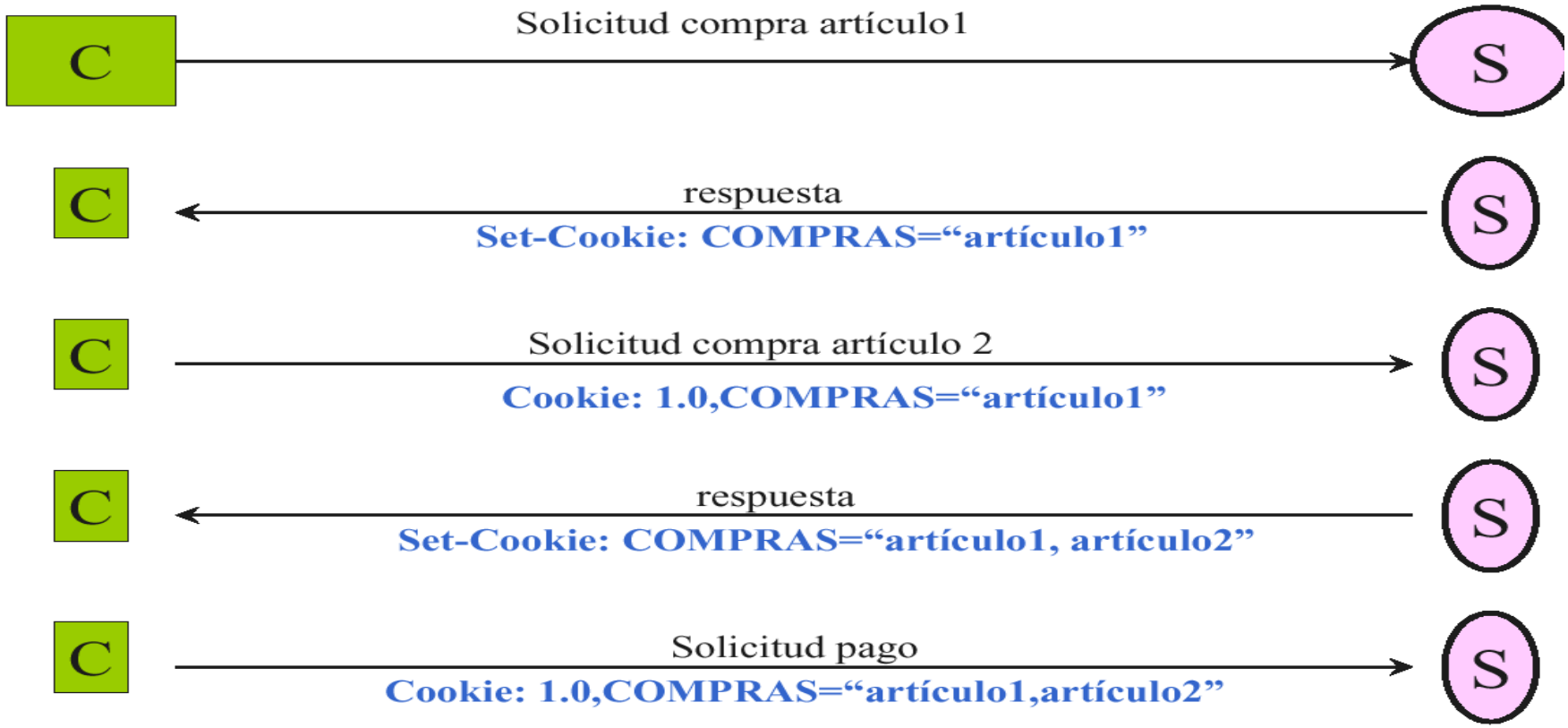
```
ServletOutputStream out = response.getOutputStream();
```

Formas obtener la información de la petición

RESUMEN:

- HTTP es un protocolo "*sin estado*"
  - Cada vez que un cliente pide una página Web, abre una conexión separada con el servidor Web y el servidor no mantiene automáticamente *información contextual* acerca del cliente
- Servlets
  - Permiten obtener y mantener una determinada información acerca de un cliente
  - La información es accesible a diferentes servlets o entre diferentes ejecuciones de un mismo servlet
- Tres soluciones típicas:
  - Cookies
  - Seguimiento de sesiones
  - Reescritura de URLs

Cookies



Las cookies se van pasando entre cliente y servidor en cada solicitud / respuesta

Enviando/Recibiendo Cookies

- Para enviar cookies al cliente se crea un objeto de la clase Cookie, y se invoca el método **addCookie** del objeto response pasándole como parámetro dicha cookie.

```
Cookie c = new Cookie("nombre", "valor");
c.setMaxAge(...); // Segundos de vida del cookie
response.addCookie(c);
```

- Para leer las cookies se emplea el método **getCookies** del objeto request. Éste devuelve un array con todos las cookies recibidas del cliente.

```
Cookie[] cookies_recb = request.getCookies();
if (cookies_recb != null)
    for(int i=0; i<cookies_recb.length; i++) {
          if (cookies_recb[i].getName().equals("alquiler"))
          && (cookies_recb[i].getValue().equals("coche"))
          {cookies_recb[i].setMaxAge(0)//Eliminaría el cookie
        } //fin del if
    } // fin del for
} // fin del if
```

Métodos del objeto Cookie (I)

- public String **getName**() /
  public void **setName** (String *nombre_cookie*)
  - Extrae / fija el nombre del cookie. La función setName raramente se utiliza, ya que el nombre de la nueva cookie normalmente se fija en el constructor del objeto.
- public String **getValue**() /
  public void **setValue** (String *valor_cookie*)
  - Extrae / fija el valor de la cookie. La función setValue normalmente no se utiliza (el valor se fija en el constructor).
- public int **getMaxAge**() /
  public void **setMaxAge** (int *segundos_vida*)
  - Extrae / fija el número de segundos que la cookie permanece guardado en el disco del cliente. (-1 elimina la cookie al cerrar el browser y 0 elimina la cookie sin más).

Métodos del objeto Cookie (II)

- public String **getDomain**() /
  public void **setDomain** (String *dominio*)
  - Extrae / fija el dominio de los servidores con acceso a la cookie.
- public String **getPath**() /
  public void **setPath** (String *camino*)
  - Extrae / fija el directorio raíz (virtual) de las páginas con acceso a la cookie.
- public boolean **getSecure**() /
  public void **setSecure** (boolean *flag_seguridad*)
  - Extrae / fija el parámetro de seguridad. Si *flag_seguridad* vale true, la cookie sólo será enviada si la conexión es segura (SSL).

Ejemplo


```
public class UtilidadesCookie
{
   // Ejemplo para devolver el valor de una cookie (nom_cookie)
   public static String ExtraeValor (Cookie[] cookies,
                                     String nom_cookie )
   {
         String valor = "";
         if ( cookie != null )
                  // Recorre las cookies hasta encontrar la buscada
                  for ( int i=0; i<cookies.length; i++) {
                     Cookie cookie=cookies[i];
                     if ( nom_cookie.equals(cookie.getName()) )
                           valor = cookie.getValue();
                  }
         return valor;
   }
```

Ejemplo


```
// Ejemplo que nos dice si una cookie es segura (SSL)
public static boolean EsSegura ( Cookie[] cookies,
                                 String nom_cookie )
{
     boolean segura = false;
     if ( cookie != null )
            for ( int i=0; i<cookies.length; i++) {
               Cookie cookie = cookies[i];
               if (nom_cookie.equals(cookie.getName()) )
                   segura = cookie.getSecure();
            }
     return segura;
}
} // Fin UtilidadesCookie
```

3.2 Seguimiento de Sesiones

RESUMEN - SESIONES:

- HTTP es un protocolo sin manejo de estados
  - Tras la respuesta, el servidor cierra inmediatamente la conexión
  - Los servidores HTTP responden a cada solicitud del cliente sin relacionar tal solicitud con alguna solicitud previa o siguiente.
    - El protocolo HTTP no maneja un estado de cada conexión realizada por el mismo usuario, sea cual sea la versión HTTP.
  - No existe el concepto de sesión.
- Las sesiones son fundamentales en las aplicaciones Web. Permiten:
  - Definir varios estados distintos en la aplicación.
  - Ampliar el contexto de las solicitudes y respuestas.
    - Los clientes y servidores intercambian información sobre el estado de la aplicación

Datos asociados a la sesión

- El servidor almacenará la información necesaria para llevar el seguimiento de la sesión.
  - Identificador de la sesión.
  - Identificador del usuario en sesión.
  - Tiempo de expiración de la sesión.
  - Dirección donde se encuentra localizado el cliente.
  - Variables asociadas a la sesión.
  - Otras variables temporales.
- Por la misma naturaleza del HTTP es imposible asegurar la existencia o la ausencia de una sesión.
  - Debe establecer un proceso que revise periódicamente los tiempos de expiración de cada proceso.
  - Debe eliminar los datos asociados a la sesión si ya excedió el tiempo.

Escola Tècnica Superior d'Enginyeria

Objeto HttpSession

- Las sesiones se implementan a través de objetos de la clase HttpSession, creados por el contenedor cuando se inicia una sesión para un nuevo usuario.
  - Para extraer la referencia a este objeto desde un servlet:

```
HttpSession mi_sesion = request.getSession(true);
```

- Las sesiones se asocian al cliente, vía cookies o reescribiendo la URL.
  - El sistema localiza el identificador de la sesión incluido dentro de la cookie, o incluido en la información extra de la URL de la petición. Cuando el identificador no corresponde a un objeto de tipo sesión previamente almacenado, crea una nueva sesión.
- Las sesiones se utilizan para almacenar variables que transmiten su valor a lo largo del conjunto de páginas visitadas por el cliente durante la sesión.

Escola Tècnica Superior d'Enginyeria

API del objeto sesión (I)

- public void **setAttribute** (String *nombre*, Object *valor*)
  - Registra una nueva variable dentro de la sesión (*nombre* y *valor* son el nombre y el valor de la variable).
- public Object **getAttribute** (String *nombre*)
  - Extrae el valor de una variable previamente registrada.
- public void **removeAttribute** (String *nombre*)
  - Borra una variable de la sesión previamente registrada.
- public Enumeration **getAttributeNames** ( )
  - Extrae el nombre de todas las variables registradas en la sesión
- public String **getId** ( )
  - Devuelve el identificador de la sesión.

Escola Tècnica Superior d'Enginyeria

API del objeto sesión (II)

- public boolean **isNew** ( )
  - Devuelve true si la sesión comienza en esta página.
- public long **getCreationTime** ( )
  - Momento de la creación de la sesión (expresado en milisegundos transcurridos desde el 1 de enero de 1970).
- public long **getLastAccessedTime** ( )
  - Momento del último acceso a una página de la sesión (milisegundos transcurridos desde el 1 de enero de 1970).
- public int **getMaxInactiveInterval** ( ) /
  public void **setMaxInactiveInterval** (int *segundos*)
  - Extrae / fija los segundos que deben transcurrir desde el último acceso para que la sesión sea cerrada.

Escola Tècnica Superior d'Enginyeria

Ejemplo


```
// Control del número de accesos en la misma sesión

HttpSession session = request.getSession(true);
Integer acc = (Integer)session.getAttribute("accesos");
String presentacion;
if (acc == null) {
        acc = new Integer(0);
        presentacion = "Bienvenido, nuevo usuario";
} else {
        presentacion = "Bienvenido de nuevo";
        acc = new Integer(acc.intValue() + 1);
}
session.setAttribute("accesos", acc);
```

Escola Tècnica Superior d'Enginyeria

Reescritura de URLs

- Puede suceder que ciertos clientes no soporten cookies o bien las rechacen
- Solución: Sesiones + Reescritura de URLs
  - El cliente añade ciertos datos extra que identifican la sesión al final de cada URL
    - `http://host/path/servlet/name?jsessionid=1234`
  - El servidor asocia ese identificador con datos que ha guardado acerca de la sesión
- Métodos: encodeURL() y encodeRedirect()
  - Leen un String (URL o URL de redirección) y si es necesario lo reescriben añadiendo el identificativo de la sesión en la URL.
- Algunas Desventajas
  - Se deben codificar todas las URLs referentes al sitio propio
  - Todas las páginas deben generarse dinámicamente