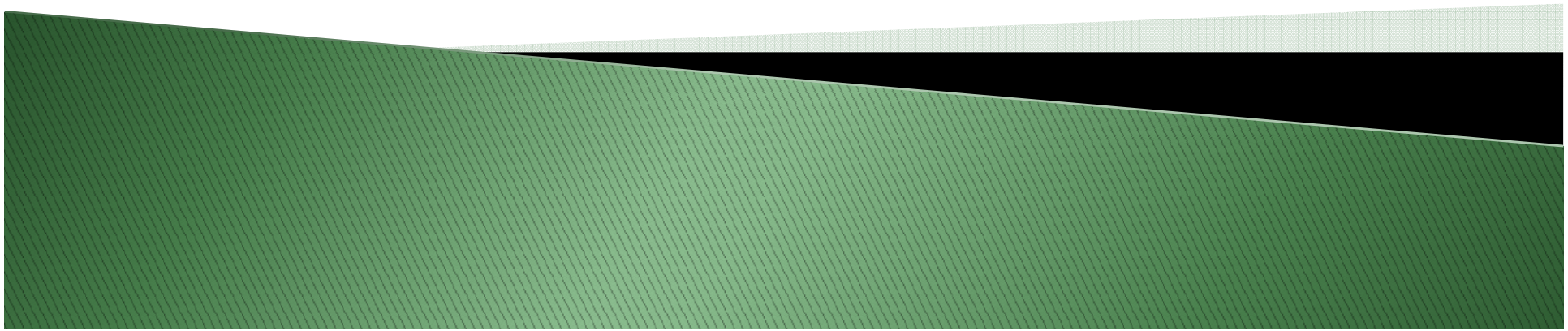
Unidad 6

Servlets

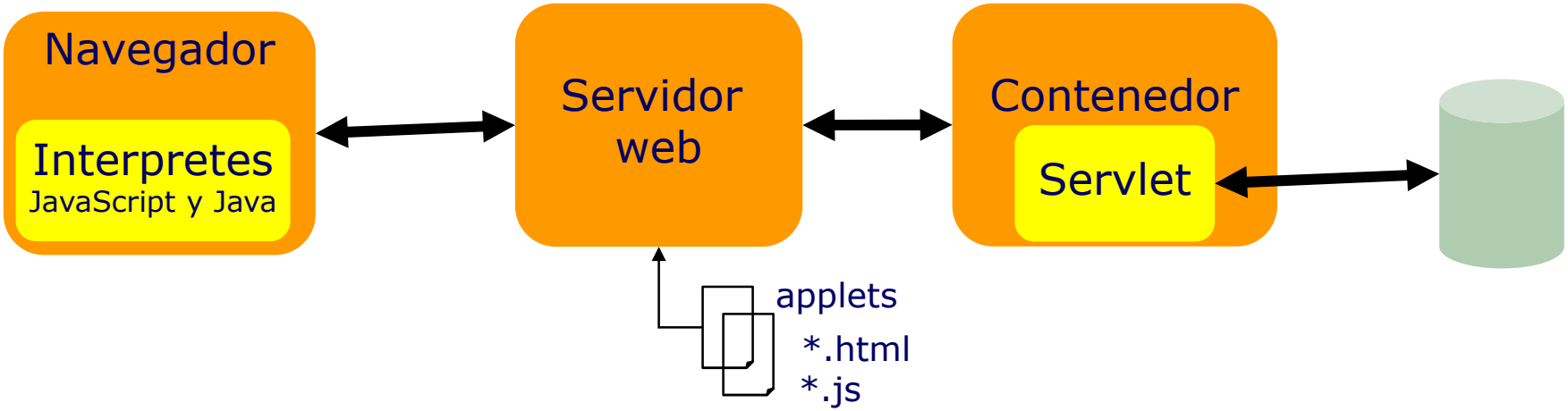
Despliegue de aplicaciones web



Índice de contenidos

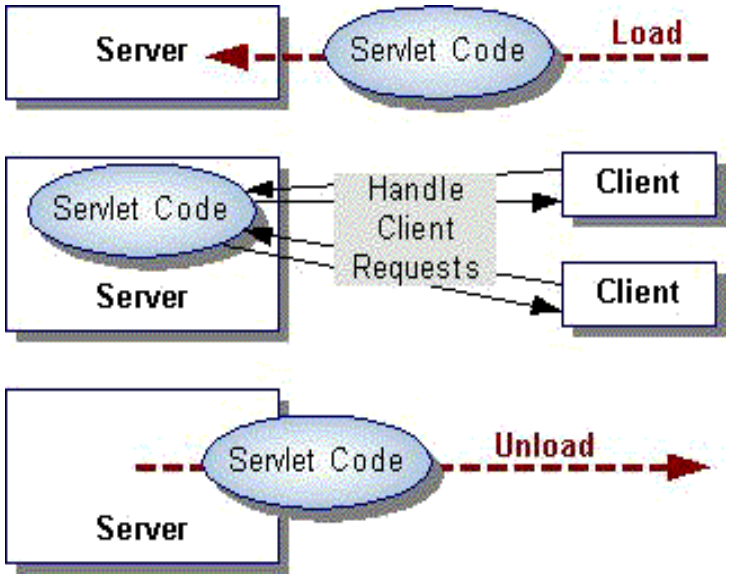
¿Qué es un Servlet?

- Un **servlet** es una especie de pequeño servidor especializado, escrito en Java, que se encarga de recibir peticiones, generar contenido dinámicamente y devolver una respuesta.
- Los servlets se ejecutan dentro de un **contenedor web** (motor de servlets) que traduce las peticiones propias del protocolo a llamadas a objetos Java y traduce las respuestas procedentes de objetos Java a respuestas nativas del protocolo.



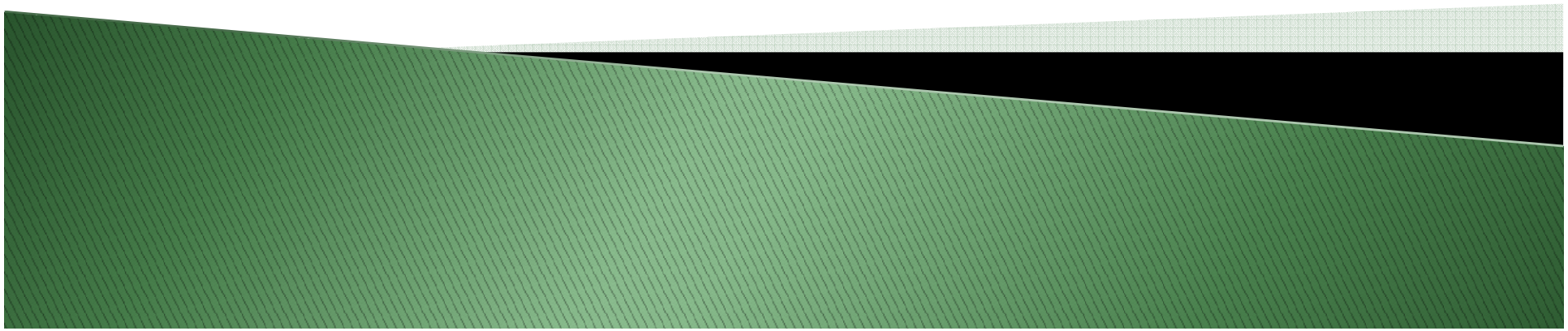
¿Qué es un Servlet?

- El contenedor se encarga de gestionar el **ciclo de vida** del servlet y proporcionarle los servicios que necesite
  - El servidor web redirige las peticiones al contenedor.
  - El contenedor delega las peticiones en un servlet.
- Este comportamiento se configura en el servidor web (según el servidor) y en el contenedor (c).



Servlets

El Contenedor Web



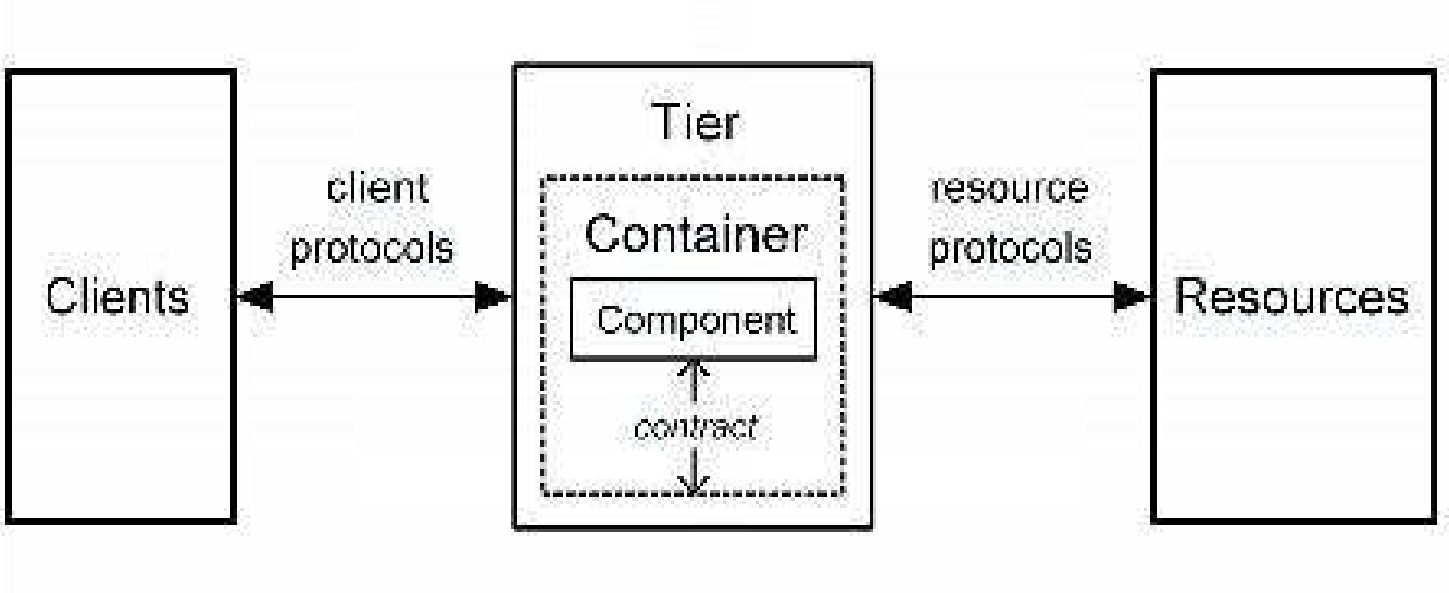
El contenedor web

- Tareas del contenedor:
  - Proporcionar servicios de red para establecimiento de peticiones y respuestas HTTP.
  - Decodificar y codificar peticiones/repuestas en formato MIME.
  - Configurar los servlets en función de los descriptores de despliegue (web.xml).
  - Gestionar el ciclo de vida de los servlets.

- El contenedor web debe implementar las funciones del servidor web (según especificación Servlet).
- Esto se consigue habitualmente conectando **servidor web** + **contenedor** (v.g. *Apache* + *Tomcat*).

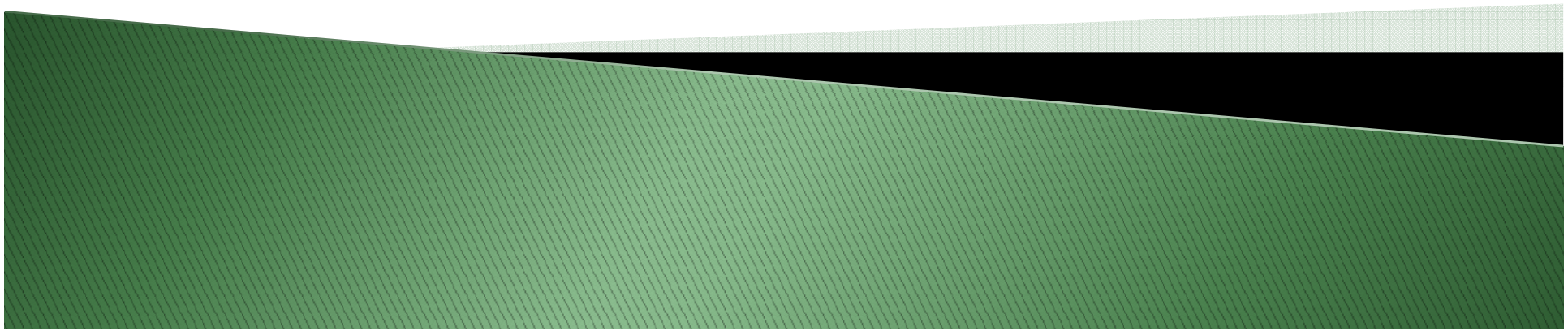
El contenedor web

- La gestión del ciclo de vida de los servlets se basa en un **contrato** (interfaz) definido entre el contenedor y el servlet
- Los servlets deben implementar **javax.servlet.Servlet**
- El contenedor instancia e invoca el servlet vía *Java Reflection*



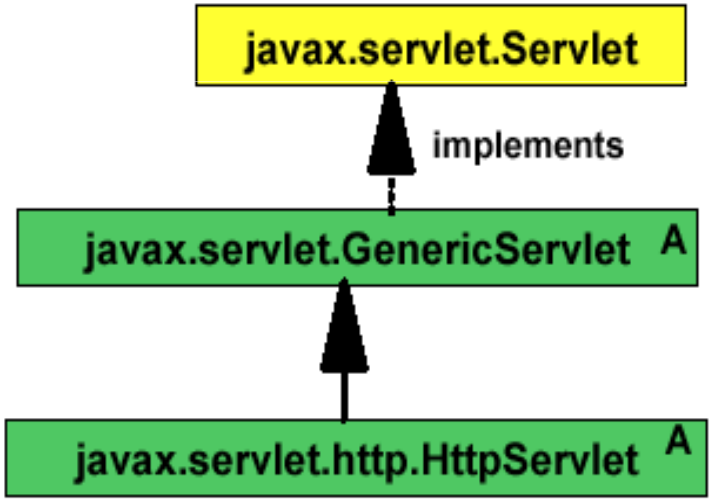
Servlets

Ciclo de vida



La interfaz Servlet

- El API servlet es independiente del protocolo
  - La mayoría de implementaciones son para HTTP.
  - Pueden implementarse par FTP, SMTP, etc.

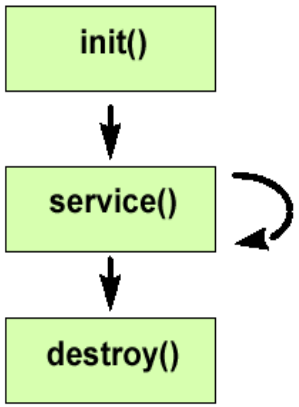


La interfaz Servlet

- Métodos:

```
public void init(ServletConfig config) throws ServletException
public void service(ServletRequest req,ServletResponse res)
throws IOException, ServletException
public void destroy()
public String getServletInfo()
public ServletConfig getServletConfig()
```

- Ciclo de vida:
  - init
  - service
  - destroy



El método init

- Es el método al que se llama justo después de instanciar el Servlet, sólo una vez
- Recibe un objeto de la clase **javax.servlet.ServletConfig** con información para la configuración del servlet, especialmente parámetros de inicio.
- Los **parámetros de inicio** se establecen en el descriptor de despliegue.
- Al instanciarse el servlet mediante mecanismos de reflexión, éste debe tener obligatoriamente un **constructor vacío.** Por no instanciar los Servlets desde dentro de la aplicación no tienen sentido constructores que reciban parámetros
- La especificación asegura que el contenedor no llamará a ningún otro método antes de que *init* haya terminado.

Instanciación de Servlets

- Al contrario que otras tecnologías sólo existe una instancia del servlet para manejar todas las peticiones.
- El servlet se instancia al iniciarse el contenedor o al recibir la primera petición (configurable en **web.xml**).
- Problemas de concurrencia: los servlets no deben tener variables de instancia/clase de escritura.

El método service

- Invocado por el contenedor (concurrentemente) cada vez que se recibe una petición para el servlet
- Antes de invocar a service, el contenedor construye dos objetos:
  - Una ServletRequest que encapsula la petición y sus parámetros
  - Una ServletResponse que encapsula la respuesta (incluye un OutputStream para escribir la respuesta que se devolverá por HTTP)
- Si hay error durante la gestión de una petición, se debe lanzar ServletException o IOException (aunque el contenedor seguirá intentando mandar otras peticiones al servlet)

El método destroy

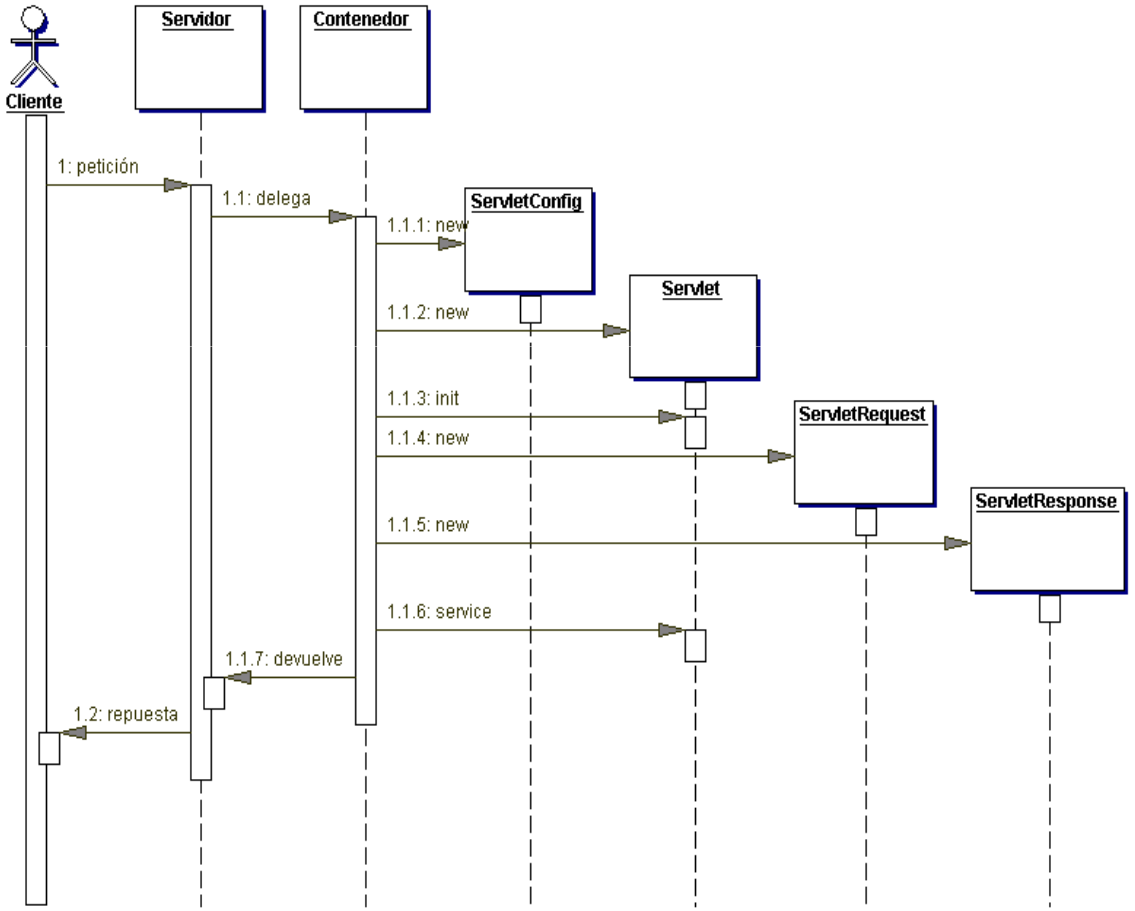
- Llamado una vez por el contenedor antes de destruir el servlet.
- Debe llevar a cabo tareas de limpieza y liberación de recursos adquiridos por el Servlet.
- La especificación asegura que:
  - El contenedor no llamará a ningún método del servlet después de destroy.
  - El contenedor no intentará destruir el servlet antes de que destroy haya terminado.

Otros métodos

- El método getServletInfo() debe devolver un String con información sobre el servlet. Usado por las herramientas de gestión del contenedor para mostrar al administrador.

- El método getServletConfig() es para uso interno del servlet y debe devolver el objeto ServletConfig que se recibió en init.

Ciclo de vida de un Servlet



Ejemplo 1: Servlet

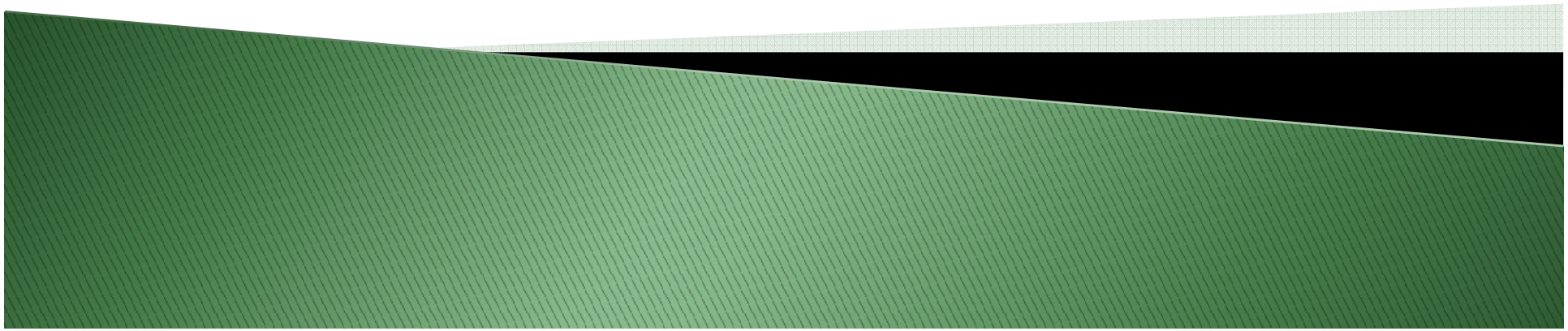
```
import *;
import javax.servlet.*;

public class EjemploServlet implements Servlet {
  private ServletConfig config;

  public void init(ServletConfig cfg) throws ServletException {
    config.getServletContext().log("Iniciando el servlet");
    config = cfg;
  }

  public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
    config.getServletContext().log("Petición recibida desde " + req.getRemoteAddr());
    res.setContentType("text/plain");
    res.getWriter().println("Mensaje desde el servlet de ejemplo");
    res.getWriter().close();
  }

  public void destroy() {
    config.getServletContext().log("Destruyendo el servlet");
  }

  public String getServletInfo() {return "Servlet de Ejemplo";}
  public ServletConfig getServletConfig() {return config;}
}
```

Ejemplo 2: GenericServlet

```
import javax.servlet.*;

public class EjemploServlet extends GenericServlet {

  public void init() {
    log("Iniciando el servlet");
  }

  public void service(ServletRequest req, ServletResponse res) throws IOException {
    log("Petición recibida desde " + req.getRemoteAddr());
    res.setContentType("text/plain");
    res.getWriter().println("Mensaje desde el servlet de ejemplo");
    res.getWriter().close();
  }

  public void destroy() {
    log("Destruyendo el servlet");
  }

  public String getServletInfo() {return "Servlet de Ejemplo usando GenericServlet";}
}
```

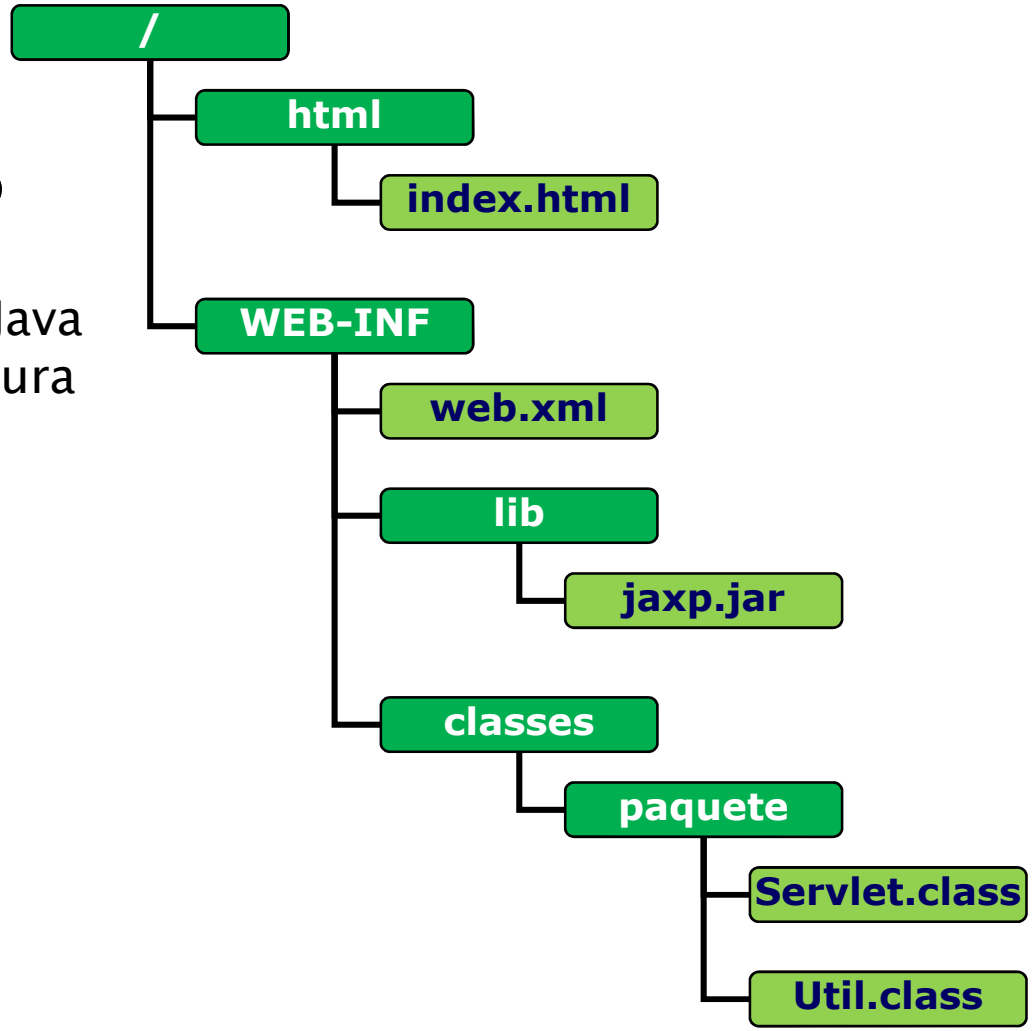
Servlets

Archivos WAR y descriptores de despliegue



Aplicaciones Web con servlets

- Forma estándar de distribuir una aplicación web = Archivo WAR (Web ARchiver)
  - Similares a archivos JAR (Java ARchiver) con una estructura interna determinada,



Descriptores de Despliegue

- Los servicios comunes a cualquier aplicación web se declaran en un archivo especial: el **descriptor de despliegue** (DD)
  - Archivo **web.xml**
- El contenedor es el encargado de proporcionar los servicios descritos en el DD:
  - Seguridad, Mapeos, Archivos de inicio, Configuración de la sesión, etc…
- Características
  - Modelo declarativo, no programático
  - Generados por el programador
  - Aplicación es más portátil y flexible
  - Para conseguir que una aplicación se comporte de maneras diferentes en contenedores distintos sin tener que recompilar

Descriptores de Despliegue – web.xml

- Elemento raíz:
  - <web-app>
- Soporte a herramientas de desarrollo:
  - <icon> ➔ Icono que mostrará la herramienta
  - <display-name> ➔ Nombre que mostrará la herramienta
  - <description> ➔ Descripción de la aplicación/servlet que mostrará la herramienta
  - Etiquetas opcionales que no afectan al comportamiento de la aplicación
- Filtros:
  - <filter> ➔ Define un filtro. <filter-name> y <filter-class> indican el nombre y la clase Java que lo implementa.
  - <filter-mapping> ➔ Define cómo invocar un filtro. <filter-name> debe coincidir con el anterior. <url-pattern> ó <servlet-name> recogen el destino de la invocación.
  - El orden del filtrado es tal cual aparece en el DD.

Descriptores de Despliegue – web.xml

- Declaración de servlets: etiqueta <servlet>, que debe contener:
  - <servlet-class> o <jsp-file> ➔ Nombre completamente cualificado de la clase que implementa el servlet/JSP
  - <servlet-name> ➔ Alias para referirse al servlet a lo largo del DD

- Otras etiquetas opcionales bajo <servlet>:
  - <init-param> ➔ Parámetros de inicio; debe contener <param-name> y <param-value>
  - <load-on-startup> ➔ Fuerza la carga del servlet
  - <security-role-ref> ➔ Mapeos de roles usados en el servlet con roles de servidor (ej: admin-root)

- <servlet-mapping> ➔ Asocia un patrón URL con a un server.
  - <servlet-name> Alias del servlet definido con <servlet>
  - <url-pattern> es el patrón URL asociado.

web.xml – Ejemplo

```
...
<servlet>
  <servlet-name>ServletBD</servlet-name>
  <servlet-class>aplicacion.Servlet1</servlet-
   class>
  <init-param>
    <param-name>driver</param-name>
    <param-value>oracle.jdbc.driver.OracleDriver
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
  <security-role-ref>
    <role-name>admin</role-name>
    <role-link>root</role-link>
  </security-role-ref>
</servlet>
...
```

```
<servlet-mapping>
   <servlet-name>ServletBD</servlet-name>
   <url-pattern>/bbdd/*</url-pattern>
</servlet-mapping>
<!-- Todas las peticiones que comiencen por
    /bbdd/ se enviarán a ServletBD -->
```

Descriptores de Despliegue – otros

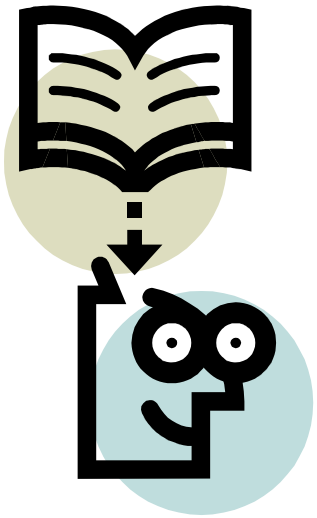
- El contexto puede tener parámetros de inicio que afectan a la aplicación:
  - <context-param> ➔ Declara un parámetro y debe contener <param-name> y <param-value>. Opcionalmente, <description>.
- Mapeo de extensiones de archivos con tipos MIME:
  - <mime-mapping>, debe contener <extension> y <mime-type>
- Vida máxima de las sesiones:
  - <session-config> indicando el número de minutos
- Página de error/excepción:
  - <error-page> donde hay que indicar una URL con la página de error, un código de error (404, p.ej.) o una excepción Java (IOException).

Descriptor Despliegue: Ejemplo

```
<web-app>
    <context-param>
            <param-name>log</param-name>
            <param-value>/log</param-value>
    </context-param>
    <context-param>
            <param-name>debug</param-name>
            <param-value>1</param-value>
    </context-param>
    <servlet>
            <servlet-name>controlador</servlet-name>
            <servlet-class>com.tienda.catalogo.ServletCatalogo</servlet-class>
            <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet>
            <servlet-name>cursos</servlet-name>
            <jsp-file>/jsp/cursos.jsp</jsp-file>
            <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
            <servlet-name>controlador</servlet-name>
            <url-pattern>/catalogo</url-pattern>
    </servlet-mapping>
    <error-page>
            <error-code>404</ error-code >
            <location>/error404.html</location>
    </ error-page>
</web-app>
```

Práctica

- Práctica 6.4
  - Despliegue de Servlets.



La clase HttpServlet

- Implementa la interfaz Servlet para dar soporte al protocolo HTTP.
- Amplía GenericServlet...
  - Redefine y sobrecarga el método service para dar soporte HTTP a la petición y a la respuesta:

    ```
    public void service(HttpServletRequest req, HttpServletResponse res)
       throws IOException, ServletException;
    ```

  - Proporciona métodos para dar soporte a los métodos de HTTP :

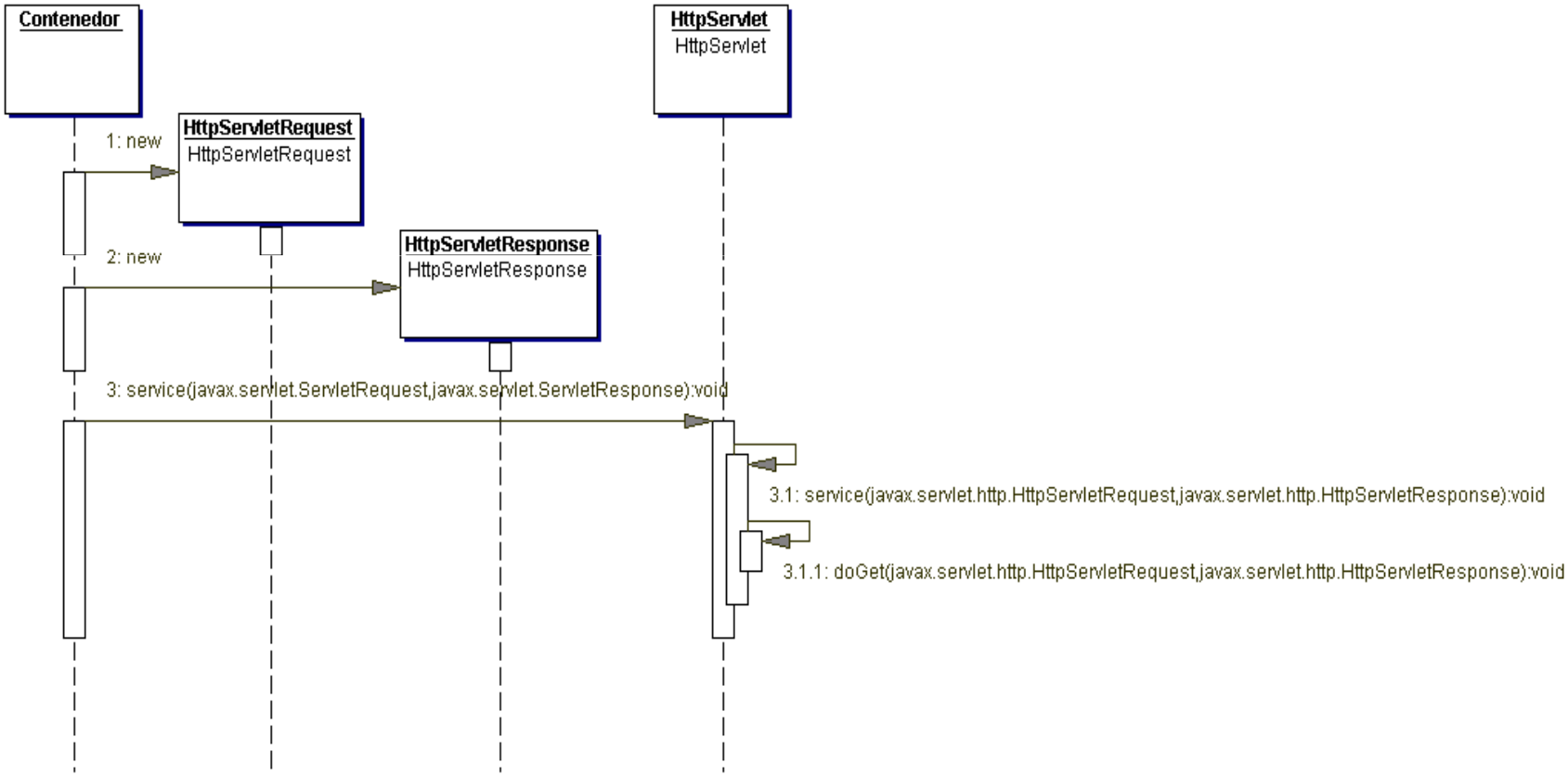
    ```
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
       throws IOException, ServletException;
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
       throws IOException, ServletException;
    ```

  - La implementación por defecto es devolver un error “Method Not Allowed”.
  - Resto de método: doHead, doOptions, doDelete, doTrace y doPut.

Métodos de soporte HTTP

- Nuevo método HttpService::service()
  - Es invocado por el contenedor, realiza los castings oportunos y delega la petición.
  - Comprueba el método HTTP de la petición y delega en el método correspondiente (doGet, doPost, etc…).
- Redefinición de service()
  - Se da soporte a todos los métodos HTTP,
  - Pero se rompe el flujo de llamadas y la petición no llegará a los métodos doXXX.
  - Por tanto hay que tener muy claro lo que se está haciendo.
- Los métodos doOptions y doTrace tienen una implementación por defecto que realiza lo que se espera de los métodos OPTIONS y TRACE.

Secuencia de petición a HttpServlet

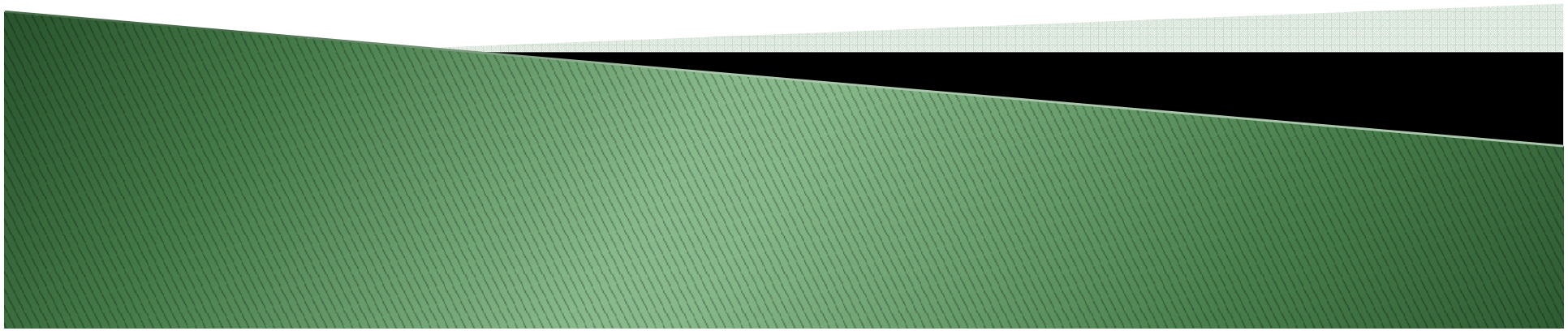


Ejemplo 3: HTTPServlet

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

public class EjemploHttpServlet extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {
     res.setContentType("text/html");
     PrintWriter out = res.getWriter();
     out.println("<html><head><title>EjemploHttpServlet</title></head>");
     out.println("<body><h1>Mensaje desde el servlet HTTP de
     ejemplo</h1></body></html>");
     out.close();
  }

  public void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException {
    doGet(req, res);
  }

  public String getServletInfo() {return "Servlet de Ejemplo usando HttpServlet";}
}
```

Servlets

Peticiones y Respuestas



Conceptos básicos

- Objetos básicos que se manejan en cualquier servlet.

Concepto	Clase General	Clase HTTP
Petición	javax.servlet.ServletRequest	javax.servlet.http.HttpServletRequest
Respuesta	javax.servlet.ServletResponse	javax.servlet.http.HttpServletResponse
Configuración	javax.servlet.ServletConfig	javax.servlet.ServletConfig
Contexto	javax.servlet.ServletContext	javax.servlet.ServletContext

- El contexto (contenedor) y la configuración del servlet son objetos independientes del protocolo.

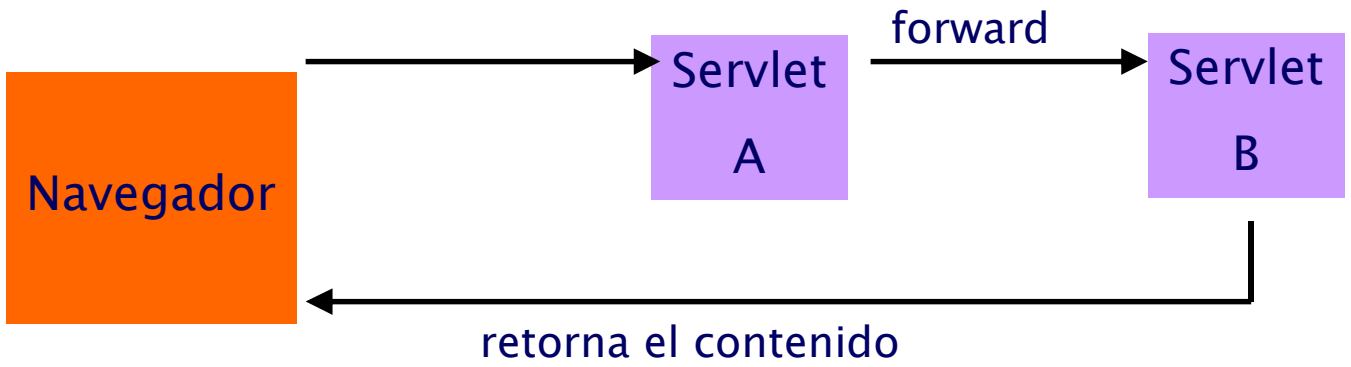
Peticiones

- Encapsuladas en la interfaz javax.servlet.ServletRequest
- Encapsula información sobre:
  - Parámetros de la petición
  - Atributos
  - Internacionalización
  - El cuerpo de la petición
  - El protocolo de la petición
  - El servidor
  - El cliente
  - Redirección de la petición (Dispatchers)
- Para dar soporte a protocolos específicos habrá que crear subinterfaces.
- El proveedor del contenedor implementa estas interfaces.

Peticiones: Parámetros

- Son pares clave-valor
- Se reciben desde el cliente en forma de cadenas de texto
- Proporcionan información para generar el contenido de la respuesta
- Métodos de manejo de parámetros de javax.servlet.ServletRequest:
  - public String **getParameter**(String name) ➔ Devuelve el valor de un parámetro en función de su clave
  - public Enumeration **getParameterNames**() ➔ Devuelve todos los nombres de los parámetros
  - public String[] **getParameterValues**(String name) ➔ Devuelve todos los valores de un parámetro compuesto.
  - public Map **getParameterMap**() ➔ Devuelve un mapa con los pares clave/valor.

Peticiones: Redirección de la petición



Peticiones: Atributos

- Una petición puede pasar por varios servlets y JSP
- Los atributos son pares clave-objeto que pueden ser recuperados por cualquier servlet/JSP durante la vida de una petición
- Diferencia parámetros/atributos
  - Los parámetros llegan desde el cliente
  - Los atributos se añaden durante la gestión de la petición (dentro del contenedor)
- Métodos de gestión de atributos de javax.servlet.ServletRequest:
  - public Object getAttribute(String name) ➔ Devuelve el valor de un atributo en función de su clave
  - public Enumeration getAttributeNames() ➔ Devuelve todos los nombres de los atributos
  - public void setAttribute(String name, Object o) ➔ Añade un atributo a la petición
  - public void removeAttribute(String name) ➔ Elimina un atributo de la petición

Peticiones: El cuerpo de la petición

- Algunas peticiones incluyen un cuerpo (v.g. PUT y POST en HTTP)
- Métodos para gestión del cuerpo de las peticiones de javax.servlet.ServletRequest:
  - public String getCharacterEncoding() ➔ Devuelve el juego de caracteres del cuerpo de la petición
  - public void setCharacterEncoding(String env) throws UnsupportedEncodingException ➔ Sobreescribe el juego de caracteres del cuerpo de la petición
  - public int getContentLength() ➔ Devuelve la longitud total del cuerpo de la petición
  - public String getContentType() ➔ Devuelve el tipo MIME del cuerpo de la petición
  - public ServletInputStream getInputStream() throws IOException ➔ Devuelve un InputStream para leer el cuerpo de la petición (binario)
  - public BufferedReader getReader() throws IOException ➔ Devuelve un Reader para leer el cuerpo de la petición (texto)

Peticiones: El servidor

- Métodos para recuperar información sobre el servidor que atiende la petición:
  - public String getServerName() ➔ Devuelve el nombre del servidor que ha recibido la petición.
  - public int getServerPort() ➔ Devuelve el puerto por el que el servidor ha recibido la petición.

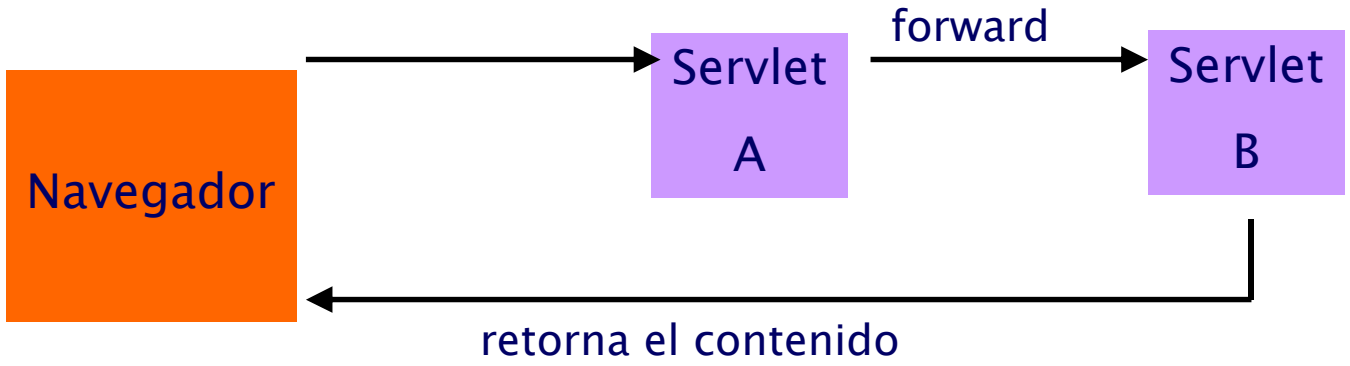
Peticiones: El cliente

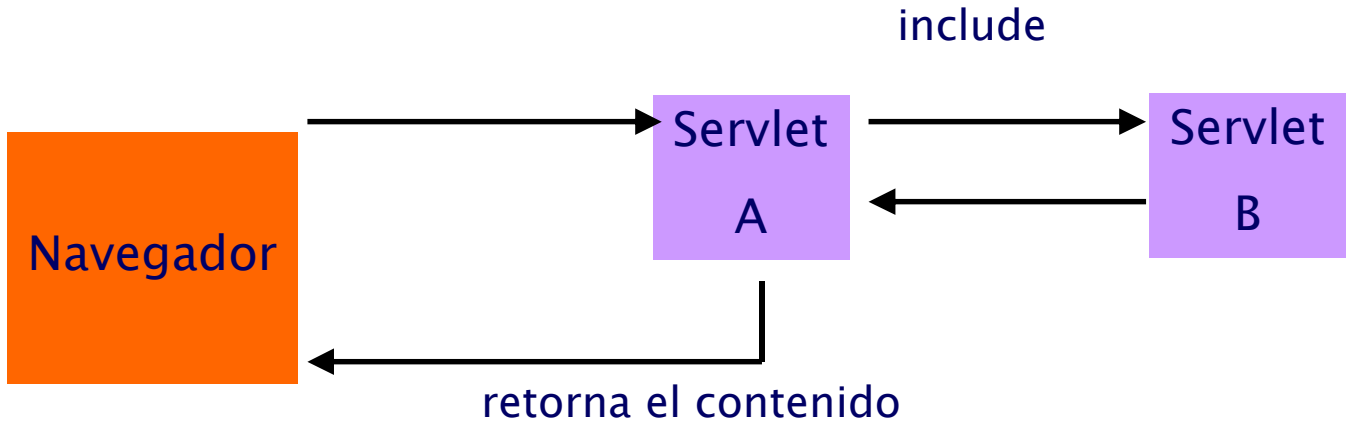
- Métodos para recuperar información sobre el cliente que hizo la petición:
  - public String getRemoteHost() ➔ Devuelve el nombre de la máquina cliente que ha realizado la petición (si puede resolverlo, si no null)
  - public String getRemoteAddr () ➔ Devuelve la dirección del host que realizó la petición

Peticiones: Redirección de la petición (Dispatchers)

- Una petición puede pasar por diferentes servlets y/o JSP's
- Cada Servlet/JSP es responsable de la gestión de una parte de la petición
- Soporte a la delegación/composición (forward/include)
  - Interfaz RequestDispatcher
  - Es posible recuperar un RequestDispatcher tanto desde javax.servlet.ServletRequest como desde javax.servlet.ServletContext
  - Desde la interfaz ServletRequest mediante el método:
    public RequestDispatcher getRequestDispatcher(String path)

Peticiones: Redirección de la petición (Dispatchers)





Peticiones: Redirección de la petición (Dispatchers)

- getServletContext().
  getRequestDispatcher("/showBalance.jsp").
  forward(req,res);

- getServletContext().
  getRequestDispatcher("/navigation_bar.html").
  include(req,res);

Respuestas

- Los servlets generan la respuesta a partir de un objeto que implementa la interfaz javax.servlet.ServletResponse que encapsula un OutputStream, sobre el que escribe el servlet.
- Los objetos ServletResponse son instanciados por el contenedor y pasados al servlet.
- Para dar soporte a protocolos específicos se debe derivar de la interfaz ServletResponse (v.g. javax.servlet.http.HttpServletResponse) .
- El OutputStream del que depende esta interfaz se implementa usando un mecanismo de buffer intermedio. La interfaz proporciona métodos para controlar este buffer.

Respuestas: El cuerpo de la respuesta

- El cuerpo de la respuesta se escribe por medio de un stream que se apoya en un buffer
  - Para contenido dinámico: javax.servlet.ServletOutputStream
  - Para texto plano: PrintWriter
- Métodos para manejar y recuperar información del flujo de escritura:
  - public ServletOutputStream getOutputStream() throws IOException
  - public PrintWriter getWriter() throws IOException

    Estos dos métodos son mutuamente excluyentes (si se ha invocado alguno de ellos, invocar el otro lanzará una IllegalStateException)
  - public void reset() ➔ Vacía el stream (cuerpo y cabeceras)
  - public boolean isCommitted() ➔ Devuelve true si el buffer ya se ha volcado (se han escrito las cabeceras y el cuerpo)

Respuestas: El cuerpo de la respuesta

- Métodos de manejo y establecimiento del buffer usado por el flujo de respuesta:
  - public void setBufferSize(int size) ➔ Establece el tamaño del buffer (el tamaño por defecto depende de la implementación).
  - public int getBufferSize() ➔ Devuelve el tamaño del buffer (o cero si no se está usando buffer).
  - public void flushBuffer() throws IOException ➔ Provoca el volcado de la respuesta (cuerpo y cabeceras). Cualquier llamada a isCommited a partir de la invocación de este método devolverá true
  - public void resetBuffer() ➔ Vacía el buffer (el cuerpo, pero no las cabeceras).

Respuestas: Cabeceras

- La interfaz javax.servlet.ServletResponse posee métodos para dar información sobre la respuesta que se va a generar.
- Esta información se copiará en la respuesta de manera dependiente del protocolo (habitualmente en forma de cabeceras).

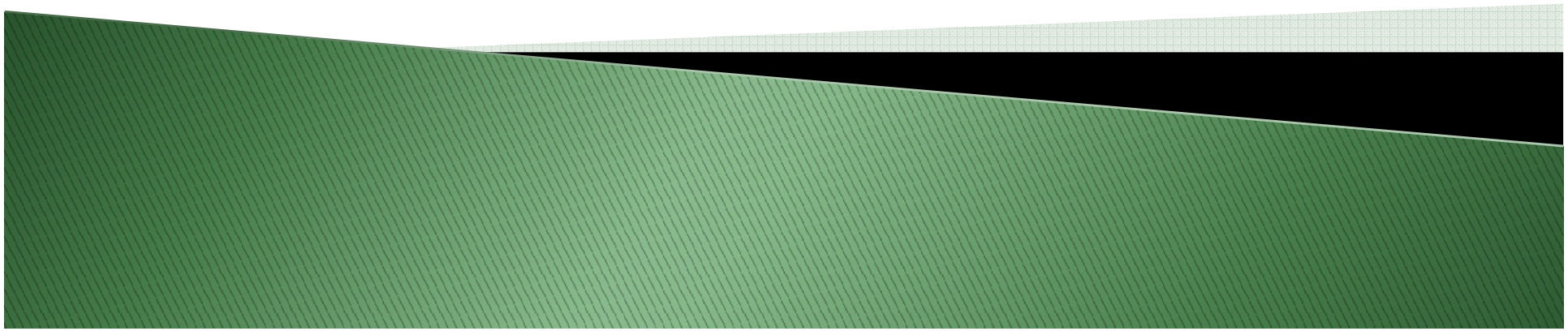
- Los métodos para dar soporte a esta característica son:

  - public void setContentLength(int len) ➔ Establece información sobre la longitud del cuerpo de la respuesta.
  - public void setContentType(String type) ➔ Establece el tipo MIME de la respuesta.
  - public void setLocale(Locale loc) ➔ Establece la localización de la respuesta.
  - public Locale getLocale() ➔ Devuelve el Locale establecido por el método anterior.
  - public String getCharacterEncoding() ➔ Devuelve el nombre del juego de caracteres usado para la generación del cuerpo de la respuesta (establecido por medio del Locale, del tipo MIME o ISO-8859-1 por defecto).

  - Todos estos métodos set se deben invocar antes de invocar flushBuffer()), y si la respuesta que se va a generar es texto plano, antes de getWriter().

Servlets

Peticiones y Respuestas HTTP



Peticiones HTTP: Cabeceras

- El protocolo HTTP define cabeceras para manejar meta información
- Métodos de la interfaz HttpServletRequest:
  - public long getDateHeader(String name)
  - public String getHeader(String name)
  - public Enumeration getHeaders(String name)
  - public Enumeration getHeaderNames()
  - public int getIntHeader(String name)
- Las cookies son un tipo especial de cabeceras que se establecen en el servidor y el navegador envía en todas las peticiones siguientes:
  - public Cookie[] getCookies()

Peticiones HTTP: Información sobre URLs

- Información sobre la URL que genera cada petición
- Métodos de la interfaz HttpServletRequest:
  - public String getPathInfo()
  - public String getContextPath()
  - public String getQueryString()
  - public String getRequestURI()
  - public String getPathTranslated()
  - public StringBuffer getRequestURL()
  - public String getServletPath()

Peticiones HTTP: Sesiones

- Aunque se tratará más tarde, aquí se presentan los métodos de gestión de la sesión de HttpServletRequest:
  - public HttpSession getSession(boolean create)
  - public HttpSession getSession()
  - public boolean isRequestedSessionIdValid()
  - public boolean isRequestedSessionIdFromCookie()
  - public boolean isRequestedSessionIdFromURL()
  - public String getRequestedSessionId()

Respuestas HTTP: Cabeceras HTTP

- Métodos de HttpServletResponse para para añadir, reemplazar y comprobar las cabeceras que el contenedor enviará al cliente en la respuesta:
  - public boolean containsHeader(String name)
  - public void setDateHeader(String name, long date)
  - public void addDateHeader(String name, long date)
  - public void setHeader(String name, String value)
  - public void addHeader(String name, String value)
  - public void setIntHeader(String name, int value)
  - public void addIntHeader(String name, int value)

  Los métodos *add* añaden si la clave no está ya presente
  Los métodos *set* añaden y, si la clave ya está presente, reemplazan

- Para añadir cookies a una respuesta:
  - public void addCookie(Cookie cookie) ➔ complementario a HttpServletRequest.getCookies()

Respuestas HTTP: Códigos de Estado HTTP

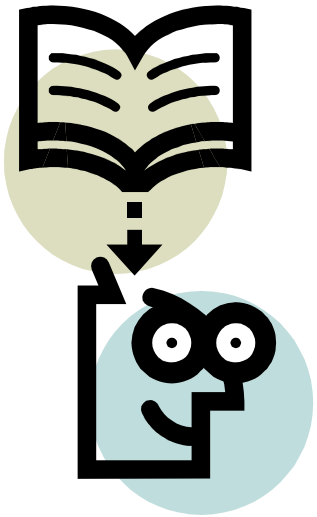
- El contenedor Web proporciona información sobre el estado de la petición al navegador mediante códigos de estado.
  - Códigos de error.
  - Códigos de redirección.
  - Códigos de información general.

- Métodos de HttpServletResponse:

  - public void sendError(int sc, String msg) throws IOException
  - public void sendError(int sc) throws IOException
  - public void sendRedirect(String location) throws IOException
  - public void setStatus(int sc)

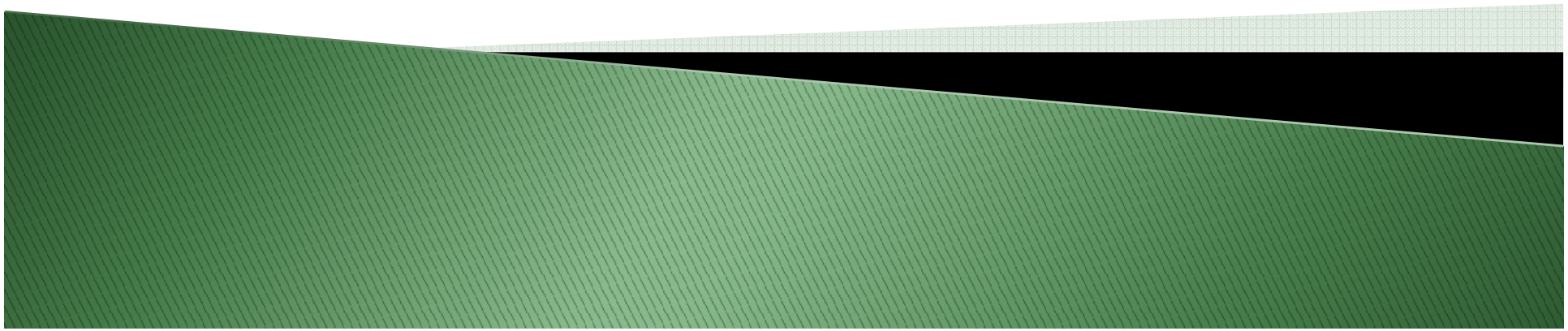
Práctica

- Práctica 6.5
  - Servlet con acceso a base de datos con JDBC



Servlets

Contexto



Contexto

- La interfaz **javax.servlet.ServletContext** es el punto de comunicación entre los servlets y el contenedor
  - Encapsula servicios para los componentes web
  - Define un ámbito para las variables (junto con la sesión y la petición)
  - Es el punto de contacto de los servlets con el sistema de archivos (sólo el contenedor conoce el mapeo de directorios virtuales a directorios físicos)
- Existe un objeto ServletContext por aplicación y máquina virtual. Todos los atributos que se definan en el contexto serán accesibles para el resto de componentes (Servlets y JSP) de la aplicación
- En aplicaciones distribuidas (en clusters), el contexto no se duplica entre nodos del cluster (Hay ciertos servidores que sí lo hacen, pero la especificación no lo requiere)
- Métodos para recuperar recursos y tipo MIME:
  - public String getRealPath(String path)
  - public InputStream getResourceAsStream(String path)
  - public URL getResource(String path) throws MalformedURLException
  - public Set getResourcePaths(String path)
  - public String getMimeType(String file)

Contexto

- El contexto representa el ámbito de mayor visibilidad dentro de un contenedor.
- Cualquier objeto que se guarde en el contexto será accesible para cualquier componente de la aplicación (sólo hay una instancia de ServletContext por aplicación).
- Métodos para ello:
  - public Object getAttribute(String name)
  - public Enumeration getAttributeNames()
  - public void setAttribute(String name, Object object)
  - public void removeAttribute(String name)

Contexto

- Acceso a los mecanismos de registro de sucesos (log) del contenedor:
  - public void log(String msg)
  - public void log(String msg, Throwable throwable)
- Comunicación entre aplicaciones distintas accediendo a otros contextos:
  - public ServletContext getContext(String uripath)

Contexto

- Dos formas de recuperar dispatchers del contexto:
  - Por medio de la URL del recurso destino y,
  - Por medio del nombre del recurso (declarado en el descriptor de despliegue).

    - public RequestDispatcher getRequestDispatcher(String path)
    - public RequestDispatcher getNamedDispatcher(String name)

- Para recuperar el nombre de la aplicación (campo *description* del descriptor de despliegue)

  - public String getServletContextName()

ServletConfig

- La interfaz javax.servlet.ServletConfig representa la configuración de un servlet individual.
- Es de donde los servlets recuperan el contexto.

- La clase GenericServlet implementa esta interfaz delegando las llamadas al objeto ServletConfig que recibe en init.

  - public String getServletName()
  - public ServletContext getServletContext()
  - public String getInitParameter(String name)
  - public Enumeration getInitParameterNames()

- Si se redefine init(ServletConfig), hay que incluir SIEMPRE una llamada a super.init(config).

Ejemplo (sin ServletConfig)

```
public class InfoServlet extends HttpServlet {
  private String url;
  private String dirInicio;
  private Connection con;
  public void init(ServletConfig config) throws ServletException {
    super.init(config);

    String sURL="jdbc:mysql://localhost/BDJugadores";
    try {
       Class.forName("com.mysql.jdbc.Driver").newInstance();
       con = DriverManager.getConnection(sURL);
    }
    catch (ClassNotFoundException cnfe) {
        throw new UnavailableException("No se encuentra la clase " + driver);
    }
```

Ejemplo (sin ServletConfig)

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws
                                                                    ServletException, IOException {
   //Recuperar parámetros de la petición
   String nombre = request.getParameter("nombre");
   String email = request.getParameter("email");
   String curso = request.getParameter("curso");

   //Insertar datos en BD
   try {

     Statement stmt = con.createStatement();
     stmt.execute("insert into peticiones values (" + nombre + "', '" + email + "', '" + curso + "')");
     stmt.close();
     con.close();
   } catch (SQLException sqle) {
     throw new ServletException("Error de la base de datos ", sqle);
   }

   //Devolver respuesta
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   String respuesta = generarRespuesta(nombre, email, curso);
   out.println(respuesta);
   out.close();
 }
```

Ejemplo (sin ServletConfig)

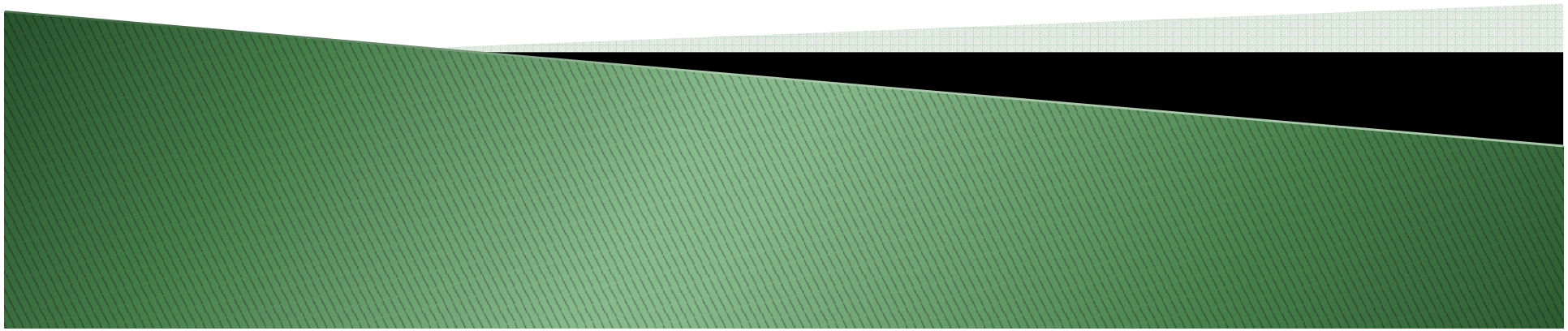
```
    private String generarRespuesta(String nombre, String email, String curso) {
        StringBuffer sb = new StringBuffer();

        sb.append("<html><head><title>Petición Procesada</title></head>");
        sb.append("<body bgcolor='black' text='Silver'><div align='center'>");
        sb.append("<h1>Su petici&oacute;n ha sido recibida y procesada</h1>");
        sb.append("<table cellpadding='2' border='2' align='center'>");
        sb.append("<tr><td>Nombre</td><td>" + nombre + "</td></tr>");
        sb.append("<tr><td>e-mail</td><td>" + email + "</td></tr>");
        sb.append("<tr><td>Curso</td><td>" + curso + "</td></tr>");
        sb.append("</table>");
        sb.append("<p><H1>Recibir&aacute; un e-mail lo antes posible</H1>");
        sb.append("</div></body></html>");

        return sb.toString();
    }

} //InfoServlet
```

Ejemplo (con ServletConfig)

```
public class InfoServlet extends HttpServlet {
  private String url;
  private String dirInicio;
  private Connection con;
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    //Recuperar parámetros de inicio (acceso a BD)
    String driver = config.getInitParameter("driver");
    url = config.getInitParameter("url");
    //Comprobación de los valores
    if (driver == null || url == null) {
      throw new UnavailableException("No se especificaron los parámetros de inicio necesarios");
    }
    //Cargar el driver
    try {  Class.forName(driver);
           con = DriverManager.getConnection(url);
    } catch (ClassNotFoundException cnfe) {
           throw new UnavailableException("No se encuentra la clase " + driver);
    }
  }
  public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException { ... }
  private String generarRespuesta(String nombre, String email, String curso) {...}
} //InfoServlet
```

Servlets

Sesiones



Sesiones

- HTTP es un protocolo “sin estado”, es decir, se envía una petición, el servidor contesta y se olvida.
- No se puede mantener una sesión (guardar estado) entre peticiones, ya que el servidor no tiene constancia de quién está realizando la petición siguiente a una dada.
- En casi cualquier aplicación web es necesario mantener una sesión por usuario (p.e. un carrito de la compra).

Sesiones

- Mecanismos alternativos:
  - URL Encoding (Codificación de URL's)
  - Campos Ocultos
  - HTTPS
  - Cookies
  - Objetos HttpSession

- Todos basados en incluir un identificador en todas las comunicaciones entre cliente y servidor.

- El más usado actualmente son las cookies.

Sesiones: URL Encoding (rewriting)

- Añadir un identificador de sesión a todos los enlaces que se devuelven al cliente
- No depende de la configuración del navegador, pero es tedioso y propenso a errores (hay que codificar todas las URLs que se devuelven al cliente)
- El contenedor web proporciona el servicio de generación de identificadores y codificación de URL:
  - encodeURL(String url) de HttpServletResponse

Sesiones: Campos Ocultos

- Mediante el uso de formularios HTML y campos ocultos se puede mantener la sesión por medio de programación.

  <input type="hidden" name="nombre" value="valor">

- No existe ningún tipo de ayuda por parte del contenedor para implementar este mecanismo; el programador debe generar los campos ocultos e incluirlos en el cuerpo de la respuesta y leerlos de la siguiente petición.

Sesiones: HTTPS

- HTTPS es una implementación de HTTP sobre SSL (Secure Socket Layer).
- SSL es una tecnología de cifrado que funciona sobre TCP/IP y por debajo de los protocolos de aplicación (p.e. HTTP).
- La encriptación de la comunicación se basa en un par de claves simétricas (claves de sesión) que se generan al establecer la conexión.
- Se pueden usar estas claves para mantener la sesión, siempre y cuando se requiera autenticación del cliente por requisitos de la aplicación. No tiene sentido usar HTTPS sólo para mantener sesión.

Sesiones: Cookies

- Una cookie es un par clave/valor aumentado para incluir más información
- Formato:
  - Respuesta del servidor:
    - Set-cookie: CLAVE=VALOR; Comment=COMENTARIO; Domain=DOMINIO; Max-age=SEGUNDOS; Path=PATH; secure; Version=version
  - Cliente:
    - Cookie: CLAVE=VALOR
- El identificador se envía y recibe en forma de cabecera HTTP y son el contenedor y el navegador los encargados de mantener la sesión sin ayuda por parte del programador
- No se mezcla el seguimiento de sesión con el código HTML
- Depende de la configuración del navegador (puede estar configurado para rechazar las cookies)

Sesiones: Cookies

- Ejemplo:

  Set-cookie: id=12; Max-age=1800; Domain="java.sun.com"; Path="/products"

  - El servidor crea una cookie con clave "id" y valor "12" con una vida de media hora para el dominio "java.sun.com" y para el contexto "/products"
  - El navegador determinará si se debe enviar la cookie mediante los atributos Domain y Path, y, al cabo de media hora el navegador debe descartar la cookie
  - El cliente devolverá en las cabeceras de las peticiones subsiguientes:

    Cookie: id=12

Sesiones: Objetos HttpSession

- Interfaz javax.servlet.http.HttpSession = abstracción del concepto de sesión
- Se usa como contenedor de atributos que representan el "estado" del cliente en el servidor
- Las sesiones se crean/recuperan de la interfaz HttpServletRequest, haciendo transparente al programador toda la creación y recuperación de sesiones.
- HttpSession utiliza internamente cookies por defecto. Si el mecanismo no funciona, el programador debe actuar. Por ejemplo, con URL encoding.
- Métodos para recuperar una sesión:
  - public HttpSession getSession(boolean create)
    - Devuelve la sesión correspondiente al cliente y si no existe:
      - Si create es true, la crea.
      - Si create es false, devuelve null.
  - public HttpSession getSession()
    - Igual a getSession(true)

Sesiones: Objetos HttpSession

- Básicamente la sesión es un **contenedor de atributos** cuyos métodos de gestión son:
  - public Object getAttribute(String name)
  - public Enumeration getAttributeNames()
  - public void setAttribute(String name, Object value)
  - public void removeAttribute(String name)

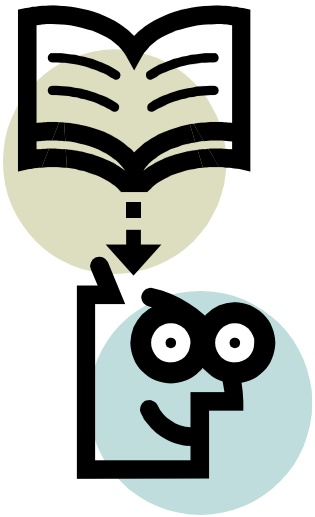
- Los métodos para la gestión de la vida de la sesión son:
  - public void invalidate() ➔ Invalida (destruye) la sesión, las siguientes peticiones de sesión, devolverán una nueva
  - public boolean isNew() ➔ Devuelve true si la sesión se acaba de crear y todavía no se ha enviado la cookie al cliente
  - public void setMaxInactiveInterval(int interval) ➔ Establece el tiempo de inactividad máximo, si el cliente está inactivo durante el número de segundos indicado la sesión se invalida. Por defecto su valor es el establecido en el DD (<session-config>), y si no se ha especificado depende del servidor

Sesiones: Ejemplo

```
public class ContadorServlet extends HttpServlet {
  private static final String NUM_ACCESOS = "numAccesos";

  public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {
    HttpSession sesion = req.getSession(true);
    if (sesion.isNew()) {
      sesion.setAttribute(NUM_ACCESOS, new Integer(1));
      sesion.setMaxInactiveInterval(600);
    } else {
      int i = ((Integer)(sesion.getAttribute(NUM_ACCESOS))).intValue();
      sesion.setAttribute(NUM_ACCESOS, new Integer(++i));
    }

    Date fCreacion = new Date(sesion.getCreationTime());

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<html><head><title>Contador</title></head>");
    out.println("<body><h1>Accessos desde " + fCreacion + ": ");
    out.println(sesion.getAttribute(NUM_ACCESOS));
    out.close();
  }
}
```

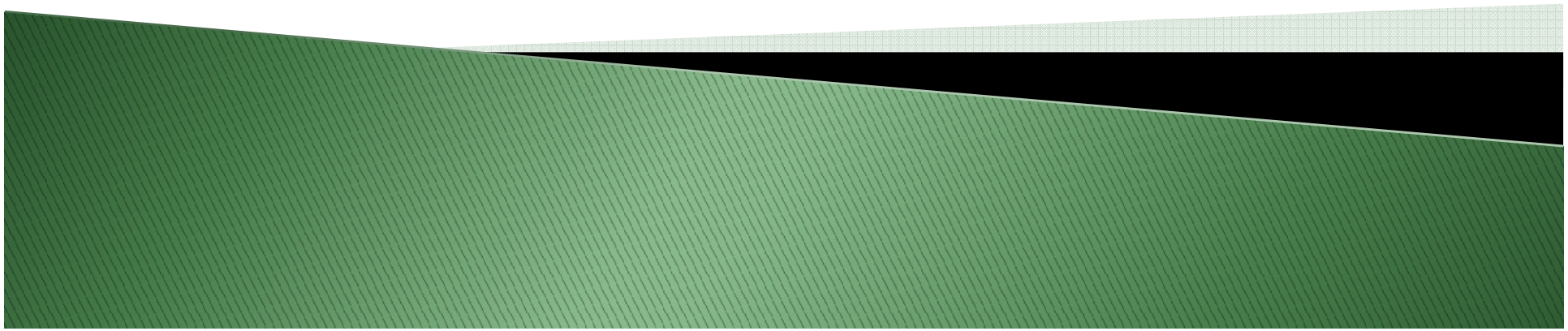
Práctica

- Práctica 6.6
  - Sesiones con Servlets.



Servlets

Filtros



Filtros

- El programador puede interceptar el flujo de las peticiones en el contenedor de una manera estándar
- Los filtros capturan la petición y pueden cambiar el flujo de ésta (decidir a qué servlet dirigirla, rechazarla, encadenar una serie de servlets para distribuir las tareas, etc…)
- Aplicaciones de los filtros:
  - Autenticación y autorización, tanto en el contenedor como en recursos externos (sistemas legados)
  - Sistemas de registro de eventos (log) y auditoría
  - Implementación de cachés de contenido generado
  - Control parametrizado de acceso (p.e. dependiendo de la hora/fecha)
  - Transformaciones de la respuesta:
    - Cifrado
    - Compresión
    - Transformaciones XSLT

Filtros: API

- Filter: Representa un filtro. Tres métodos definen el ciclo de vida:
  - public void init(FilterConfig filterConfig)
  - public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
  - public void destroy()
- FilterConfig
  - public String getFilterName()
  - public String getInitParameter(String parameterName)
  - public java.util.Enumeration getInitParameterNames()
  - public ServletContext getServletContext()
- FilterChain
  - public void doFilter(HttpServletRequest request, HttpServletResponse response)

Filtros: Interfaz Filter

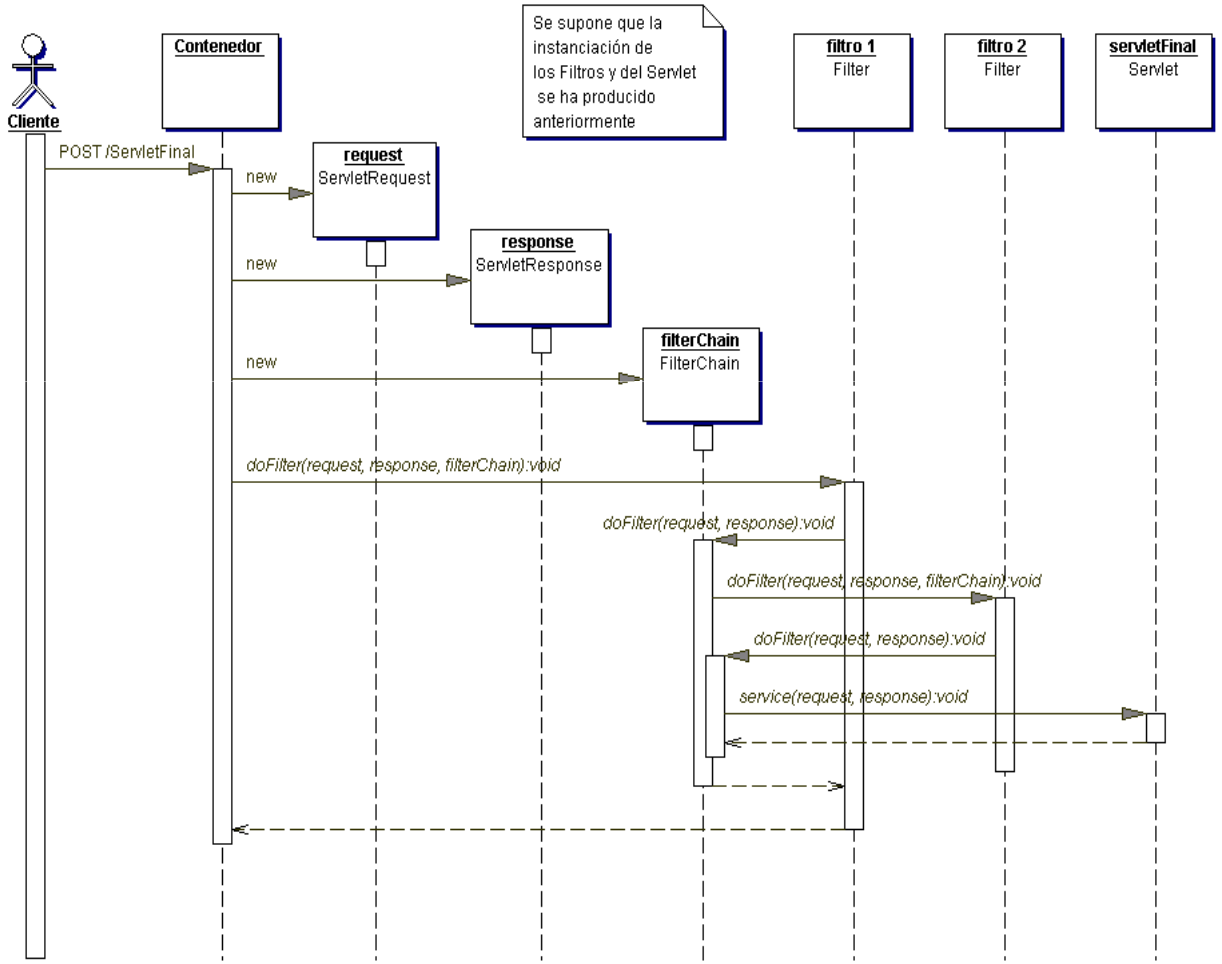
- init: Inicialización del filtro
  - Recibe un argumento de configuración, de manera similar a la interfaz Servlet:

    ```
    public void init(FilterConfig fg) {ServletContext ctx = fg.getServletContext();}
    ```

- doFilter: Es equivalente al método doService de Servlet
  - HttpServletRequest request: Posibilidad de añadir y modificar atributos.
  - HttpServletResponse response: Posibilidad de interceptar el objeto PrintWriter.
  - FilterChain chain: Pasar el control a otro filtro. Hay que hacerlo siempre. Si no hay más en la cadena, se dará control al siguiente recurso (servlet, JSP).

- destroy: Finalización y limpieza.

Filtros: ciclo de vida



Ejemplo de filtros: filtrado básico

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

public class BasicFilter implements Filter {
    private FilterConfig _filterConfig;
    private ServletContext _ctx;

    public void init(FilterConfig fg) {
            _filterConfig = filterConfig;
            _ctx = filterConfig .getServletContext();
    }

    public void doFilter(ServletRequest rq, ServletResponse rs, FilterChain fc)
       throws ServletException, IOException {
            ctx.log("Conexión desde " + rq.getRemoteAddr());
            request.setAttribute("Conexion desde", request.getRemoteAddr());
            fc.doFilter(rq, rs);
    }

    public void destroy() {}
}
```

Ejemplo de filtros: DD

```
<web-app>
    <servlet>
        <servlet-name>Filter Demo</servlet-name>
        <servlet-class>ServletDemo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Filter Demo</servlet-name>
        <url-pattern>/servlet/ServletDemo</url-pattern>
    </servlet-mapping>
    <filter>
        <filter-name>Basic Filter</filter-name>
        <filter-class>BasicFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>Basic Filter</filter-name>
        <servlet-name>Filter Demo</servlet-name>
    </filter-mapping>
</web-app>
```

Ejemplo de filtros: DD para todos los recursos

```
<web-app>
    <filter>
        <filter-name>Logging Filter</filter-name>
        <filter-class>BasicFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>Logging Filter</filter-name>
        <url-pattern>/*</ url-pattern >
    </filter-mapping>
</web-app>
```

Ejemplo de filtros: quitar espacios

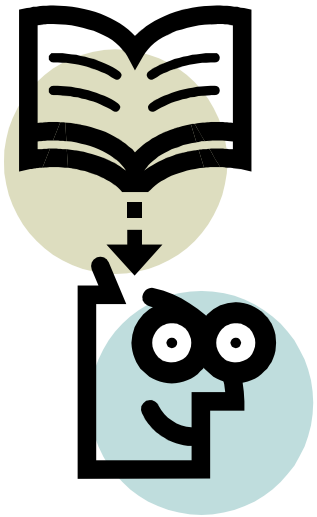
```
import java.io.*;
import javax.servlet.*;
import java.util.Enumeration;

public class TrimFilter implements Filter {

    public void init(FilterConfig fg) {}

    public void doFilter(ServletRequest rq, ServletResponse rs, FilterChain fc)
       throws ServletException, IOException {
            Enumeration enum = request.getParameterNames();
            while (enum.hasMoreElements()) {
                        String parameterName = (String) enum.nextElement();
                        String parameterValue = request.getParameter(parameterName);
                        request.setAttribute(parameterName, parameterValue.trim());
            }
            fc.doFilter(rq, rs);
    }

    public void destroy() {}
}
```

Ejemplo de filtros: filtrando la salida

```
public class ResponseFilter implements Filter {

    public void init(FilterConfig fg) {}

    public void doFilter(ServletRequest rq, ServletResponse rs, FilterChain chain)
        throws ServletException, IOException {
            PrintWriter out = rs.getWriter();

            // this is added to the beginning of the PrintWriter
            out.println("<html><body><center>Page header<hr />");

            chain.doFilter(rq, rs);

            // this is added to the end of the PrintWriter
            out.println("<hr /><center>Page footer</body></html>");
    }

    public void destroy() {}
}
```

Práctica

- Práctica 6.7
  - Servlets, contexto, configuración y filtros.



Práctica

- Práctica 6.8
  - Pool de conexiones con JDNI y Servlets 3.0.