

Servlets

- ① Características generales
- ② Servicios que proveen los Contenedores Web
- ③ Ciclo de vida de los Servlets: `init()`, `service()` y `destroy()`
- ④ La interface de programación

La interface `javax.servlet.Servlet`

La clase `javax.servlet.http.HttpServlet`

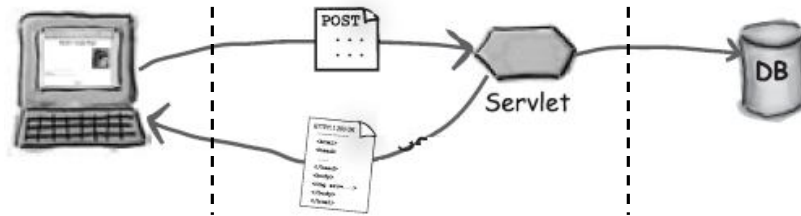
Las interfaces `HttpServletRequest` y `HttpServletResponse`

- ⑤ Ejemplos de Servlet
- ⑥ Archivos de configuración de Servlet: archivo descriptor del “deploy” (Deployment descriptor-DD): `WEB.XML`
- ⑦ Deploy de Servlet
- ⑧ Parámetros de inicialización de servlets
- ⑨ Redireccionar el requerimiento: `sendRedirect()` y delegar requerimiento y la respuesta: `forward()` e `include()`.

Servlets

Características generales

- Un **servlet** es una componente web escrita en Java que es gerenciada por un **Contenedor Web**. Procesa requerimientos y construye respuestas dinámicamente. Son ideales para realizar procesamiento en la capa del medio (*middleware*). Es la tecnología básica para la construcción de aplicaciones web JAVA.



- Los **servlets** son clases Java independientes de la plataforma, se compilan a código de bytes (*bytecodes*), se cargan dinámicamente y se ejecutan en un **Contenedor web**.
- Los servlet hacen uso de la Plataforma Java y de servicios provistos por el Contenedor Web:
 - La **Plataforma Java** provee un API robusta basada en POO para construir servlets.
 - El **Contenedor Web**, evita que el programador se ocupe de la conectividad con la red, de capturar los pedidos, de producir las respuestas, seguridad, etc. Gerencia el ciclo de vida del servlet.
- La última versión de Servlets es la 3.1, está especificada en la JSR 340 perteneciente a la JEE 7.

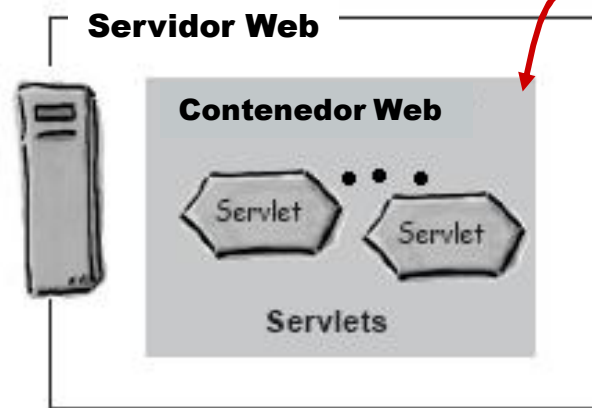
Servlets

El Contenedor Web

El Contenedor Web es responsable de:

1. la **conectividad** con la red
2. **capturar los requerimientos HTTP**, traducirlos a objetos que el servlet entienda, entregarle dichos objetos al servlet quién los usa para producir las respuestas
3. **generar una respuesta HTTP** en un formato correcto (MIME-type)
4. **gerenciar** el ciclo de vida del servlet
5. **manejar errores y proveer seguridad**

Un Contenedor web es una extensión del servidor web que provee soporte para servlets. Es parte del servidor web o del servidor de aplicaciones.



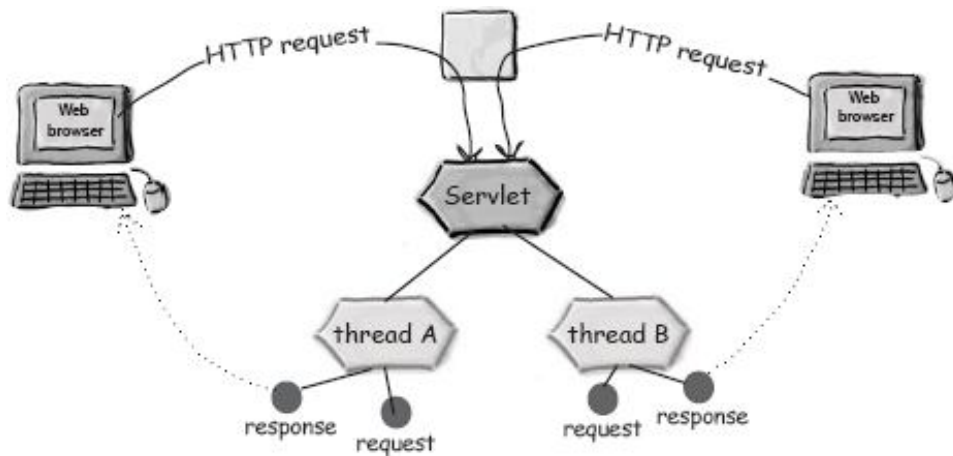
El Contenedor web interactúa con los servlets invocando métodos de gerenciamiento o métodos callback. Estos métodos definen la interface entre el Contenedor y los servlets (API de servlets).

El Contenedor Web está construido sobre la plataforma estándar de Java (JSE™), implementa la API de servlets y todos los servicios requeridos para procesar pedidos HTTP. Cada contenedor web tiene su propia implementación de la API de servlets.

Contenedor Web

Los servlet son controlados por el Contenedor

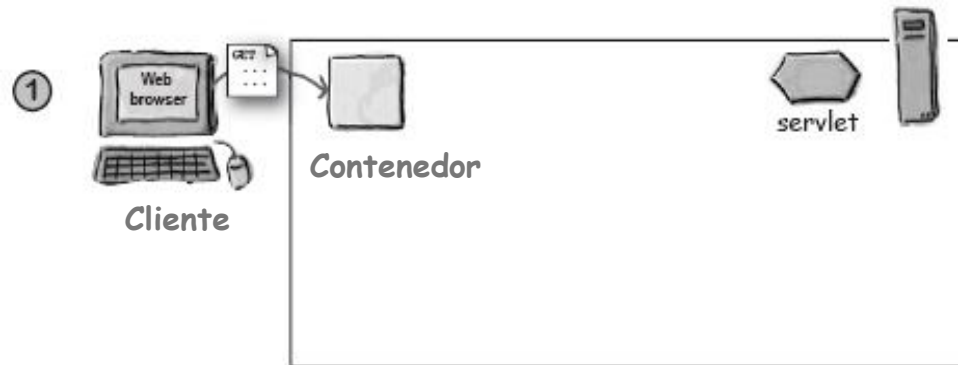
- El Contenedor Web gerencia el **ciclo de vida de cada servlet**, invocando a 3 métodos definidos en la interface **javax.servlet.Servlet: `init(..)`, `service(..)` y `destroy(..)`**
- El Contenedor Web, es responsable de la carga e instanciación de los servlets, que puede suceder en el arranque, cuando una aplicación es actualizada (recargada) o puede ser postergada hasta que el servlet es requerido por primera vez.
- El Contenedor Web, crea una única instancia de cada servlet declarado en la aplicación web.
- El Contenedor Web, maneja los requerimientos concurrentes a un mismo servlet, ejecutando el método **`service()`** concurrentemente en múltiples threads Java.



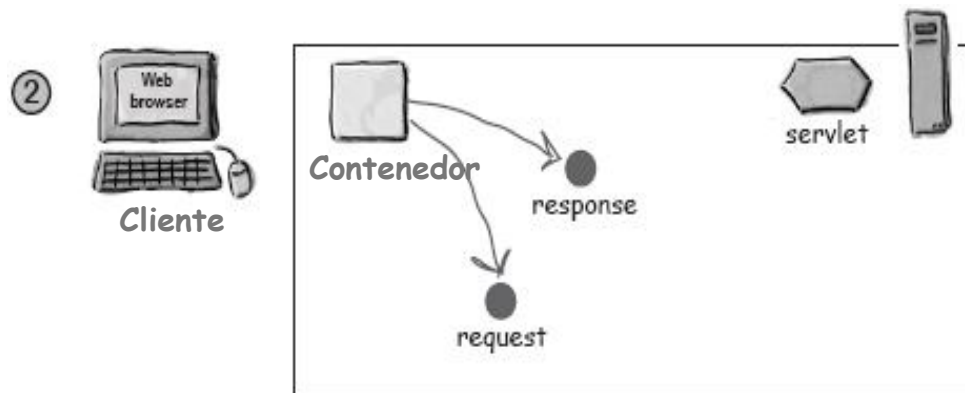
El programador de servlets, debe considerar los efectos colaterales que tiene el procesamiento concurrente y tomar las previsiones necesarias sobre los recursos compartidos (acceso a variables de instancia, variables de clase, recursos externos como archivos, etc.)

Contenedor Web

Los servlet son controlados por el Contenedor

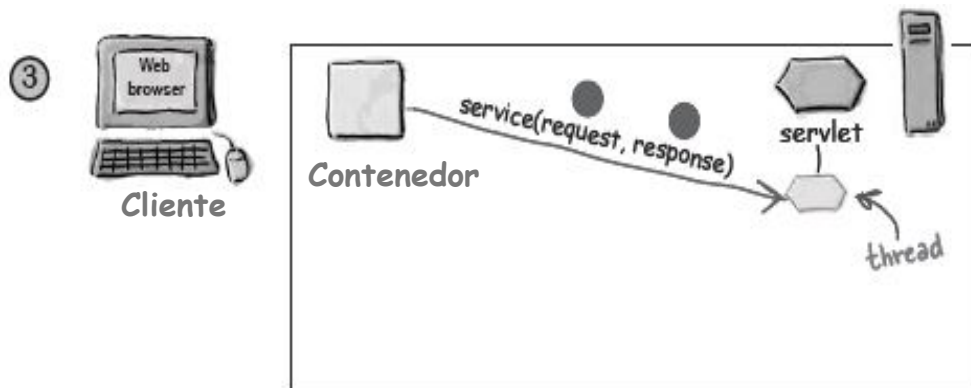


El usuario presiona sobre un link que tiene una URL a un Servlet



El Contenedor crea 2 objetos:

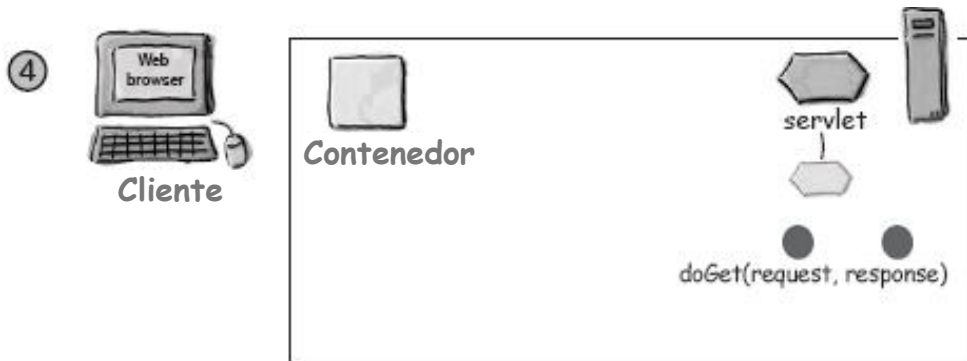
- 1) **HttpServletResponse**: representa la respuesta que se le enviará al cliente
- 2) **HttpServletRequest**: representa el requerimiento del cliente.



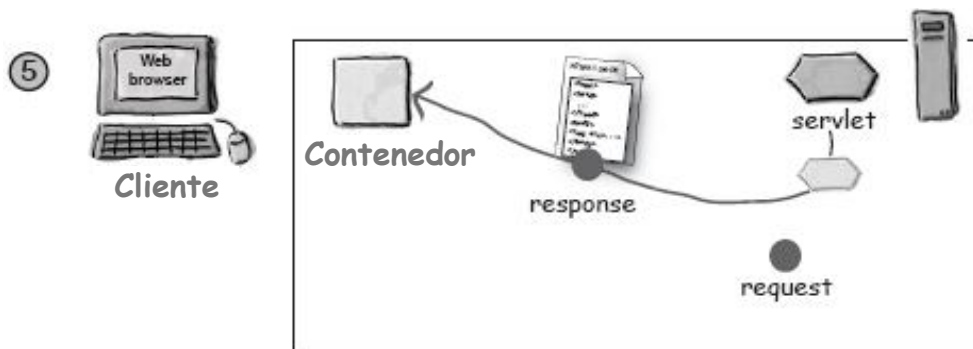
El Contenedor encuentra el servlet correcto usando la URL del requerimiento. Luego crea o aloca un **thread** para ese requerimiento e invoca al método **service()** pasándole los objetos **request** y **response** como argumento.

Contenedor Web

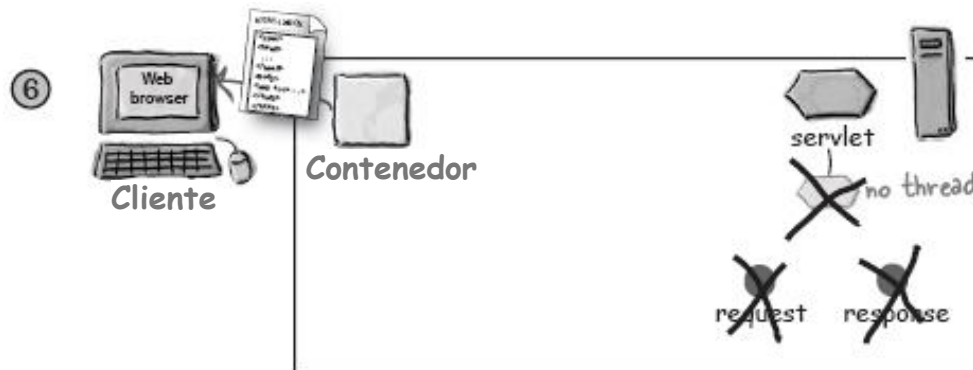
Los servlet son controlados por el Contenedor



El contenedor invoca al método **service()** del **servlet**, el cual determina que método invocar dependiendo del método HTTP (get o post) enviado por el cliente.



El servlet usa el objeto **response** para escribir la respuesta al cliente. La respuesta regresa a través del contenedor.

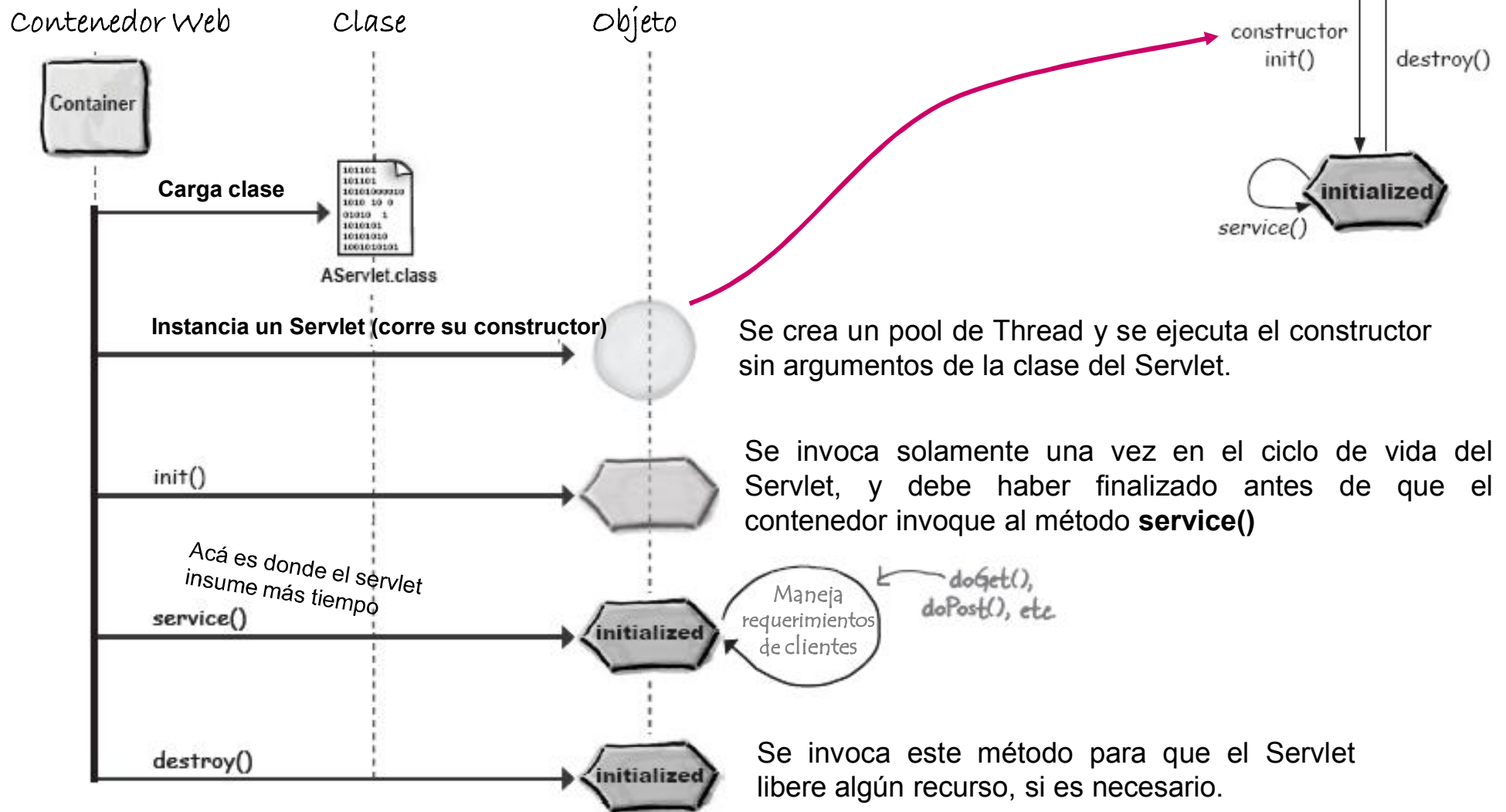


El método **service()** finaliza. El thread termina o retorna al pool de threads del contenedor. El cliente recibe la respuesta.

Servlets

Ciclo de Vida

El ciclo de vida de un servlet es muy simple; hay solamente un estado “**inicializado**”. Si no está en estado inicializado, “**no existe**” (se está inicializando o destruyendo).



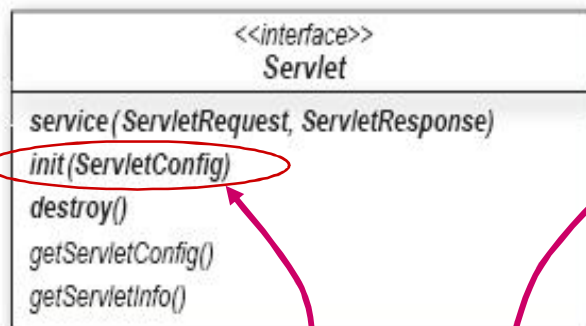
Servlets

Los servlets heredan los métodos del Ciclo de Vida

La interface Servlet

(javax.servlet.Servlet)

Todos los servlet tienen estos métodos, los 3 primeros son los del ciclo de vida.

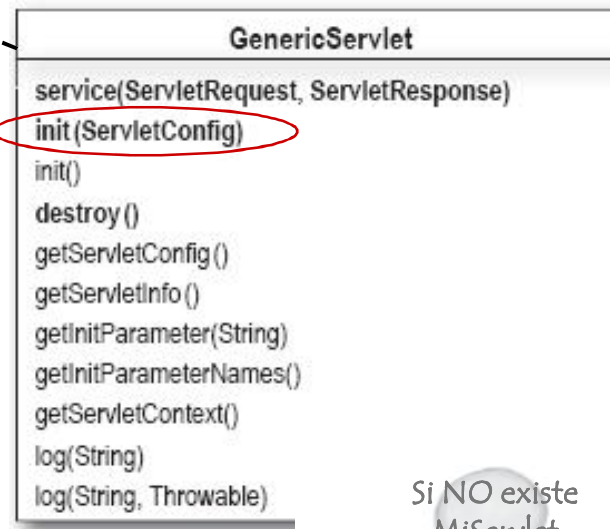


El Contenedor crea un objeto ServletConfig, que es pasado como parámetro al método init y de esta manera el servlet puede acceder a parámetros de inicialización (de la forma nombre-valor)

La clase GenericServlet

(javax.servlet.GenericServlet)

Es una clase abstracta que implementa los métodos de la interface **Servlet**.



La clase HttpServlet

(javax.servlet.http.HttpServlet)

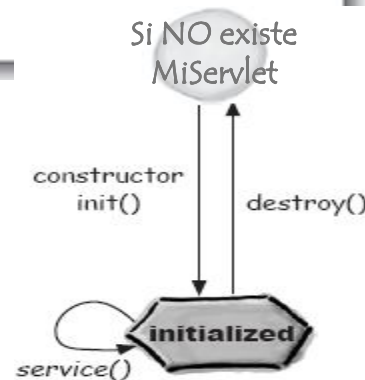
Es una clase abstracta que implementa el método **service()** para que decida qué método invocar, basado en el método HTTP.



métodos vacíos



Se sobrescribe para proveer código de inicialización (acceso a DB, guardar objetos en el contexto, etc.)



La interface de programación

La interface Servlet – La clase HttpServlet

Las clases e interfaces para implementar servlets están agrupadas en dos paquetes:

- **javax.servlet**: contiene la interface básica de servlets, llamada **Servlet**, la cual es la abstracción central de la API de servlets.

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

- **javax.servlet.http**: contiene la clase **HttpServlet** que implementa la interface Servlet y una serie de clases e interfaces específicas para atender requerimientos HTTP. La clase HttpServlet provee una implementación específica para HTTP de la interface javax.servlet.Servlet. Agrega métodos adicionales, que son invocados automáticamente por el método service() para ayudar al procesamiento de requerimientos HTTP. Es la clase a partir de la cual se crean la mayoría de los servlets HTTP.

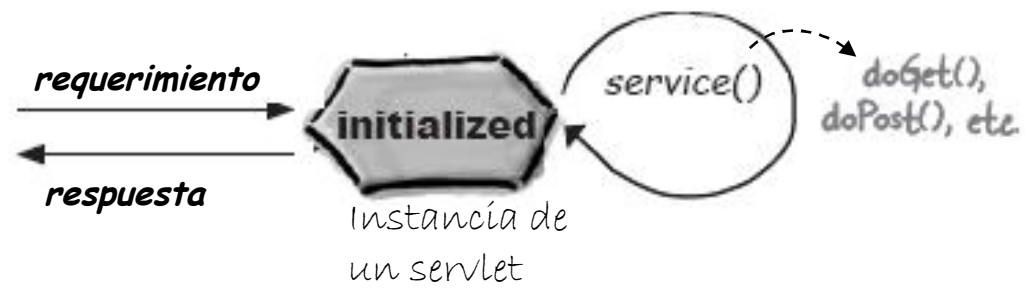
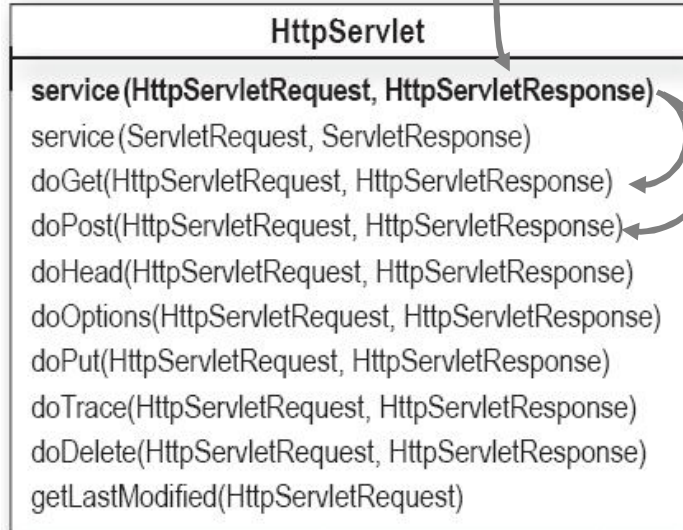
```
public void service(HttpServletRequest req, HttpServletResponse res) throws . . .
```

La interface de programación

La clase HttpServlet – El método service(...)

¿ cómo funciona el método **service (...)** de la clase **HttpServlet**?

El método **service (...)** mapea cada método del requerimiento HTTP con un método java de la clase HttpServlet.

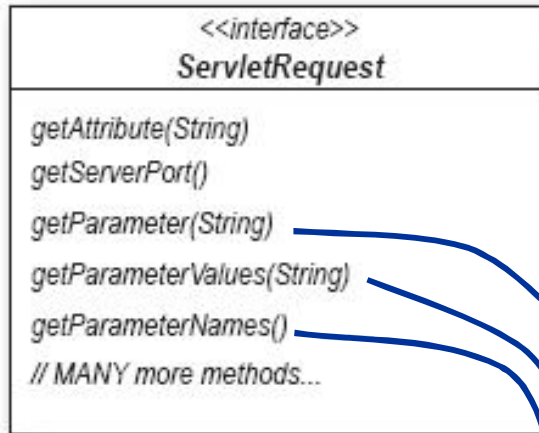


- La secuencia de métodos **service() --> doGet()** ó **service() --> doPost()** sucede cada vez que un cliente hace un requerimiento.
- Cada vez que un **doGet()** o **doPost()** ejecutan, lo hacen en un *thread* separado.
- Cuando se escribe un Servlet, los métodos que se sobrescriben son **doGet()** y **doPost()**, los restantes están relacionados con la programación más cercana al protocolo HTTP.

La interface de programación

Las interfaces `HttpServletRequest`

ServletRequest interface
(`javax.servlet.ServletRequest`)



El requerimiento HTTP de un cliente está representado por un objeto **`HttpServletRequest`**.

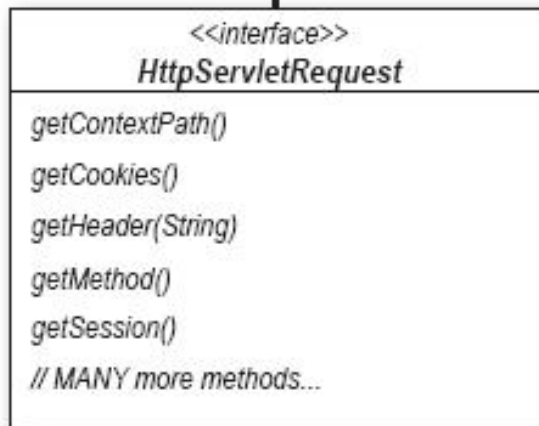
Un objeto **`HttpServletRequest`** se puede usar para recuperar el header del requerimiento HTTP, recuperar los parámetros del requerimiento HTTP, asociar atributos con el requerimiento, redireccionar requerimientos entre servlets, recuperar la sesión del usuario, etc.

Devuelve un **`String`** con el valor del parámetro del requerimiento con la clave dada. Si hay múltiples valores para esa parámetro, devuelve el primero.

Retorna un **arreglo de `Strings`** que contiene todos los valores de un parámetro del request con la clave dada o **`null`** si el parámetro no existe.

Devuelve una lista de **`Strings`** con los nombres de todos los parámetros del requerimiento.

HttpServletRequest interface
(`javax.servlet.http.HttpServletRequest`)

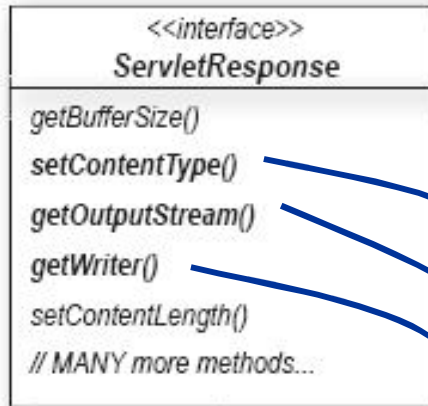


La interface de programación

Las interfaces `HttpServletResponse`

ServletResponse interface

(`javax.servlet.ServletResponse`)



El objeto **HttpServletResponse** representa la respuesta que se le enviará al cliente. Por defecto, la respuesta HTTP está vacía. Para generar una respuesta *customizada*, es necesario usar los métodos **getWriter()** o **getOutputStream()**, para obtener un *stream* de salida donde escribir contenido.

Permite setear el tipo MIME de la respuesta HTTP (text/html, image/JPG, etc.) antes de devolver la respuesta.

El objeto **ServletOutputStream** es usado para enviar al cliente datos binarios (imágenes por ejemplo).

El objeto **PrintWriter** que devuelve, es usado por el servlet para escribir la respuesta como texto.

HttpServletResponse interface

(`javax.servlet.http.HttpServletResponse`)



Un ejemplo simple

Un servlet que recupera parámetros del requerimiento

Este es un ejemplo simple de un servlet que genera una página HTML usando un parámetro del requerimiento.

saludo.html

```
<html>
<body>
<form action="ServletHola" method="post">
  Ingresá tu nombre: <input type="text" name="nombre">
  <input type="submit" value="Enviar">
</form>
</body>
</html>
```

HttpServlet, extiende **GenericServlet**, la cual implementa la interface **Servlet**

```
package servlets;
public class ServletHola extends javax.servlet.http.HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        out.println("<h1> Hola " + request.getParameter("nombre") + "</h1>");
        out.print("</body></html>");
        out.close();
    }
}
```

A partir del objeto **response** se puede obtener un objeto **PrintWriter** que nos permite escribir texto HTML en la respuesta

A partir del objeto **request** se puede obtener los parámetros del requerimiento

El archivo descriptor de la Aplicación Web

- El archivo descriptor de la aplicación web, **web.xml**, define TODO lo que el servidor necesita conocer sobre la aplicación web.
- Es estándar y se ubica SIEMPRE en la carpeta **/WEB-INF/web.xml**.
- La especificación de Servlets incluye un Document Type Descriptor (DTD) para el web.xml que define su gramática. Por ej. los elementos descriptores **<filter>**, **<servlet>** y **<servlet-mapping>** deben ser ingresados en el orden establecido por el DTD. En general los contenedores fuerzan estas reglas cuando procesan los archivos **web.xml**
- Al ser declarativa la información contenida en el archivo **web.xml** es posible modificarla sin necesidad de modificar el código fuente de las componentes.
- En ejecución, el contenedor web lee el archivo **web.xml** y actúa en consecuencia.

Los IDEs (Eclipse, JDeveloper, etc.) proveen editores visuales y ayudas durante el desarrollo de la aplicación web, que permiten crear, actualizar y editar en forma simple y consistente el web.xml.

Usando el archivo descriptor de la Aplicación Web

- Para que un cliente pueda acceder a un servlet, debe declararse una URL o un conjunto de URL's asociadas al servlet, en el archivo descriptor de la aplicación web o **web.xml**. Además, el archivo **.class** del servlet se debe ubicar en la carpeta estándar **/WEB-INF/classes** de la aplicación web, junto con otras clases Java. Cualquier contenido de la carpeta **/WEB-INF** no está accesible directamente por un cliente http.
- El archivo **web.xml** usa los elementos **<servlet>** y **<servlet-mapping>** para declarar los servlets que serán cargados en memoria por el contenedor web y para mapearlos con una URL o conjunto de URL's respectivamente.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ... http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <servlet>
    <servlet-name>ServletHola</servlet-name>
    <servlet-class>servlets. ServletHola</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletHola</servlet-name>
    <url-pattern>/ServletHola</url-pattern>
  </servlet-mapping>
</web-app>
```

Se declara un servlet, asignándole un nombre único y una clase Java que lo implementa

*Se mapea un servlet con una URL. Este es un mapeo 1 a 1, **/ServletHola**, siempre empieza con /*

URL completa del servlet: **http://www.servidor.gov.ar:8080/appPruebas/ServletHola**

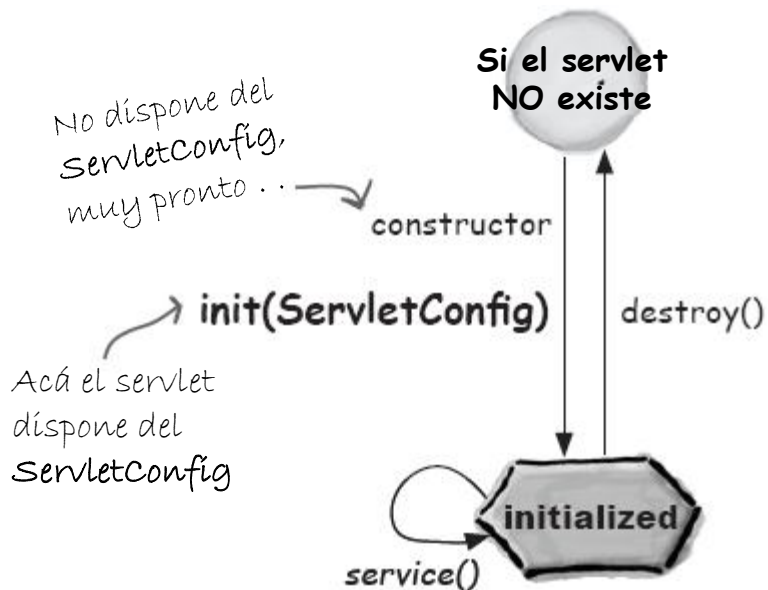
URL de la aplicación web

url-pattern o mapping

Parámetros de Inicialización del Servlet

Hasta ahora vimos que mediante los métodos `doGet()` y `doPost()` podemos tomar parámetros del requerimiento, pero además, los servlet pueden recuperar parámetros de inicialización desde el archivo `web.xml`. Cuando el contenedor web inicializa un servlet, crea un objeto `ServletConfig` y se lo pasa al servlet en el método `init()`.

A partir de este objeto se pueden recuperar parámetros desde el archivo `web.xml` usando el método `getInitParameter(String s)`



Para definir parámetros de configuración inicial de un servlet, se usan los sub-elementos `<init-param>`, `<param-name>` y `<param-value>` en el `web.xml`.

`web.xml`

```
<servlet>
  <servlet-name>ServletHola</servlet-name>
  <servlet-class>servlets.ServletHola</servlet-class>
  <init-param>
    <param-name>saludo</param-name>
    <param-value>Hola</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>ServletHola</servlet-name>
  <url-pattern>/ServletHola</url-pattern>
</servlet-mapping>
...
```

Parámetros de Inicialización

Otro ejemplo de Servlet

Este Servlet retorna una página HTML con un mensaje tomado de parámetros inicialización y con la hora/fecha actual.

```
public class ServletFecha extends HttpServlet {
    private String dia, hora;
    public void init(){
        dia = this.getServletConfig().getInitParameter("dia");
        hora = this.getServletConfig().getInitParameter("hora");
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ..{
        PrintWriter out = response.getWriter();
        java.util.Date d = new java.util.Date();
        out.println("<html><body>");
        out.println("<h1>"+dia+DateFormat.getDateInstance().format(d) + "</h1>");
        out.println("<h1>"+hora+DateFormat.getTimeInstance().format(d)+" hs.</h1>");
        out.println("</body></html>");
        out.close();
    }
}
```

web.xml

```
<servlet>
<servlet-name>ServletFecha</servlet-name>
<servlet-class>misServlets30.ServletFecha</servlet-class>
  <init-param>
    <param-name>dia</param-name>
    <param-value>Hoy es:</param-value>
  </init-param>
  <init-param>
    <param-name>hora</param-name>
    <param-value>Son las:</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>ServletFecha</servlet-name>
  <url-pattern>/ServletFecha</url-pattern>
</servlet-mapping>
```

También podría no sobrescribirse el método init() y recuperar los valores de inicialización cuando se quiera usar, de alguna de estas dos maneras:

```
this.getServletConfig().getInitParameter("hora")
this.getInitParameter("hora")
```

Parámetros de Inicialización del Servlet

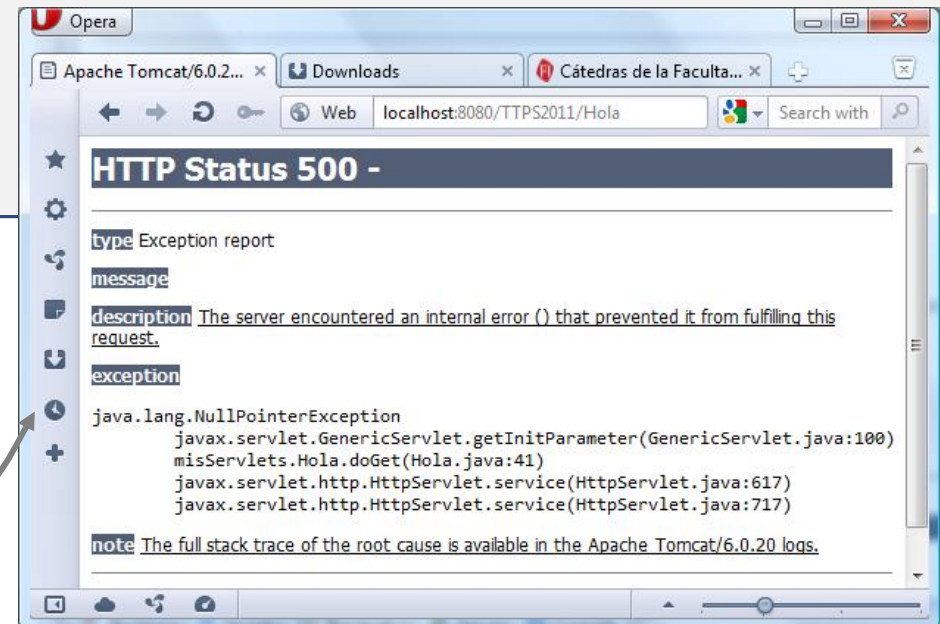
```
public class ServletFecha extends HttpServlet {
    private String dia, hora;

    public void init(SevletConfig config) {
        super.init(config);
        // dia = config.getInitParameter("dia");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ... {
        PrintWriter out = response.getWriter();
        java.util.Date d = new java.util.Date();
        out.print("<html><body>");
        out.print("<h1>" + this.getInitParameter("dia") + "</h1>");
        out.print("<h1>" + DateFormat.getDateInstance().format(d) + "</h1>");
        out.print("</body></html>");
        out.close();
    }
}
```

NOTA: es recomendable sobrescribir el **init()** sin parámetros.
Si sobrescriben el **init(ServletConfig c)** deben invocar al **init(config)** de la superclase para que el servlet quede bien inicializado!!

Si se quiere invocar al **getInitParameter()** desde el **doGet()** / **doPost()** y en el **init(ServletConfig config)** sobrescrito no se invocó al **init(ServletConfig)** de la super clase, la ejecución del servlet disparará este error:



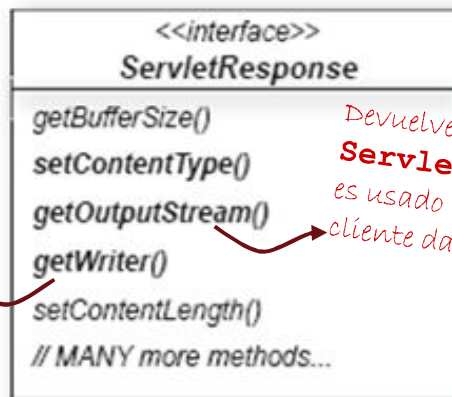
Las interfaces de programación

HttpServletResponse y ServletContext

Vimos que el objeto **HttpServletResponse** tiene métodos que retornan objetos donde se puede escribir el contenido que se enviará en la respuesta.

También existe la interfaz **ServletContext** que define una vista de la aplicación para los Servlets. El objeto **ServletContext** permite obtener referencias a URLs de recursos web.

ServletResponse interface (javax.servlet.ServletResponse)



Devuelve un objeto **PrintWriter** que, es usado por el servlet para escribir la respuesta como texto.

Devuelve un objeto **ServletOutputStream** es usado para enviar al cliente datos binarios.

HttpServletResponse interface (javax.servlet.http.HttpServletResponse)



<<interface>> ServletContext

```
getInitParameter(String)
getInitParameterNames()
getAttribute(String)
getAttributeNames()
setAttribute(String, Object)
removeAttribute(String)
-----
getResourceAsStream(String)
getRequestDispatcher(String)
-----
log(String)
// more methods
```

Cada Contenedor Web provee una implementación específica de esta interface. La funcionalidad es idéntica, no depende de la implementación, está establecida en la interface.

Devuelve una referencia a un objeto **InputStream** para acceder a un recurso web indicado en el parámetro

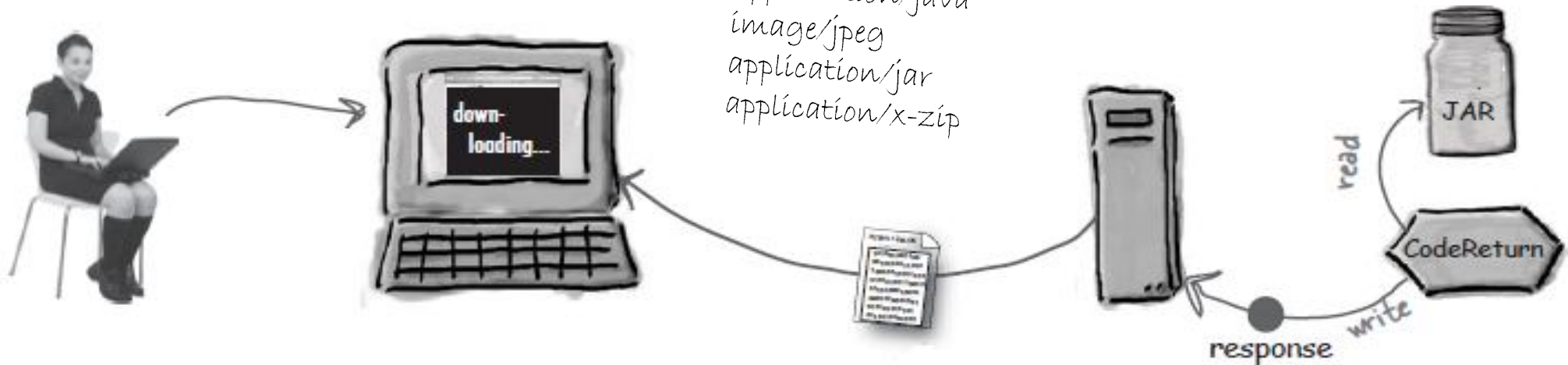
Devuelve un objeto **RequestDispatcher** al que se le puede delegar un requerimiento.

Servlets

Tipos de respuestas

Los tipos mas comunes de MIME son:

- text/html
- application/pdf
- video/quicktime
- application/java
- image/jpeg
- application/jar
- application/x-zip



Tenemos un JAR con código fuente java que queremos enviárselo a nuestros clientes usando un Servlet.

¿Qué tipo MIME usamos?

¿Qué objeto usamos para escribir la respuesta?

Servlets

Tipos de respuestas

Para enviar datos binarios desde un servlet se debe usar el objeto **OutputStream** retornado por el método **getOutputStream()**

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

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

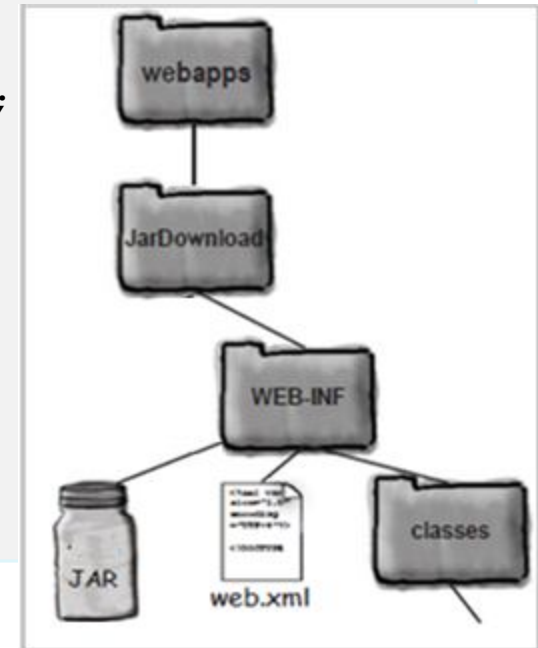
        response.setContentType("application/jar");

        ServletContext ctx = this.getServletContext();

        InputStream is = ctx.getResourceAsStream("/WEB-INF/codigo.jar");
        // Devuelve el recurso ubicado en el path
        // especificado, como un objeto InputStream

        int read = 0;
        byte[] bytes = new byte[1024];
        OutputStream os = response.getOutputStream();
        while ((read = is.read(bytes)) != -1) {
            os.write(bytes, 0, read);
        }
        os.flush();
        os.close();
    }
}
```

El método **getResourceAsStream()** requiere que el argumento comience con **"/"**, que representa la raíz de la aplicación web.



Servlets

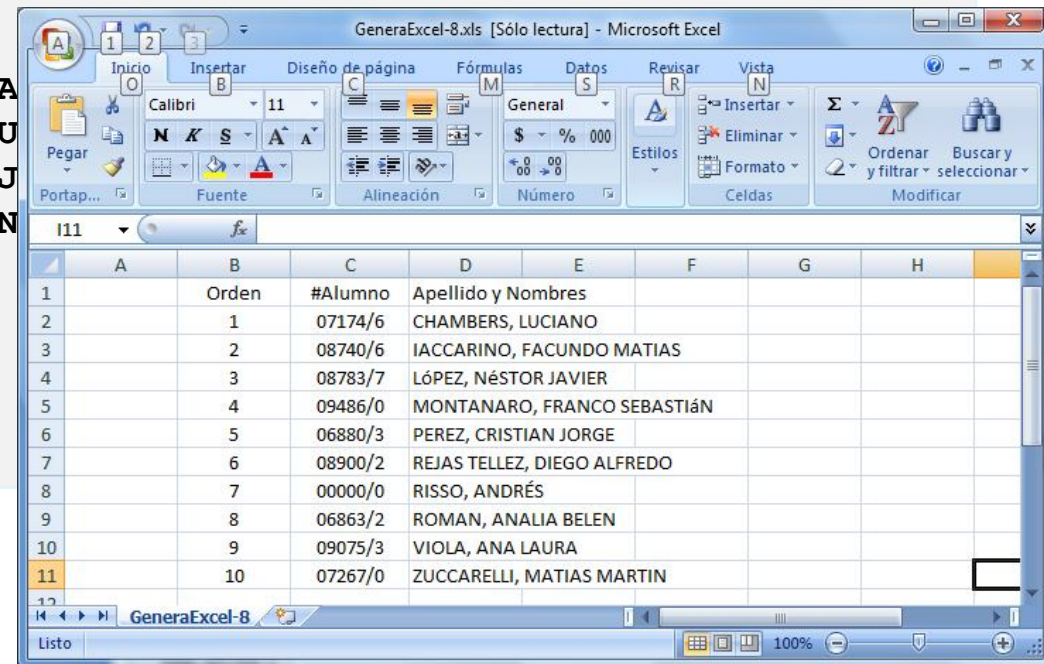
Tipos de respuestas

Otro tipo de respuesta podría una planilla excel y para ello el content-type se debe setear con **"application/vnd.ms-excel"** y se debe obtener un objeto **PrintWriter**.

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class GeneraExcel extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {

        response.setContentType("application/vnd.ms-excel");
        PrintWriter out = response.getWriter();
        out.println("\tOrden\t#Alumno\tApellido y
        out.println("\t1\t07174/6\tCHAMBERS, LUCIA
        out.println("\t2\t08740/6\tIACCARINO, FACU
        out.println("\t3\t08783/7\tLÓPEZ, NÉSTOR J
        out.println("\t4\t09486/0\tMONTANARO, FRAN
        . . .
        out.close();
    }
}
```



The screenshot shows a Microsoft Excel window titled 'GeneraExcel-8.xls [Sólo lectura] - Microsoft Excel'. The spreadsheet contains a table with 4 columns: Orden, #Alumno, and Apellido y Nombres. The data is as follows:

Orden	#Alumno	Apellido y Nombres
1	07174/6	CHAMBERS, LUCIANO
2	08740/6	IACCARINO, FACUNDO MATIAS
3	08783/7	LÓPEZ, NÉSTOR JAVIER
4	09486/0	MONTANARO, FRANCO SEBASTIÁN
5	06880/3	PEREZ, CRISTIAN JORGE
6	08900/2	REJAS TELLEZ, DIEGO ALFREDO
7	00000/0	RISSO, ANDRÉS
8	06863/2	ROMAN, ANALIA BELEN
9	09075/3	VIOLA, ANA LAURA
10	07267/0	ZUCCARELLI, MATIAS MARTIN

Los datos para la planilla se podrían tomar de alguna fuente de datos.

Transferir el control

`sendRedirect()` del objeto `HttpServletResponse`

Algunas veces el servlet puede redireccionar el requerimiento a otro recurso del mismo contenedor web o a una URL de otro dominio.

El `sendRedirect()` hace trabajar al navegador

¿Cómo lo hace?

El servlet invoca al método `sendRedirect(String url)` sobre la respuesta. La respuesta HTTP lleva el código 302 que indica "El recurso que está buscando el cliente fue temporariamente movido". El navegador obtiene la respuesta, ve el código de estado "302" genera un nuevo requerimiento usando la URL que recibió como parámetro (el usuario puede observar en la barra del navegador que la URL cambia) y finalmente muestra una página al usuario que no fue la que el originalmente pidió.

<code><<interface>></code>
<code>HttpServletResponse</code>
<code>addCookie()</code>
<code>addHeader()</code>
<code>encodeURL()</code>
<code>sendError()</code>
<code>setStatus()</code>
<code>sendRedirect()</code>
<code>// MANY more methods...</code>

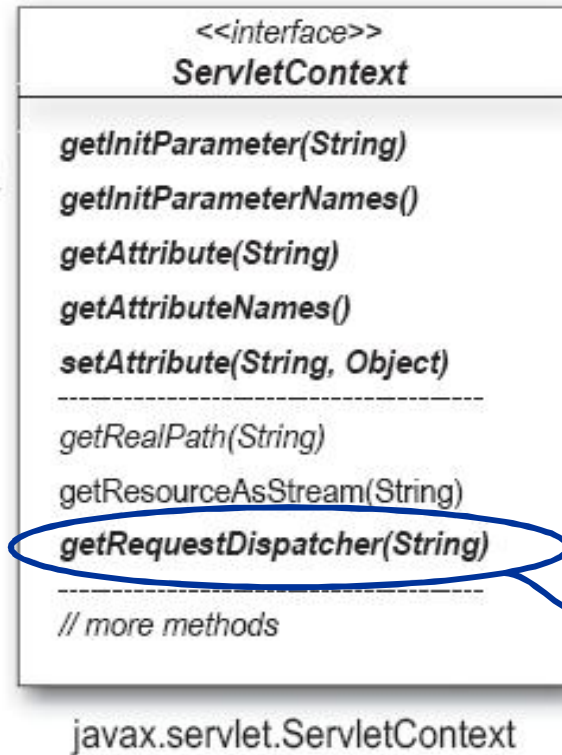
```
public void doPost (HttpServletRequest request, HttpServletResponse response) {  
    String person = request.getParameter("name");  
    if(person==null) {  
        response.sendRedirect("/app/DatosMal.html");  
        return;  
    }  
    . . .  
}
```

La url que se quiere que el navegador use para crear el requerimiento. Es la que verá el cliente

Se puede usar una url relativa o absoluta: **"http://www..."**

Transferir el control

forward() del objeto ServletContext



Usar el método forward() del objeto **RequestDispatcher** es otro mecanismo para transferir el control.

A diferencia del redireccionamiento de la respuesta, este mecanismo no requiere de ninguna acción por parte del cliente ni del envío de información extra entre el cliente y el servidor.

El proceso íntegro de delegación del requerimiento se realiza del lado del servidor. Además, este mecanismo permite pasar el requerimiento a otro servlet para que continúe el procesamiento y responda al cliente.



Instancia un
RequestDispatcher
para un servlet

```
public class ForwardServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        . . .
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/mostrar");
        if (dispatcher != null) {
            request.setAttribute("hora", new Date());
            dispatcher.forward(request, response);
        } . . .
    }
}
```

Con el forward() trabaja más el servidor

Transferir el control

include() del objeto ServletContext



El objeto **RequestDispatcher** también cuenta con el método **include()** que se utiliza de manera similar que el **forward()** y que incluye contenido del lado del servidor en la respuesta que se está generando. El servlet que funciona como receptor del método `include()` – en el ejemplo los servlets **header** y **footer**– tienen acceso al requerimiento y a la respuesta original.

```
public class IncludeServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        . . .
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/header");
        if (dispatcher != null)
            dispatcher.include(request, response);
        . . .
        dispatcher = getServletContext().getRequestDispatcher("/footer");
        if (dispatcher != null)
            dispatcher.include(request, response);
    }
    . . .
}
```

Acá se podrían setear atributos en el request antes de delegar

Servlets con Anotaciones

Una alternativa al web.xml

A partir de la versión de Servlet 3.0 (la actual es 3.1) se pueden utilizar anotaciones para la configuración de los Servlets. Las anotaciones en la API de servlets se utilizan para reemplazar a las declaraciones/los mapeos del archivo web.xml.

¿qué son las anotaciones?

- Las anotaciones son metadatos que nos permiten agregar información a nuestro código fuente para ser usado posteriormente –en tiempo de compilación o en tiempo de ejecución–.
- Las anotaciones fueron incorporadas al lenguaje java en la versión 5. La motivación de las anotaciones es la tendencia a combinar metadatos con código fuente, en lugar de mantenerlos en archivos descriptores separados.

La API de Servlets 3.0

Anotaciones

Las anotaciones *se declaran* de manera parecida a las interfaces, solo que el signo **@** precede a la palabra clave `interface`. Se compilan a archivos `.class` de la misma manera que las clases e interfaces.

```
package java.lang;
import java.lang.annotation.*;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override {
}
```

Las anotaciones *se utilizan* en el código fuente precediendo a la declaración de la clase, atributos y métodos. En este caso, la anotación **@Override** es para métodos y se usa así:

```
public class Paciente {

    @Override
    public String toString() {
        return super.toString();
    }
}
```

Precede a un
método

La anotación **@Override** indica que se está sobrescribiendo un método de la superclase. Si un método está precedido por esta anotación pero NO sobrescribe el método de la superclase, los compiladores deben generar un mensaje de error y la clase no compila.

@Target: indica dónde se aplican las anotaciones (métodos, clases, variables de instancia, variables locales, paquetes, constructores, etc).

@Retention: indica dónde están disponibles las anotaciones y cuánto se mantiene la información de las anotaciones. Esto permite determinar si pueden ser leídas solo por el compilador o también en tiempo de ejecución. Los valores posibles son: **RetentionPolicy.SOURCE**, **RetentionPolicy.CLASS** y **RetentionPolicy.RUNTIME**.

La API de Servlets 3.0

Anotaciones

Este es un ejemplo de una declaración de la anotación `@Column` para el mapeo de objetos con tablas de una base de datos, donde tiene entre otros el método `name()` para identificar en nombre de la columna en la tabla de la base de datos.

Definición de la anotación `@column`

```
import javax.persistence.*;
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@interface Column extends Annotation{

    public String name() default "";
    . . .
}
```

La declaración de una anotación al igual que las interfaces tiene métodos abstractos pero además puede tener valores por defecto.

Uso de la anotación `@column`

```
package taller;

import javax.persistence.*;
@Entity
@Table(name="MENSAJES")
public class Mensaje {

    @Column(name="MENSAJE_ID")
    private Long id;
    . . .
}
```

Preceden a la declaración de la clase

La anotación tiene una lista entre paréntesis de pares **elemento-valor**. Los valores de los elementos deben ser constantes definidas en compilación

La anotación **@WebServlet** es usada para declarar la configuración de un Servlet. Si no se usa el atributo **name** se usa el nombre de la clase.

La API de Servlets 3.0

Un Servlet con Anotaciones

```
package misServlet;  
@WebServlet(  
    urlPatterns = {"/ServletFecha"},  
    initParams = {  
        @WebInitParam(name = "dia", value = "Hoy es: "),  
        @WebInitParam(name = "hora", value = "Son las: ")  
    })  
public class ServletFecha extends HttpServlet{  
    private String dia, hora;  
  
    public void init(){  
        dia = this.getServletConfig().getInitParameter("dia");  
        hora = this.getServletConfig().getInitParameter("hora");  
    }  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        PrintWriter out = response.getWriter();  
        java.util.Date d = new java.util.Date();  
        out.print("<html><body>");  
        out.print("<h1>" + dia + DateFormat.getDateInstance().format(d) + "</h1>");  
        out.print("<h1>" + hora + DateFormat.getTimeInstance().format(d) + " hs.</h1>");  
        out.print("</body></html>");  
        out.close();  
    }  
}
```

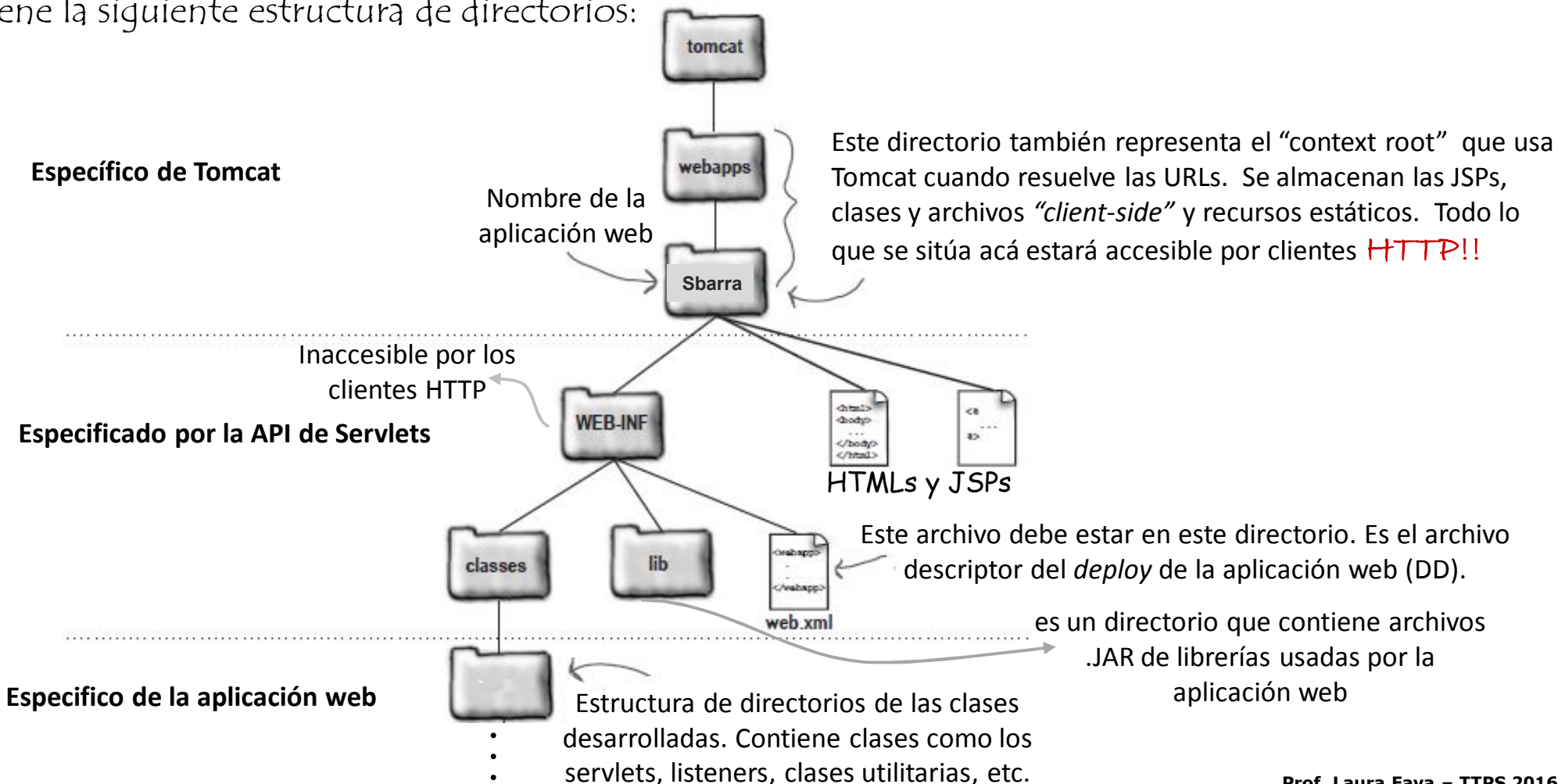
El atributo **urlPatterns** define un conjunto de url-patterns que pueden ser usadas para invocar al Servlet.

La anotación **@WebInitParam** se usa para definir los parámetros de inicialización del servlet

El Módulo web

Un **módulo web** es una unidad “desplegable” de recursos web (componentes web y archivos estáticos que pueden referenciarse por una URL). También puede contener clases utilitarias “*server-side*” (por ej: javaBeans) y clases “*client-side*” (applets y clases utilitarias).

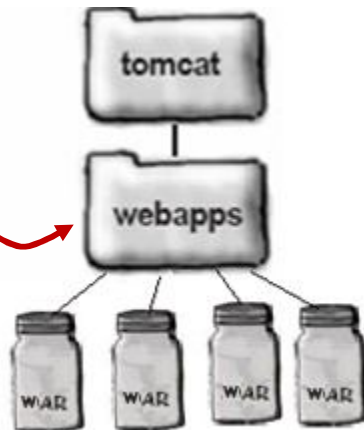
De acuerdo a la especificación de servlets, un **módulo web** se corresponde con una **aplicación web** y tiene la siguiente estructura de directorios:



¿Cómo se hace el “deploy” de una aplicación?

- Las aplicaciones web JAVA pueden empaquetarse en un archivo *Web ARchive* (WAR). El archivo WAR es ideal para distribuir e instalar una aplicación. El formato “desempaquetado” es útil en la *etapa de desarrollo*.
- Un WAR tiene una estructura de directorios específica, donde la raíz, es el “*context root*” de la aplicación web.
- El archivo WAR es un archivo JAR que contiene un módulo web: páginas HTML, archivos de imágenes, JSPs, clases, páginas de estilo, código JavaScript, el directorio **WEB-INF** y sus subdirectorios (classes, lib, tag, el archivo web.xml, etc.).
- Los archivos WAR están definidos oficialmente en la especificación de Servlets a partir de la versión 2.2. **Son estándares**. Todos los contenedores que implementan la especificación de la API de Servlets 2.2 y superiores deben soportar archivos WAR.
- Los IDEs proveen opciones que permiten construir el WAR en forma automática. Se puede crear el archivo WAR usando la herramienta *jar* del J2SDK.

En el servidor Tomcat, el archivo WAR de la aplicación web se debe copiar en el directorio *webapps*



Cuando Tomcat arranca, automáticamente expande a partir de *webapps* el contenido de cada uno de los archivos .war al formato “desempaquetado”.

Si usamos esta técnica para hacer el “deployment” de nuestra aplicación y necesitamos actualizarla, debemos reemplazar el .WAR y **ELIMINAR** la estructura de directorios expandida y luego re-iniciar Tomcat.

Referencias

- Servlets y JavaServer Pages, Jayson Falkner, Kevin Jones
- Head First Servlets & JSP, Bryan Basham, Kathy Sierra, Bert Bates. O'Reilly
- Concurrencia con JAVA: <http://download.oracle.com/javase/tutorial/essential/concurrency/>

Herramientas necesarias para el desarrollo de aplicaciones web:

- (1) Apache Tomcat 8, <http://tomcat.apache.org/download-80.cgi> (implementa las especificaciones Servlet 3.1 y JSP 2.3 del JCP).
- (2) Eclipse, IDE para desarrollar aplicaciones Java EE. La versión mas actual es Mars, <http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/marsr>
- (3) Además se necesita disponer de la plataforma estándar, JSE 8 (JDK), <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>