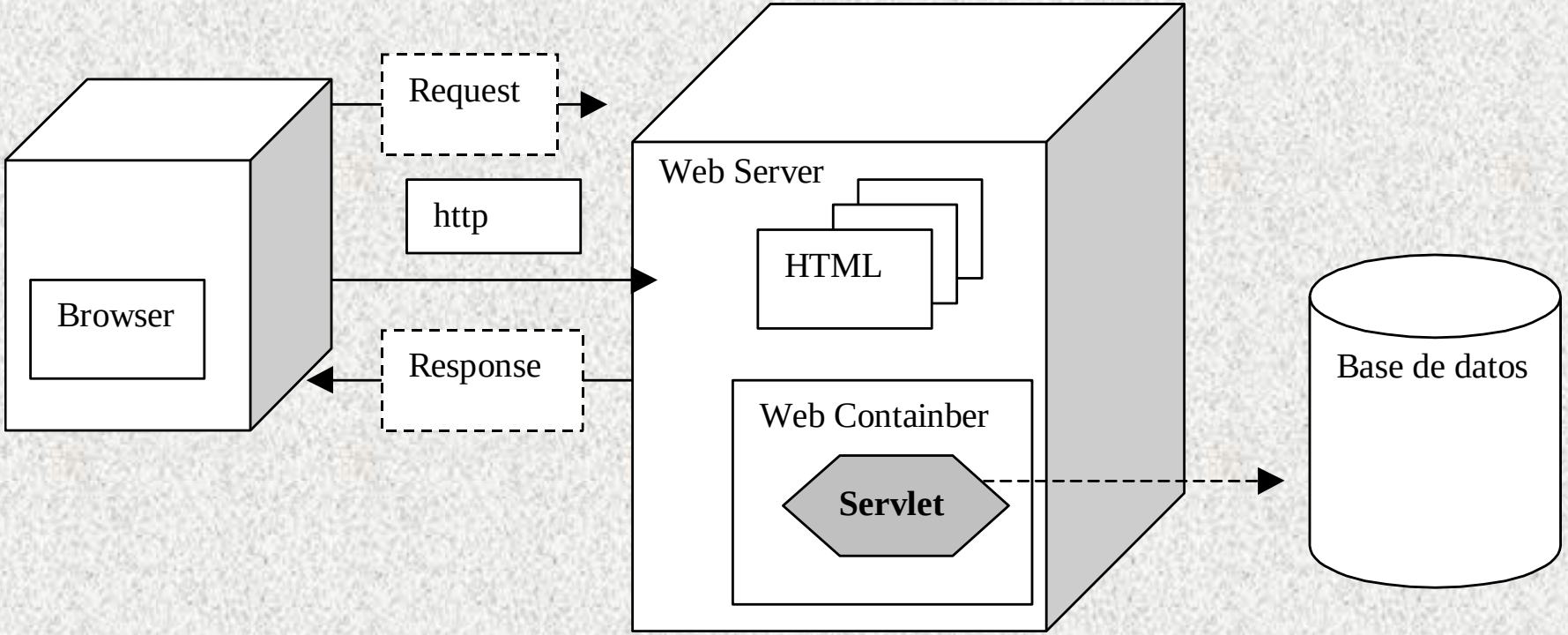
Desarrollo de Software para Sistemas Distribuidos

Curso 2015

Servlet

- Un servlet Java es un programa Java que, al igual que el CGI, corre del lado del servidor.
- A diferencia de los CGI, los servlet Java se ejecutan en una contenedor que es una componente mas de la arquitectura del Web Server y se denomina **Web Container**. El Web Container es una maquina virtual Java (JVM) que brinda una implementación de la interface de aplicación del servlet (API). Las instancias de los servlets son componentes que utiliza el Web Container para responder a los requerimientos HTTP.

Servlet


Request
http
Browser
Response
Web Server
HTML
Web Containber
Servlet
Base de datos

Ventajas de los servlet

- Cada requerimiento corre en hilos separados que lo hace muchos mas rápido que los procesos CGI.
- Son escalables.
- Son robustos y orientados a objetos.
- Son escritos en un único lenguaje: Java.
- Son independientes de la plataforma (porque están escritos en Java).
- El Web Container les proporciona servicios adicionales para seguridad y manejo de errores.

Desventajas de los Servlet

- Frecuentemente mezclan lógica de negocios con lógica de la presentación.
- Deben manejar la concurrencia.

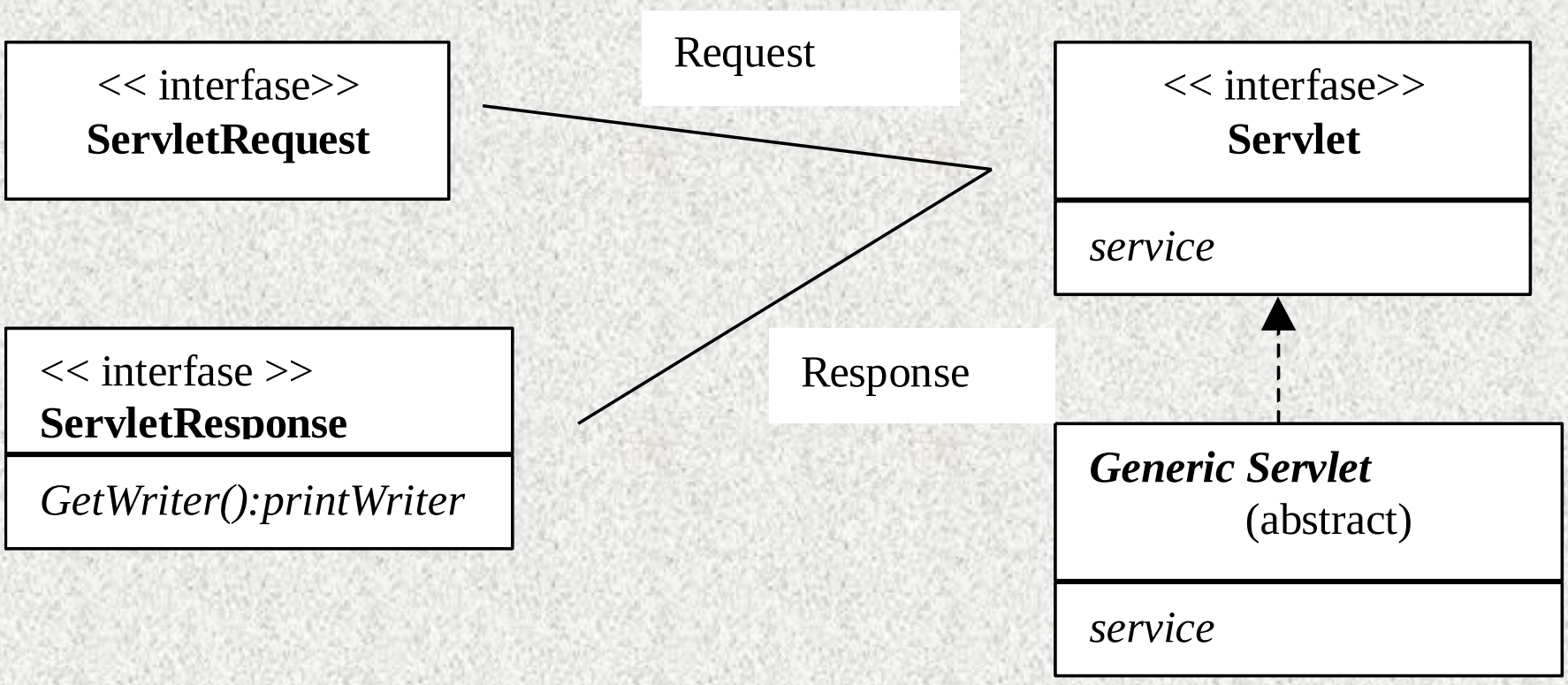
JSP

- JSP (Java Server Page) son páginas HTML con código embebido, donde ese código es Java.
- JSP son el opuesto a los servlets. Estos últimos son programas de tecnología Java que contienen HTML mientras que JSP son paginas HTML que contienen código Java. Las JSP son convertidas por el Web Container en instancias de servlets.

Tecnología J2EE

- Los servlets y las paginas JSP son un parte integral de la plataforma Java 2 Enterprise Edition (**J2EE**). En esta plataforma también se incluyen un Enterprise Java Bean container (**EJB**). EJB provee un marco de desarrollo para componentes de negocio proveyendo también servicios para manejo de transacciones, persistencia, seguridad y manejo del ciclo de vida de los Beans.

API Servlet


<< interfase>>
ServletRequest
Request
<< interfase>>
Servlet
service
<< interfase >>
ServletResponse
GetWriter():printWriter
Response
Generic Servlet
(abstract)
service

API Servlet

- Usualmente, un proyecto no crea nuevos protocolos de red, sino que usa HTTP. En tal caso no se utiliza el Servlet genérico sino su descendiente HTTP.
- El protocolo HTTP especifica el requerimiento como un recurso ubicable a través de una URL. El método por el cual se realiza este requerimiento, usualmente es el método GET.
- El API de Java define dos interfases HTTPServletRequest y HTTPServletResponse para manejar este Los servlets Java son componentes (clases) que deben existir en un Web Container.

Uso de forms HTML en Servlet

- Un formulario HTML es un tag HTML que permite incluir áreas de ingreso de datos dentro de una página y que serán luego utilizados para pasar como parámetro en el GET o en el POST como pares (nombre,valor).

Tags

- Existe conjunto de componentes GUI (textfield, SubmitButton, Checkbox, HiddenField, TextArea, etc.) que pueden utilizarse dentro del tag FORM.
- Así cada tag se identifica por:
  - TYPE: tipo de componente
  - NAME: nombre del parámetro
  - VALUE: valor del parámetro

Método GET y POST

- En el método GET los parámetros son pasados como parte de la URL (en el encabezado), mientras que en el POST, se pasan como parte del cuerpo del mensaje.
- El API Servlet provee de métodos en su interfase HTTPServletRequest que permite acceder a los datos del FORM. Estos son:
  - getParameter (name): String
  - getParameterValues (name): String[]
  - getParameterNames (): Enumeration

Deployment Descriptor

- El uso de Servlets implica que el código fuente se encuentre en algún directorio conocido.
- Se define un "mapeo" entre el Servlet y la URL que lo accede. Este mapeo se guarda en un archivo que se comporta como descriptor (por eso el nombre "deployment descriptor") y que es un archivo XML

Ejemplo

```
<servlet>
  <servlet-name>Hola</servlet-name>
  <servlet-
  class>sl314.eb.FormBaseHola</servlet-
  class>
</servlet>
<servlet-mapping>
  <servlet-name>Hola</servlet-name>
  <url-pattern>/bienvenida</url-pattern>
</servlet-mapping>
```

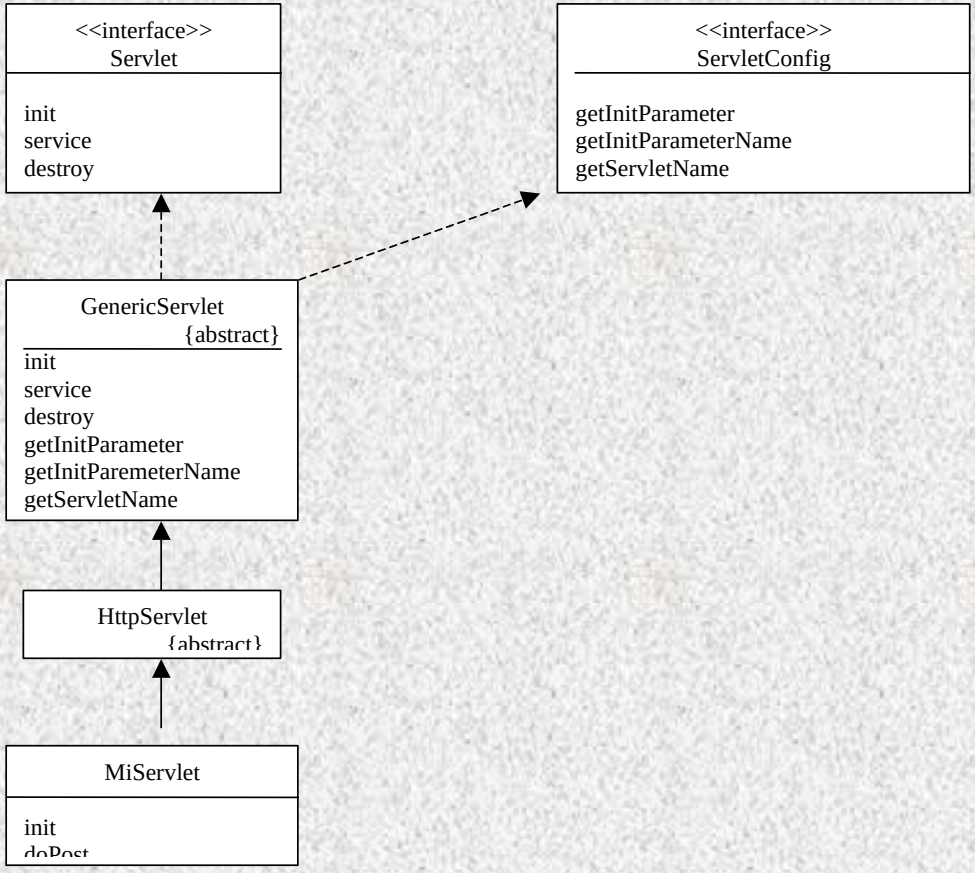
Ambiente de distribución

- Archivos HTML estático almacenados en la raíz del directorio de la aplicación
- Servlets y clases Java almacenados en el directorio *WEB-INF/classes*
- Archivos con librerías Java auxiliares (archivos con extensión JAR) almacenados en el directorio *WEB-INF/lib*
- El archivo deployment descriptor (WEB.XML) almacenado en la raíz del directorio *WEB-INF*

Ciclo de vida de un Servlet

- El Web Container administra el ciclo de vida de una instancia de servlet invocando tres métodos definidos en la interface Servlet: *init*, *service* y *destroy*
  - El método *init* es invocado por el Web Container cuando se crea la instancia del servlet.
  - El método *service* es invocado por el Web Container para procesar los requerimientos del cliente. La clase HttpServlet implementa el método *service* a través de los métodos doGet o doPost.
  - El método *destroy* es invocado por el Web Container cuando la instancia del servlet se destruye.

Clase "Servlet"



Parámetros de inicialización

- La clase GenericServlet implementa la interface ServletConfig, que otorga acceso directo a los parámetros de configuración.
- Los parámetros de inicialización no son más que pares nombre-valor que se declaran en el deployment descriptor.

Ejemplo "Hola en Ingles"

```
<servlet>
   <servlet-name>Hola en Ingles</servlet-name>
   <servlet-class>.....</servlet-class>
   <init-param>
        <param-name>textoBienvenida</param-name>
        <param-value>Hello</param-value>
   </init-param>
</servlet>
```

Ejemplo "Hola en Frances"

```
<servlet>
   <servlet-name>Hola en Frances</servlet-name>
   <servlet-class>.....</servlet-class>
   <init-param>
        <param-name>textoBienvenida</param-name>
        <param-value>Bonjour</param-value>
   </init-param>
</servlet>
```

Ejemplo: servlet "Bienvenida"

```
public class MiServlet extends HttpServlets {
    private string bienvenida;
    public void init() {
        bienvenida = getInitParameter("bienvenida");
        system.out.println(">> bienvenida =" + bienvenida + "");
    public void doPost   (HttpServletRequest request,
                          HttpServletResponse response) throws
    IOException{
    String name = request.getParameter ("name");
    If (name == null) || (name.length() == 0)) {name = 'Mundo'}
    // especifica el tipo de contenido es HTML
    response.setContentType("text/html");
    PrintWriter out = response.GetWriter();
```

Ejemplo: servlet "bienvenida" (Cont.)

```
// Genera la respuesta HTML          out.println("HTML");
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Mi Servlet </TITLE>");
    out.println("</HEAD>");
    out.println("<B>" + bienvenida + ", " + name + "</B>");
    out.println("</HTML>");
}
}
```

Servlet Context

- Una aplicación Web es un colección autocontenida de recursos estáticos y dinámicos:
  - páginas HTML
  - archivos de medios
  - archivos de recursos y de datos
  - servlets y paginas JSP
  - clases y objetos Java
- El deployment descriptor detalla la estructura y servicios usados por la aplicación Web.

Objeto Servlet Context

- El objeto *ServletContext* es la representación en tiempo de ejecución de la aplicación Web. Provee:
  - Acceso a los parámetros de inicialización y archivos de recursos
  - Acceso de lectura/escritura a los atributos de la aplicación
  - Funcionalidd de Logging

Acceso a parámetros de inicialización

```
ServletContext context = sce.getServletContext();
String catalogoNombre =
context.getInitiParameter("archivoCatalogo");
InputStream is = null;
BufferedReader catReader = null;
try {
    is =
context.getResourceAsStream(catalogoNombre);
    catReader = new BufferReader(new
InputStreamReader(is));
```

Acceso a atributos de la aplicación

```
context.setAttribute ("catalog", catalog);
    public class ShowProductList extends HttpServlet
    {
    public void doGet    (HttpServletRequest request,
                          HttpServletResponse response)
    throws IOException{
    ServletContext context = getServletContext();
    ListaProductos catalog = (ListaProductos)
    context.getAttribute("catalog");
    Iterator Items = catalog.getProductos();
```

Alcances de un Servlet

- Cada servlet dentro de una aplicación Web tiene acceso al objeto ServletContext utilizando el método *getServletContext* que es heredado de GenericServlet.
- Así como cada servlet puede tener varios parámetros de inicialización, la aplicación Web también puede tenerlos. Se almacenan bajo el tag context-param del deployment descriptor y son accedidos por el método *getInitParameter* de la clase *ServletContext*.

Ciclo de vida de una app Web

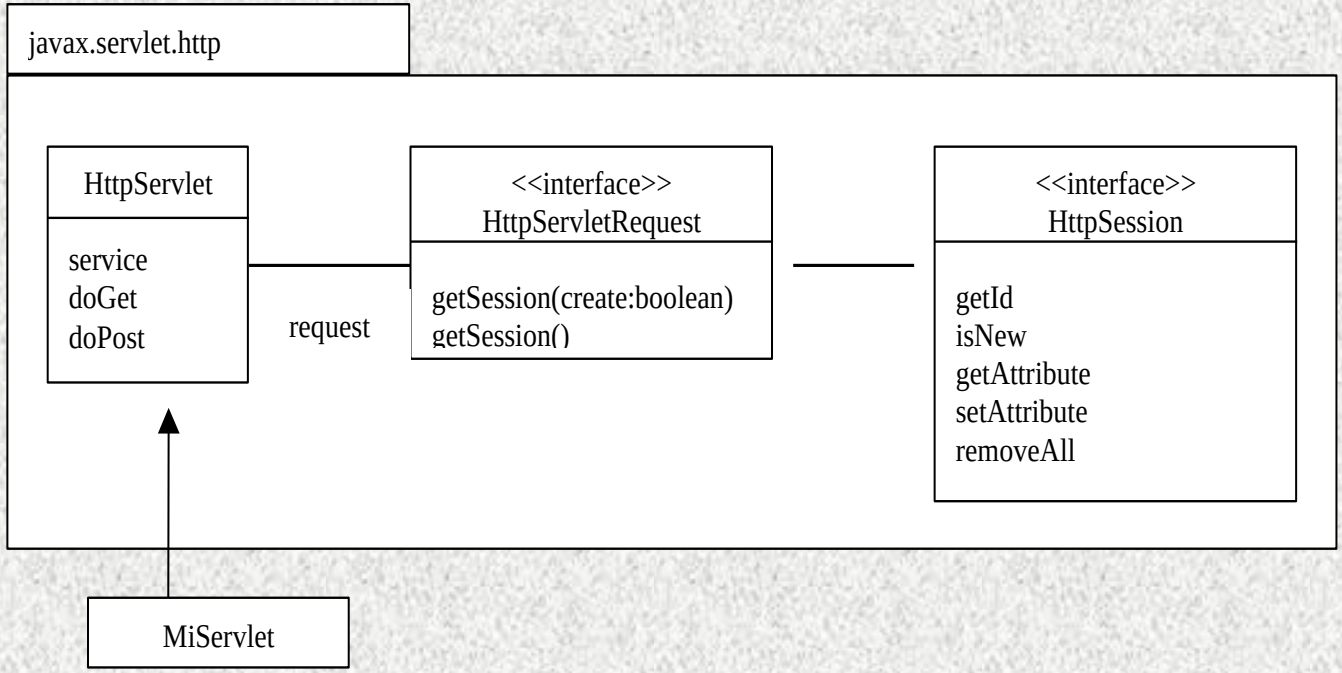
- De la misma manera que un servlet tiene un ciclo de vida, también lo tiene una aplicación Web. Cada aplicación Web se inicializa cuando se inicia el Container (como servicio) y se destruye cuando el mismo se detiene.
- Existe la interface *ServletContextListener* que detecta estos eventos de arranque y detección del Container y que pueden programarse. Asociada a esta interface esta la clase *ServletContextEvent* que provee acceso al objeto ServletContext.

Manejo de sesión

- HTTP es un protocolo sin estado. Entre un requerimiento y otro el servidor HTTP olvida lo sucedido.
- El Container debe crear un mecanismo que permita almacenar la información de la sesión de un usuario en particular.

Objeto Sesión

- Implementan la interfase HttpSession y poseen atributos que persisten a través de múltiples requerimientos



Uso de la sesión

- El servlet MiServlet tiene acceso al objeto sesión a través del request.

```
HttpSession sesion = request.getSession();
```

- El metodo getSession retorna la sesión asociada al request y si este no tiene sesión crea una. Si el objeto sesión ya existe, las próximas invocaciones a getSession, retornan siempre el mismo objeto.
- El objeto sesión puede mantener objetos utilizando los métodos xyzAttribute.

```
sesion.setAttribute ("estado", estado);
```