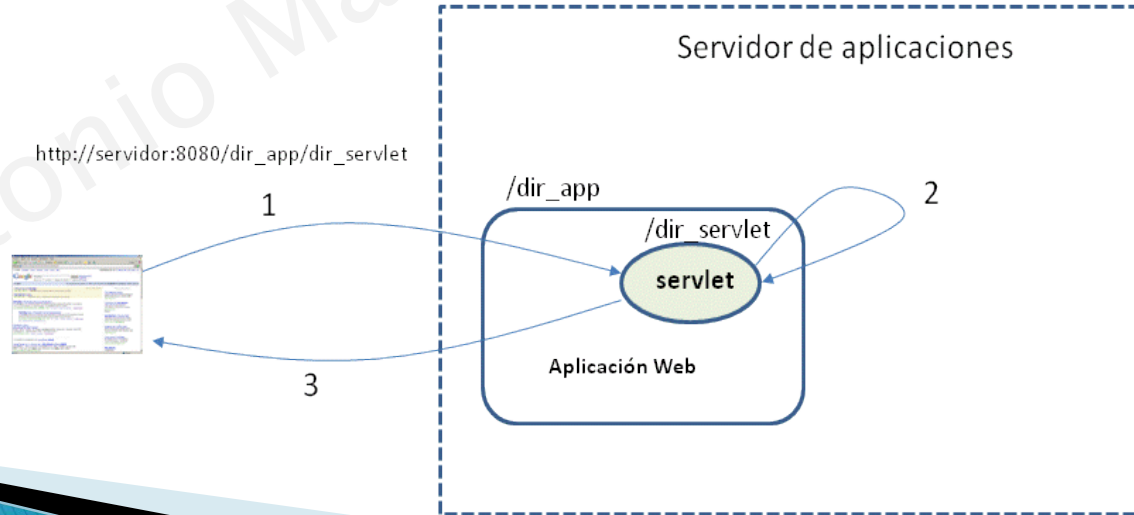


# Programación con servlets

# ¿Qué es un servlet?

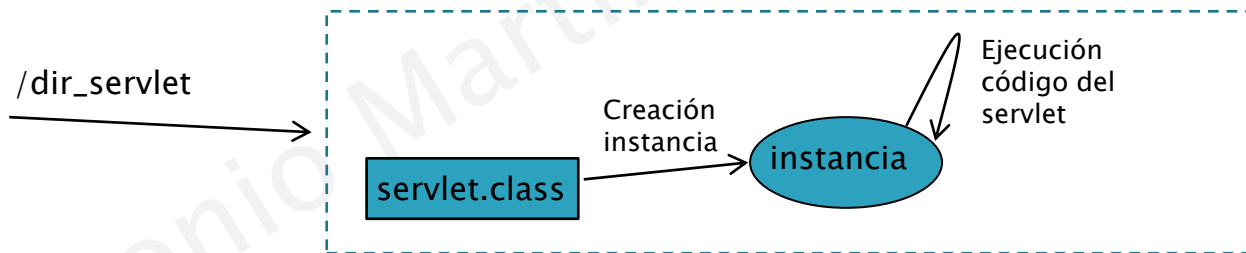
- Objeto que forma parte de una aplicación Web Java EE, y es ejecutado cuando se recibe una petición del mismo.
- Un servlet tiene asociada una dirección dentro de su aplicación



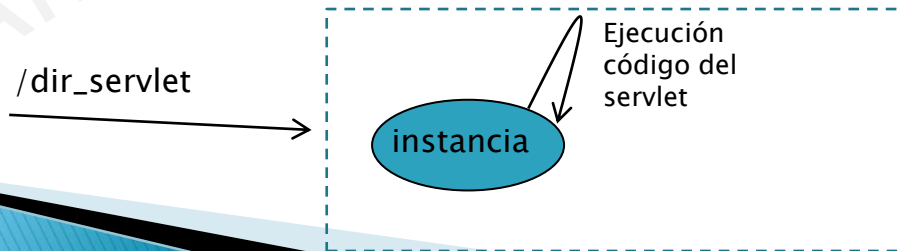
# Ciclo de vida

- Un servlet es una clase, de la que se crea una instancia la primera vez que el servlet es solicitado.

1ª petición



2ª y restantes peticiones



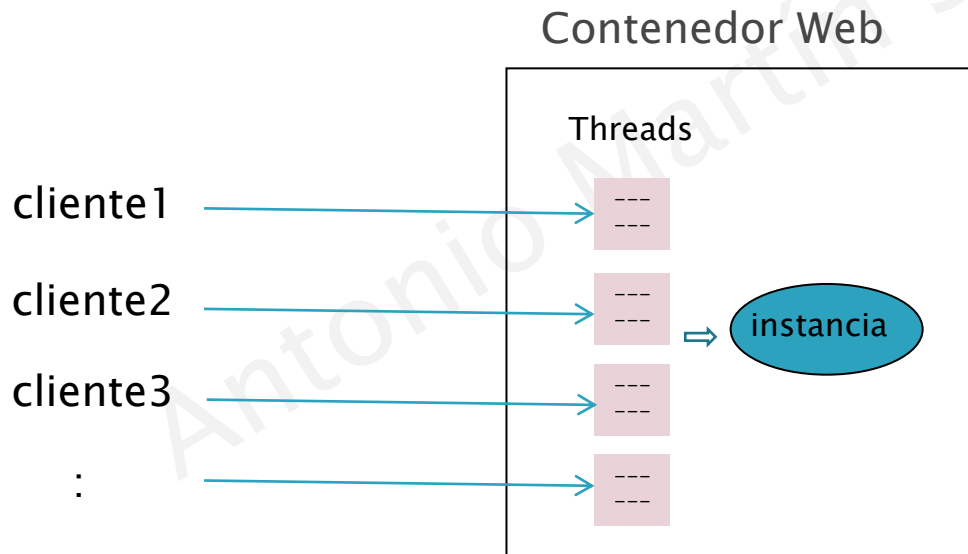
# Métodos del ciclo de vida

➤ Llamados por el contenedor durante la vida del servlet:

- **init.** Es llamado inmediatamente después de crear la instancia
- **service.** Llamado con cada petición realizada al servlet. Es el método donde implementamos la funcionalidad del servlet
- **destroy.** Llamado antes de destruir la instancia

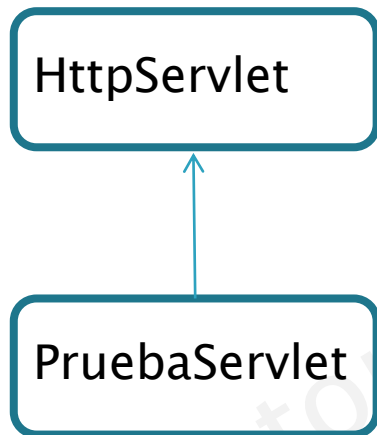
# Concurrencia de peticiones

➤ La primera petición del servlet provoca la creación de una instancia, que atiende a todas las llamadas desde la capa cliente.



➤ Utilizando multitarea, la misma instancia gestiona todas las peticiones.

# La clase servlet



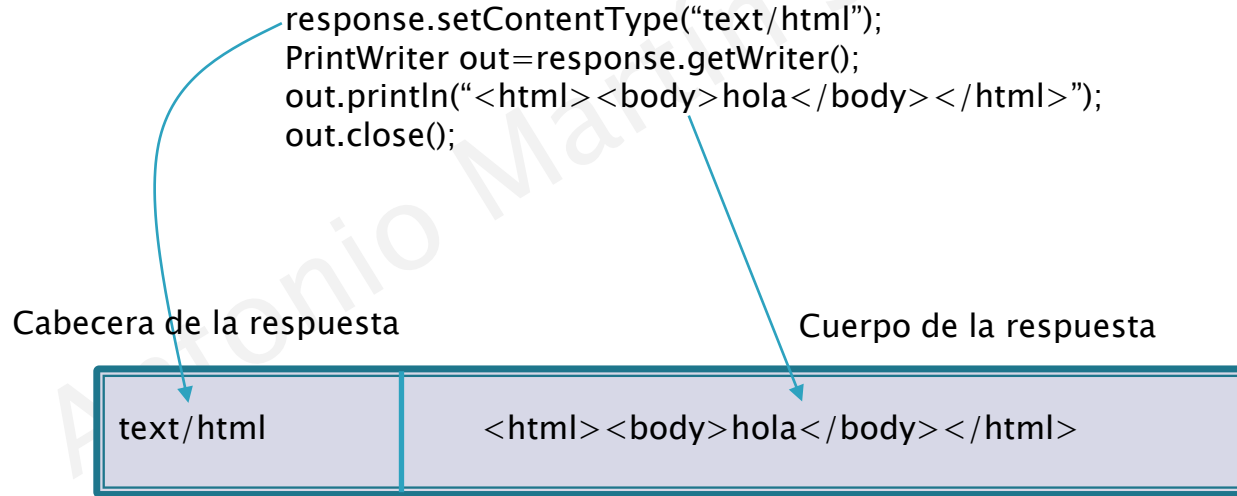
```
@WebServlet("/PruebaServlet")
public class PruebaServlet extends HttpServlet {
    protected void service(HttpServletRequest request,
                           HttpServletResponse response)
        throws ServletException, IOException {
        //código del servlet
    }
}
```

Dirección del servlet

Método ciclo vida

# Generación de respuestas

- El objeto `HttpServletResponse` proporciona un método para obtener el objeto `PrintWriter` asociado a la salida



Respuesta HTTP

# Recogida de parámetros desde un servlet





# Formulario HTML

➤ Recoge los datos de usuario a través de controles gráficos.

➤ Se genera con la etiqueta `<form>`

Dirección asociada al  
servlet

Método de envío HTTP

`<form action="dir_servlet" method="POST">`

`:`

`</form>`

# Controles gráficos HTML

- Se generan con etiquetas especiales: input, select, textarea.
- Mediante atributo *name* se indica el nombre del parámetro que será enviado

```
<form action="dir_servlet" method="POST">  
  Código:<input type="text" name="codigo"/><br/>  
  Unidades:<input type="number" name="unidades"/><br/>  
  <input type="submit" value="Enviar"/>  
</form>
```



Código: AB3

Unidades: 4

Enviar



	cabecera	cuerpo
POST		codigo=AB3&unidades=4
GET	dir_servlet?codigo=AB3&unidades=4	

# Recuperación de parámetros

➤ La interfaz `HttpServletRequest` dispone de los métodos:

- `String getParameter(String name)`. Devuelve el valor del parámetro a partir del nombre
- `String[] getParameterValues(String name)`. Devuelve un array con todos los valores de un parámetro.

Tanto si van en el cuerpo como en la URL, los parámetros se recuperan de la misma manera

```
:
protected void service(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
    String cod=request.getParameter("codigo");
    int unit=Integer.parseInt(request.getParameter("unidades"));
}
:
}
```

# Parámetros en URL

- Se pueden enviar parámetros a un servlet como parte de la dirección del mismo:

```
<a href="dir_servlet?param1=valor1&param2=valor2" >Entrar</a>
```

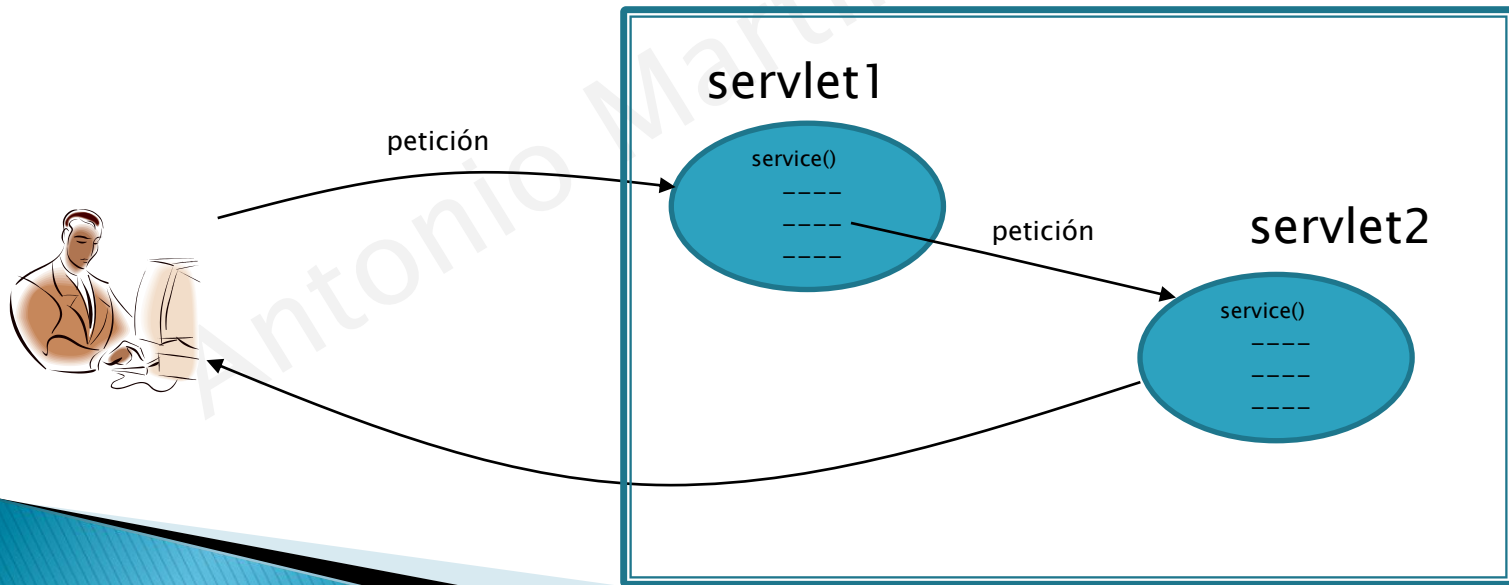
- Se recogen de la misma forma que los parámetros del formulario

```
String dato=request.getParameter("param1");
```

# Transferencia de peticiones

# Concepto de transferencia

- Cuando un servlet recibe una petición, durante el procesamiento de la misma puede transferir el control a otro servlet o componente de la misma aplicación



# Proceso


1. En servlet origen, obtenemos objeto `RequestDispatcher` asociado a servlet destino:

```
RequestDispatcher dispatch=request.getRequestDispatcher("servlet2");
```

2. Transferimos la petición mediante los métodos `forward()` o `include()` de `RequestDispatcher`

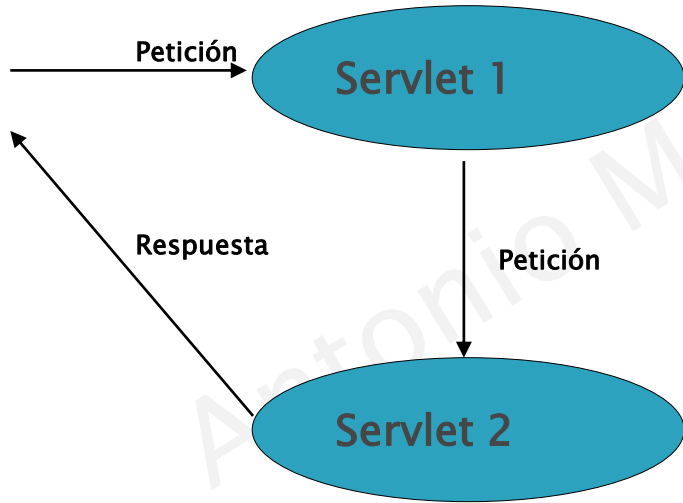
```
dispatch.forward(request,response);
```

Pasamos al servlet destino los  
objeto request y response

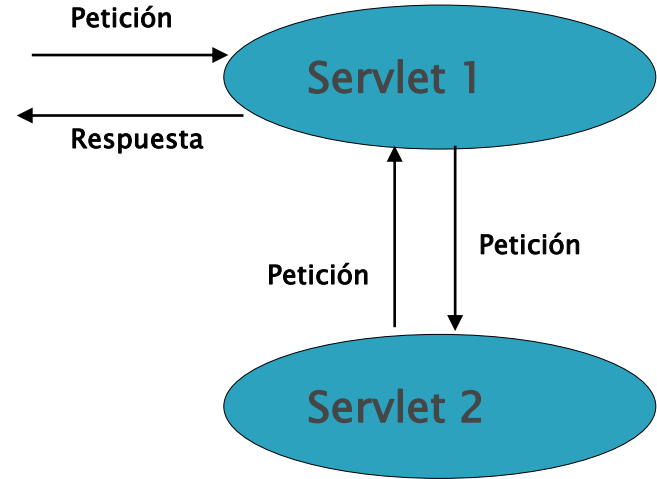


# Métodos forward e include

forward



include

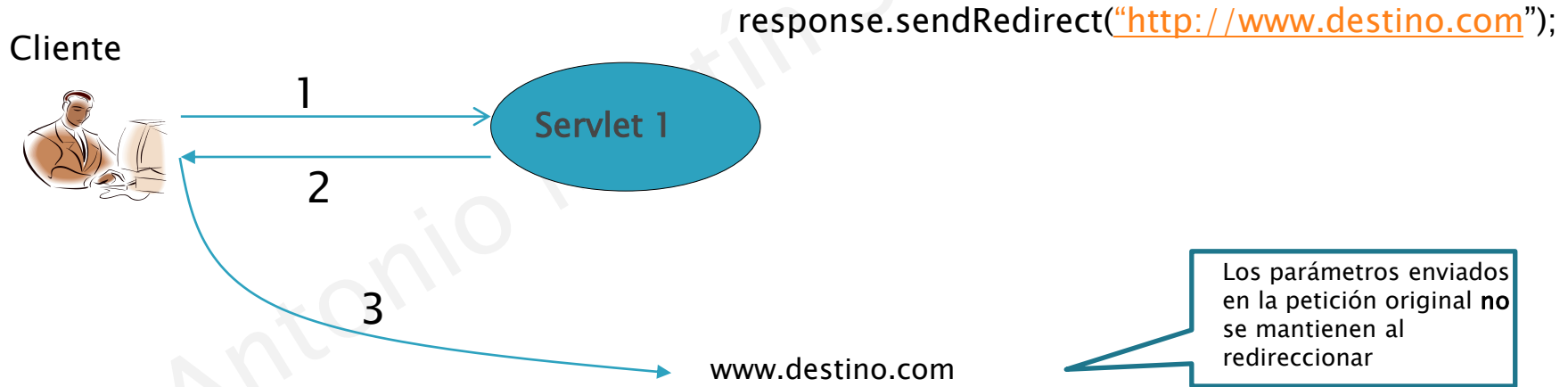


Los parámetros enviados en la petición están accesibles para ambos servlets



# Redireccionamiento

➤ Utilizado para transferir el control de la petición a otra aplicación:



# Mantenimiento de datos en la aplicación

# Mantenimiento de datos

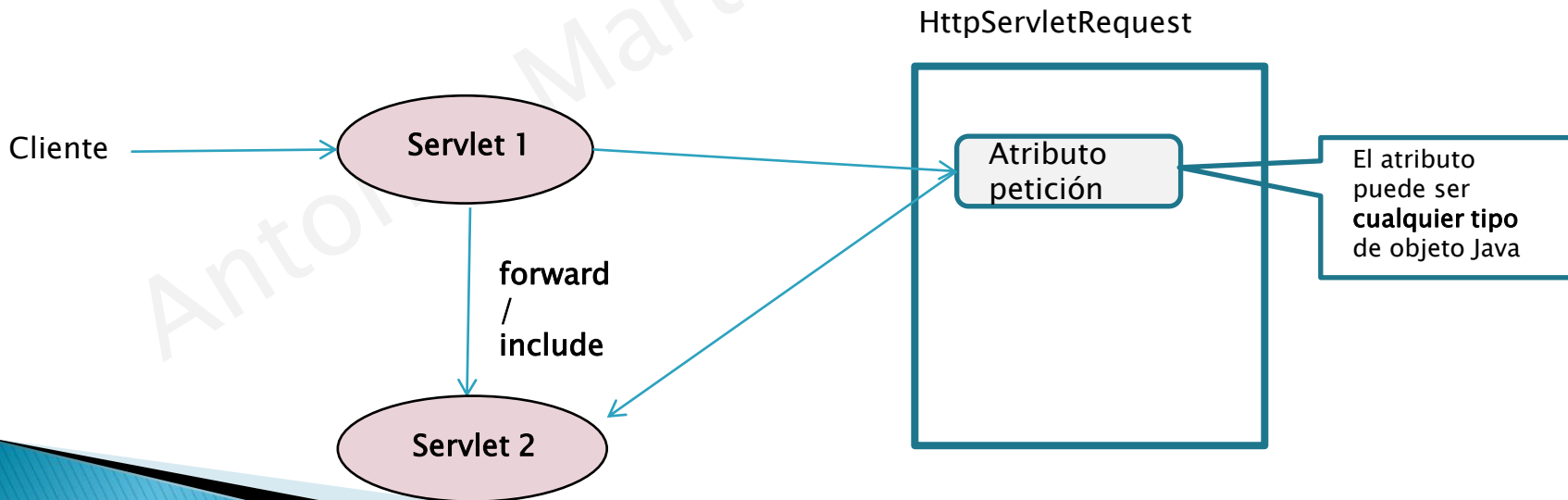
## ➤ Compartir datos entre los componentes de una aplicación:

- Atributos de petición
- Atributos de sesión
- Atributos de aplicación
- Cookies

# Atributos de petición

# Definición

- Permite almacenar datos que son compartidos entre todos los servlets que se ejecutan en la misma petición
- Se almacenan en el objeto `HttpServletRequest`



# Acceso a atributos

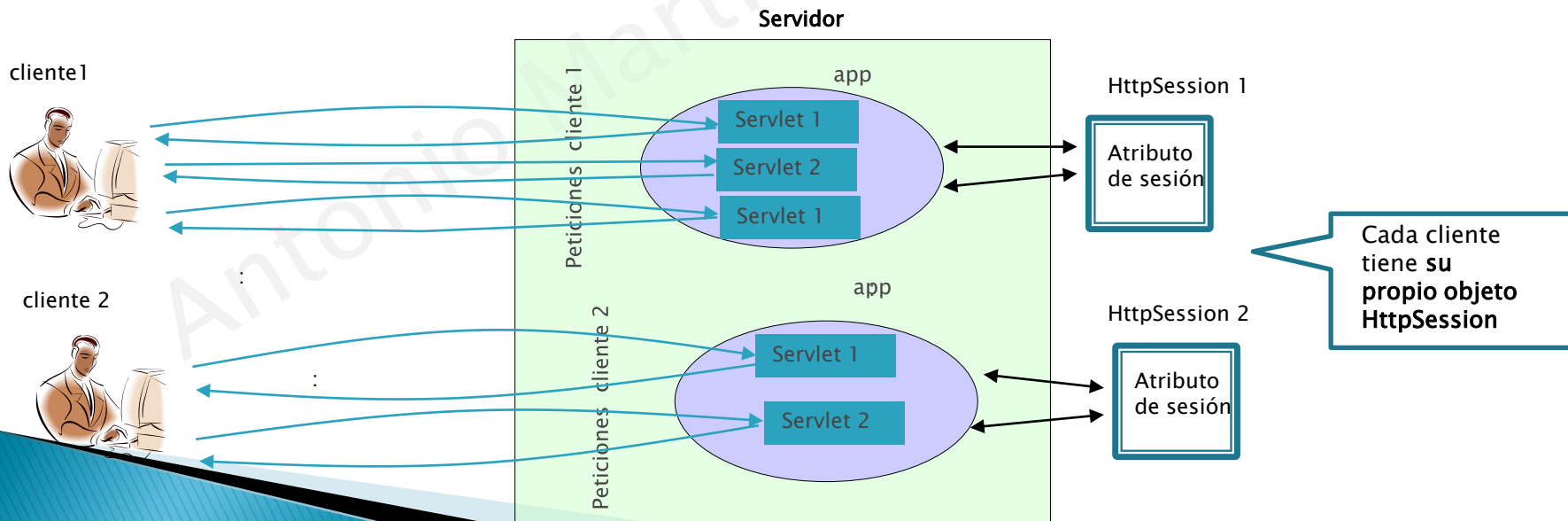
➤ Para establecer y recuperar atributos de petición, se emplean los siguientes métodos de `HttpServletRequest`:

- `void setAttribute(String nombre, Object value)`. Almacena un atributo con el nombre especificado en el primer parámetro y cuyo valor se indica en el segundo
- `Object getAttribute(String nombre)`. Devuelve el valor del atributo cuyo nombre se indica. Si no existe, devuelve null

# Atributos de sesión

# Características

- Permite almacenar datos que son compartidos entre todos los componentes de la aplicación durante la sesión de usuario
- Se almacenan en el objeto HttpSession





# Acceso a atributos

- Se debe obtener el objeto `HttpSession` del usuario mediante el método *`getSession()`* de `HttpServletRequest`:

```
//Si la sesión de usuario ya existe, se  
//recupera, sino se crea una nueva  
HttpSession s=request.getSession();
```

- Para establecer y recuperar atributos de petición, utilizaremos los métodos *`setAttribute()`* y *`getAttribute()`* de `HttpSession`.

# Control de sesiones

## ➤ Tiempo máximo de inactividad de una sesión:

```
<session-config>  
    <session-timeout>10</session-timeout>  
</session-config>
```

web.xml

## ➤ Eliminar atributos de sesión:

- método `removeAttribute(String nombre)` de `HttpSession`

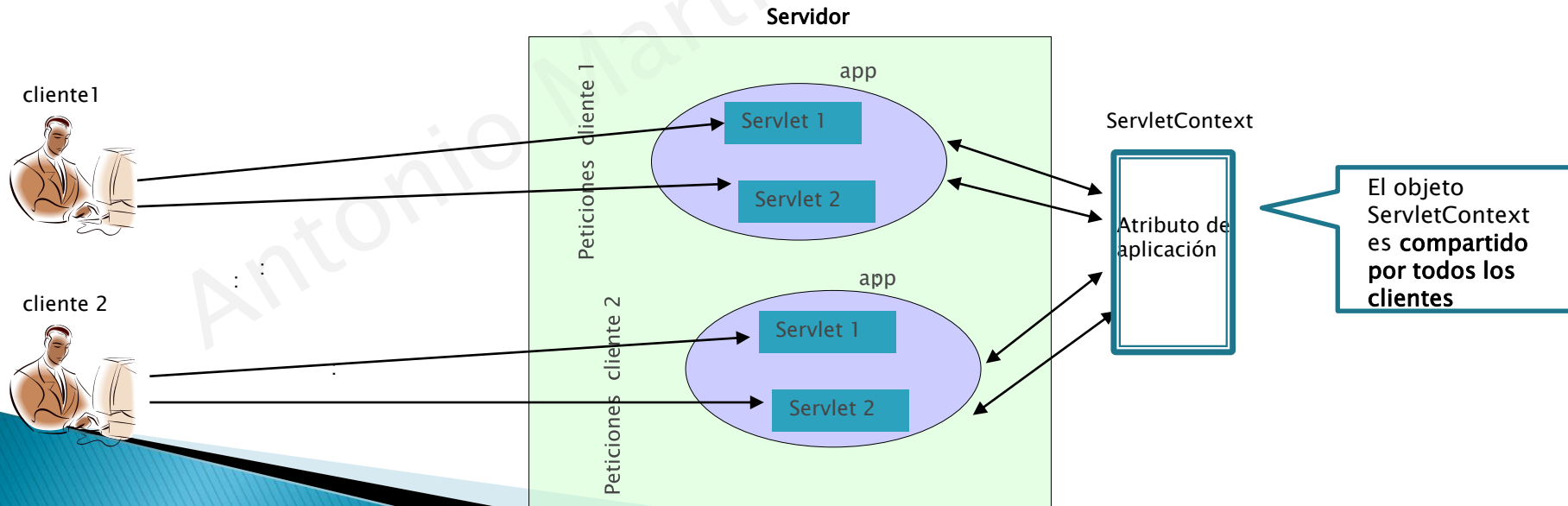
## ➤ Invalidar una sesión:

- método `invalidate()` de `HttpSession`

# Atributos de aplicación

# Características

- Permite almacenar datos que son compartidos entre todos los usuarios de la aplicación
- Se almacenan en el objeto ServletContext



# Acceso a atributos

- Se debe obtener el objeto `ServletContext` mediante el método *`getServletContext()`* de `HttpServletRequest`:

```
//Obtiene el objeto ServletContext  
//que es el mismo para todos los usuarios  
ServletContext context=request.getServletContext();
```

- Para establecer y recuperar atributos de petición, utilizaremos los métodos *`setAttribute()`* y *`getAttribute()`* de `ServletContext`.

# Control de atributos de aplicación

- No hay un timeout. El objeto Servlet Context se destruye al detener la aplicación
- Eliminar atributos de aplicación:
  - método `removeAttribute(String nombre)` de `HttpSession`

# Cookies

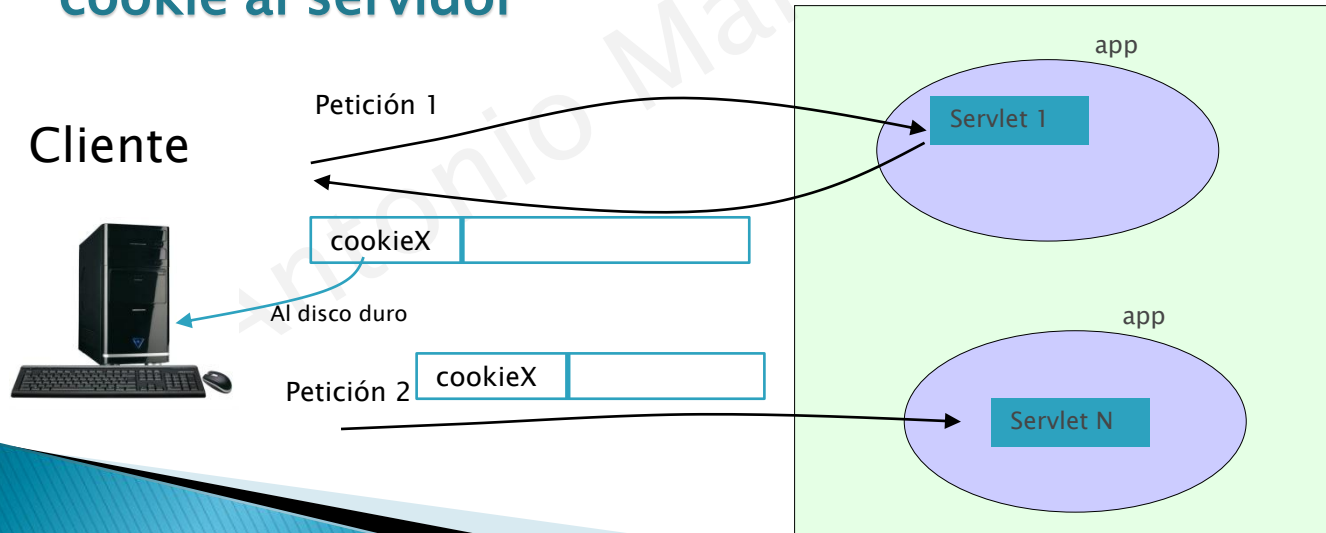
# Características

- Datos creados por la aplicación y que se almacenan en el disco duro del cliente
- Solo admite cadenas de caracteres
- El tiempo de vida es limitado
- Se manejan a través de `javax.servlet.http.Cookie`



# Funcionamiento

- La cookie es creada por el servidor y enviada al cliente en la cabecera de la respuesta.
- En posteriores conexiones, el cliente envía de nuevo la cookie al servidor



# Creación de una cookie

## 1. Creación del objeto Cookie:

```
Cookie ck=new Cookie("codigo","E27");
```

## 2. Establecimiento de periodo de vida:

```
ck.setMaxAge(18000);
```

## 3. Añadir cookie a la respuesta:

```
response.addCookie(ck);
```

# Recuperación de una cookie

## 1. Obtención de array de cookies:

```
Cookie[] cookies=request.getCookies();
```

## 2. Recorrido y búsqueda:

```
for(Cookie ck:cookies){  
    if(ck.getName().equals("codigo")){  
        String valor=ck.getValue();  
    }  
}
```

# Eventos de aplicación

# Características

- Sucesos que tienen lugar durante la vida de la aplicación, como llegada de petición, inicio de sesión, fin de sesión, etc.
- A través de las clases de escucha, se pueden implementar acciones de respuesta a estos eventos, como inicializar atributos.
- Los métodos de respuesta están definidos en las interfaces de escucha

# Interfaces de escucha

## ➤ javax.servlet:

- `ServletContextListener`. Define métodos para responder a eventos inicio y fin de aplicación
- `ServletRequestListener`. Define métodos para responder a eventos llegada de petición y fin de petición

## ➤ javax.servlet.http:

- `HttpSessionListener`. Define métodos para responder a eventos inicio y fin de sesión

# Implementación de un manejador

- Se crea una clase que implemente la interfaz de escucha y se registra con @WebListener:

```
@WebListener
public class EscuchadorSesion implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent se) {

    }
}
```

Registro del escuchador

Clase escuchadora

Proporciona acceso a información de sesión