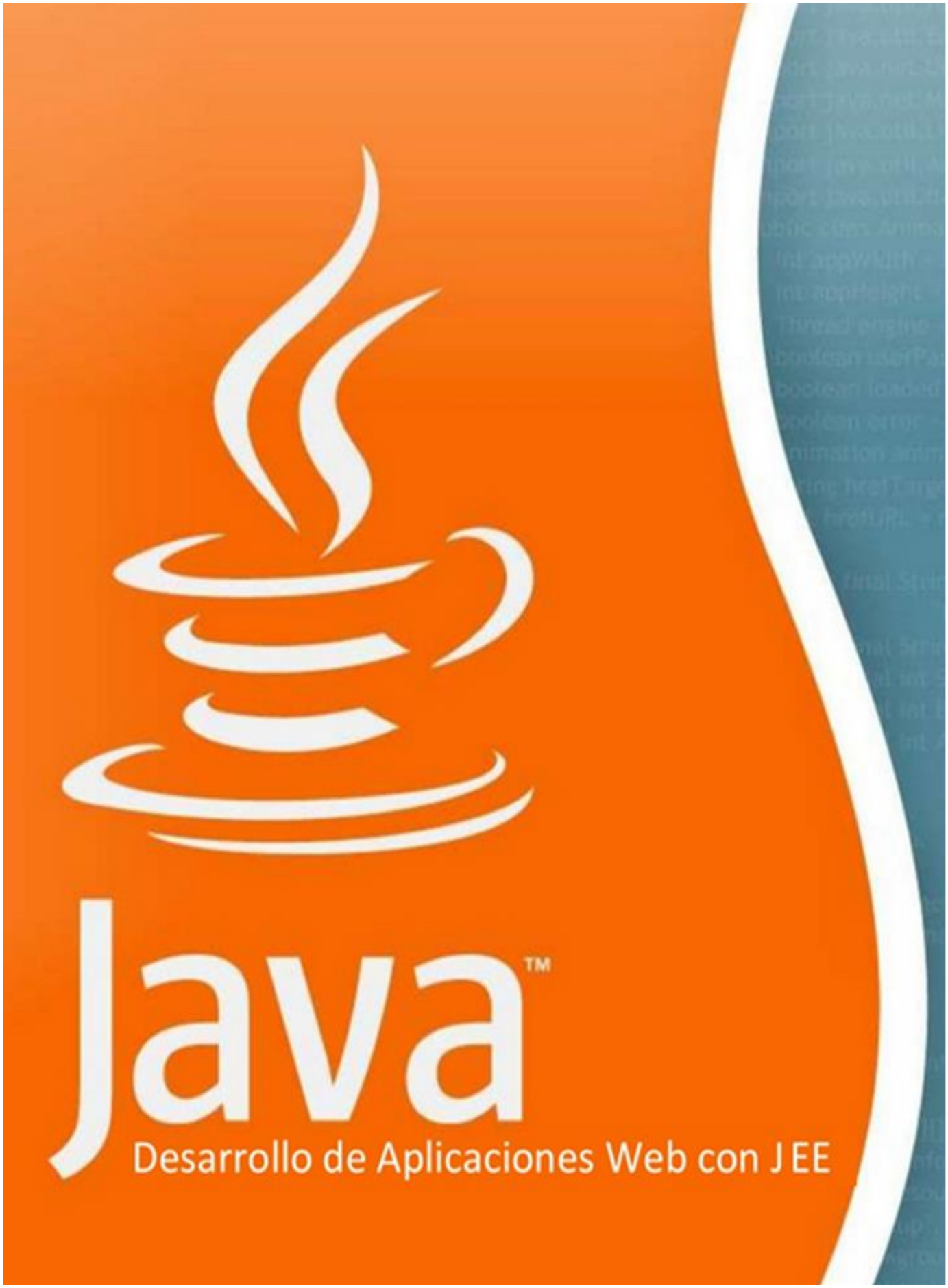

Java™
Desarrollo de Aplicaciones Web con J EE

PARTE IV

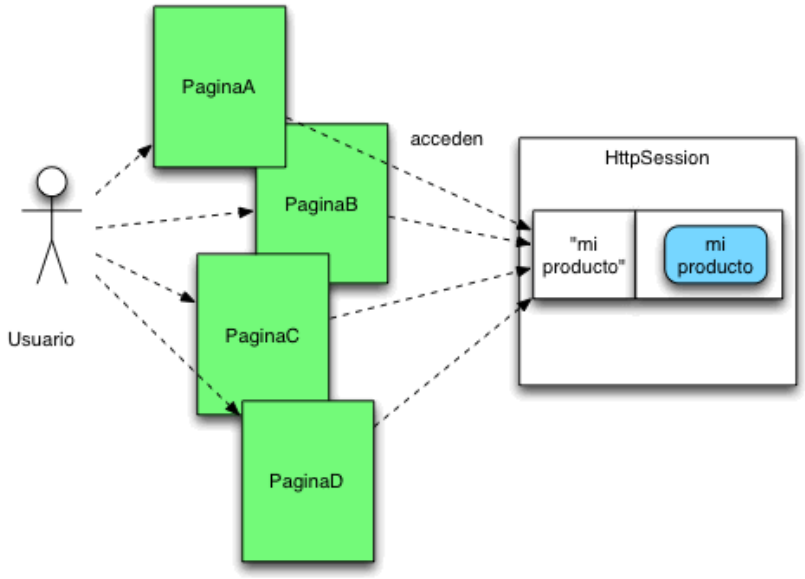
- **SESIONES Y COOKIES**

4.1 Sesiones

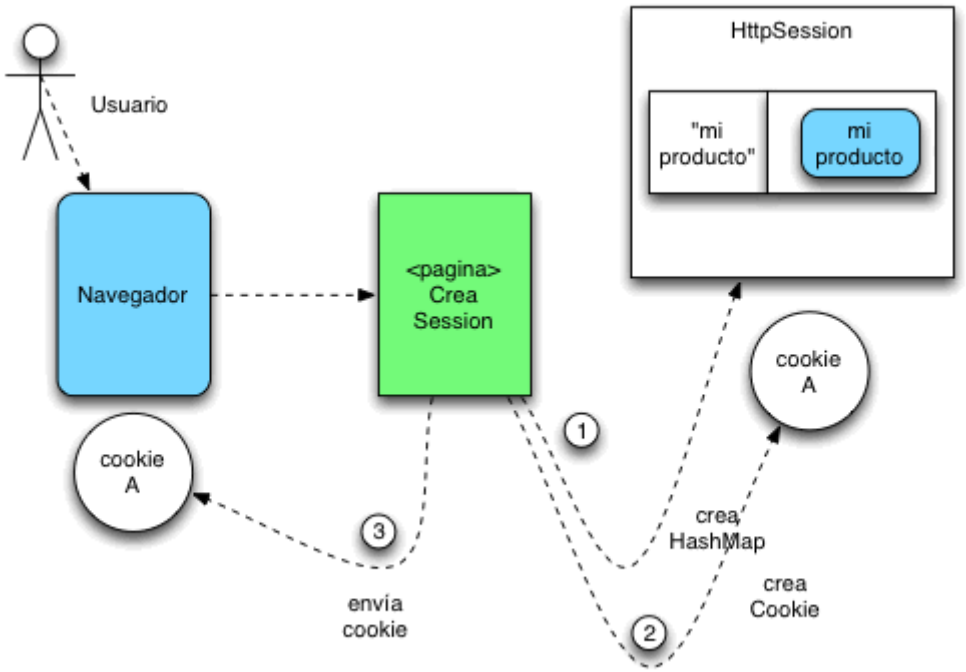
En un gran porcentaje de aplicaciones existe un **proceso de autenticación/validación de usuario**. Una vez el usuario ha accedido al sistema con su nombre de usuario y contraseña, en función de sus permisos se le permiten unas operaciones u otras.

- ✓ **Problema:** El protocolo HTTP no es capaz de almacenar el estado en el que se encuentra, cada petición es independiente.
- ✓ Sin embargo, en las aplicaciones web que conocemos, el usuario no inserta su clave y nombre de usuario cada vez que realiza una operación, sino que la realiza una sola vez al inicio.
- ✓ Los servidores de aplicaciones suplen esa carencia permitiendo que almacenemos variables en un entorno de ejecución de nuestra aplicación web denominado **sesión** (espacio de tiempo ocupado por la actividad del usuario desde que accede al sistema hasta que lo abandona).

Los servlets son capaces de conectarse a una determinada sesión en curso o crear una nueva y añadir, modificar o borrar atributos de la sesión. **Estas sesiones permanecen activas mientras no se destruyan o la aplicación no se encuentre inactiva durante un periodo configurable de tiempo.**



El funcionamiento del sistema de sesiones es relativamente sencillo. **Cada vez que un usuario crea una session (clase HttpSession) accediendo a una página se crea un objeto a nivel de Servidor con una estructura similar a un HashMap vacío que nos permite almacenar la información que necesitamos relativa a este usuario.** Realizado este primer paso se envía al navegador del usuario una Cookie que sirve para identificarle y asociarle el HashMap que se acaba de construir para que pueda almacenar información en él. Este HashMap puede ser accedido desde cualquier otra página permitiéndonos compartir información.



Cada petición del cliente (request) tiene asociada una sesión, para acceder a ella se realiza una llamada al método ***getSession()*** de los objetos HTTPServletRequest. Cuando se llama al método getSession() se pasa un argumento a true, de forma que si la sesión no existe se crea. Posteriormente podemos ver cómo podemos acceder y modificar atributos de la sesión (**setAttribute()/getAttribute()**). También podemos consultar algunos atributos de la sesión, como el identificador que asigna a la sesión el servidor de aplicaciones.

Para que el servidor de aplicaciones Web (Tomcat) pueda saber qué objeto sesión corresponde a cada petición HTTP se emplean dos mecanismos principalmente:

- ✓ **El navegador del cliente no acepta cookies:** cuando el navegador del usuario no soporta cookies el usuario pulsa sobre una URL no re-escrita se pierde la sesión de usuario. El servlet contactado a través de ese enlace crea una nueva sesión, pero la nueva sesión no tiene datos asociados con la sesión anterior. Una vez que un servlet pierde los datos de una sesión, los datos se pierden para todos los servlets que comparten la sesión. Debemos **utilizar la re-escritura de URLs consistentemente** para que nuestro servlet soporte clientes que no soportan o aceptan cookies.

  Cuando se utiliza la reescritura de URL se llama a los métodos que, cuando es necesario, incluyen el ID de sesión en un enlace. Debemos llamar a esos métodos por cada enlace en la respuesta del servlet. **El método que asocia un ID de sesión con una URL es HttpServletResponse.encodeURL.** Si el usuario pulsa sobre un enlace con una URL re-escrita, el servlet reconoce y extrae el ID de sesión. Luego el método getSession utiliza el ID de sesión para obtener el objeto HttpSession del usuario.

  response.sendRedirect(response.encodeURL("address"));

- ✓ **El navegador del cliente acepta cookies:** se envía en una cookie al navegador al inicio de la sesión el identificador de la sesión (jsessionid)

Los métodos del objeto sesión son:

Método	Descripción
public void setAttribute(string nombre, Object valor)	Guardar valor en la sesión (a partir de versión 2.2 de API de los servlets)
public Object getAttribute(String nombre)	Obtener valor de la sesion
public void removeValue(String nombre)	Elimina los valores asociados al nombre pasado como argumento.
public void removeAttribute(String nombre)	Elimina los valores asociados al nombre pasado como argumento (a partir de versión 2.2 de API de los servlets
public Enumeration getAttributeNames()	Devuelve los nombres de las claves asociadas a la sesion (a partir de versión 2.2 de API de los servlets)
public boolean isNew()	Devuelve true si el cliente no se encontraba en una sesión. Devuelve false si la sesion ya existía.
public void invalidate()	Invalida la sesión y la desliga de todos los objetos asociados con ella.
public String getId()	Devuelve el identificador de la sesión activa
public long getCreationTime()	Devuelve cuando la sesión fue creada
public long getLastAccessedTime ()	Devuelve cunado se accedió por última vez a la sesión
public int getMaxInactiveInterval()	El tiempo máximo que una sesión puede permanecer inactiva

4.1.1 Ejemplo8

Por ejemplo, podemos desarrollar una aplicación web que, mientras esté activa la sesión, cuente el número de veces que accedemos a una página:

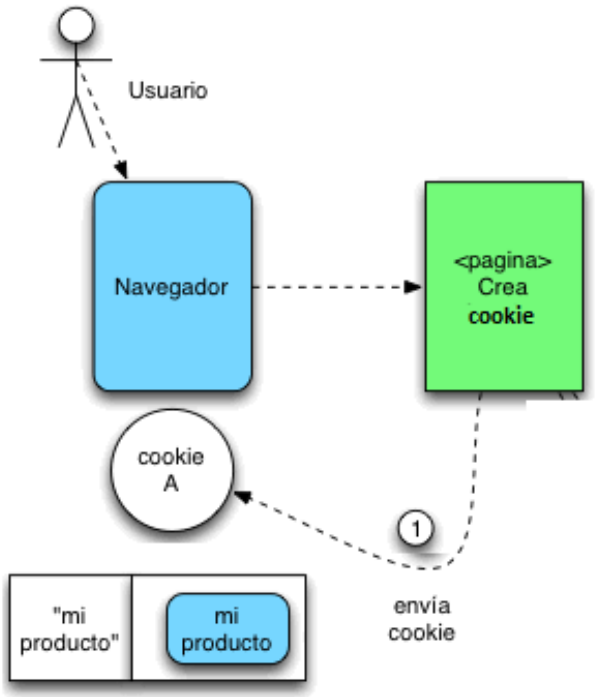
```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/plain");
    int c=0;
    // Creamos la sesión si no existe
    HttpSession sesion = request.getSession(true);
    if (sesion.getAttribute("cont")==null) {
        response.getWriter().append("HOLA DAM \n");
        //Reescritura de URL si el navegador no acepta cookies
        //response.sendRedirect(response.encodeRedirectURL("Ejemplo8"));
    } else { // Recoger de la sesión el valor del contador
        response.getWriter().append("HOLA DE NUEVO DAM \n");
        c = (int) sesion.getAttribute("cont");
    }
    // Incrementar el contador guardandolo en la sesión
    sesion.setAttribute("cont", c + 1);
    response.getWriter().append("Recargas:" + c + " \n");
    response.getWriter().append("La sesión fue creada" + new Date(sesion.getCreationTime()) + " \n");
    response.getWriter().append("La sesión tiene como identificador:" + sesion.getId() + " \n");
    response.getWriter().append("Se accedió a la sesión por última vez:" + new Date(sesion.getLastAccessedTime()) + " \n");
    response.getWriter().append("Una sesión puede permanecer inactiva:" + sesion.getMaxInactiveInterval() + " \n");
    response.getWriter().close();

}
```

4.2 Cookies

Otra posibilidad que ofrecen los servidores de aplicaciones para almacenar el estado de una aplicación web es el uso de cookies. **Las cookies son pequeños ficheros de texto que almacenan información en el lado del cliente.** Esta es la mayor diferencia con las sesiones, la información está en el ordenador del cliente, no en el lado del servidor. Esta técnica permite al cliente ciertas ventajas: Identificar a un usuario que ha realizado una compra online; Evitar la autentificación constantemente; Personalizar un sitio; Publicidad personalizada: (Un motor de búsqueda, como Google, puede mantener una pista de las preferencias de un usuario a lo largo del tiempo).

Respecto a la seguridad, las cookies no se "interpretan" o ejecutan en un sistema, por lo que no se puede insertar virus en ellas para comprometer la integridad del sistema. Los navegadores, generalmente sólo **aceptan 20 cookies por sitio**, además **un navegador nunca poseerá más de 300 cookies en total**, y **cada cookie** está limitada a **un tamaño de 4 KB**, así que no se pueden usar para llenar el disco duro de un cliente. Sin embargo, aunque las cookies no presentan una amenaza a la seguridad, si **pueden presentar una seria amenaza a la privacidad**. Pero más que culpa de la cookie en sí, los problemas de privacidad radican en programadores con prácticas poco seguras.



Las cookies están formadas por el nombre de la cookie y el valor asociado a dicha cookie. Además, una aplicación puede trabajar con todas las cookies que necesite. En Java disponemos de la **clase Cookie** para el trabajo con cookies.

El concepto de cookie es similar al de un diccionario (conjunto de pares <clave,valor>). La creación de nuevas cookies requiere tres pasos:

- ✓ Creación de un nuevo objeto cookie

Cookie c= new Cookie (nombre,valor);

- ✓ Fijar los atributos de la cookie (ej.- caducidad, versión, path, dominio)

c.setMaxAge(80);/ c.setMaxAge(-1);

- ✓ Añadir la cookie al objeto respuesta (la cookie se remite al navegador)

response.addCookie(c)

Normalmente, el navegador únicamente devuelve las cookies al servidor que los envió, pero podemos utilizar **setDomain()** para indicar al navegador que remita los cookies a otros hosts del mismo dominio o también recuperar el valor del dominio getDomain(), por ejemplo para enviar la cookie c a todo host del Gregorio Fernández.

c.setDomain("gregoriofer.es")

Podemos utilizar cookies de sesión (desaparecen al cerrar el navegador) o persistentes (sobreviven el tiempo de vida indicado, o hasta que el usuario los borra de forma explícita). **setMaxAge** permite especificar la caducidad en segundos. Un valor negativo (o ningún valor) indica que se trata de una cookie de sesión, 0 indica al navegador que borre la cookie, y un valor positivo indica tiempo de vida en segundos.

Por defecto, las cookies se envían con independencia de acceder o no a través de SSL. Con **setSecure(true)** indicamos que sólo se devuelvan las cookies al acceder a través de una conexión cifrada (SSL).

Para leer cookies utilizamos el método **getCookies definido en HttpServletRequest.** El método getCookies() devuelve un vector de objetos cookie (todos las cookies enviados por el navegador). Si no hay cookies se obtiene un vector de longitud 0. Para acceder a cada cookie individual iteramos sobre el vector. **GetName y getValue** resultan muy útiles para la lectura de cookies.

4.2.1 Ejemplo9

Creamos un servlet simple que registra el número de 'visitas' al mismo. Deseamos mostrar el número de visitas para cada usuario, para lo cual creamos una cookie correspondiente al contador, e incrementamos la cookie en cada visita. El esqueleto de la aplicación queda como sigue:

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

    boolean bandera = false;
    int n = 0;
    response.setContentType("text/plain");
    // Se recupera las cookies del navegador
    Cookie[] cs = request.getCookies();
    if (cs != null) {
        for (int i = 0; i < cs.length && !bandera; i++) {
            // Busco la cookie "cont" en el vector
            if (cs[i].getName().equals("cont")) {
                n=Integer.parseInt(cs[i].getValue())+1;
                bandera = true;
            }
        }
    }
    // Creo la cookie
    Cookie c = new Cookie("cont", String.valueOf(n));
    c.setMaxAge(60 * 60 * 365);// caduca al cabo de un año
    // Se envía la cookie como respuesta al navegador
    response.addCookie(c);
    response.getWriter().append("Número de visitas:" + n);
    response.getWriter().close();


}
```

Por defecto, una cookie tiene una duración igual al tiempo de la sesión. Es decir, mientras el usuario está conectado a la aplicación web. En este caso la única diferencia respecto a las sesiones es donde se almacena la información, y normalmente, los desarrolladores optan por las sesiones. Sin embargo, en este ejemplo se crea el método **crearCookie()** que crea cookies con un tiempo de duración independiente del tiempo de sesión utilizando el método **setMaxAge()**. En este caso un año. Esto hace que, durante un año, el contador funcione independientemente de si el usuario se conectó o no, o incluso de si el servidor de aplicaciones se reinicia o no.

Como podemos observar en el ejemplo, **las cookies se solicitan al cliente mediante el objeto HttpServletRequest.** También es interesante señalar que **las cookies, una vez recibidas y procesadas deben añadirse a la respuesta al cliente**. El navegador será el encargado de almacenar la cookie en la carpeta asignada para esto en el navegador.

También es importante tener en cuenta que para que las cookies funcionen tienen que estar habilitadas en el navegador.