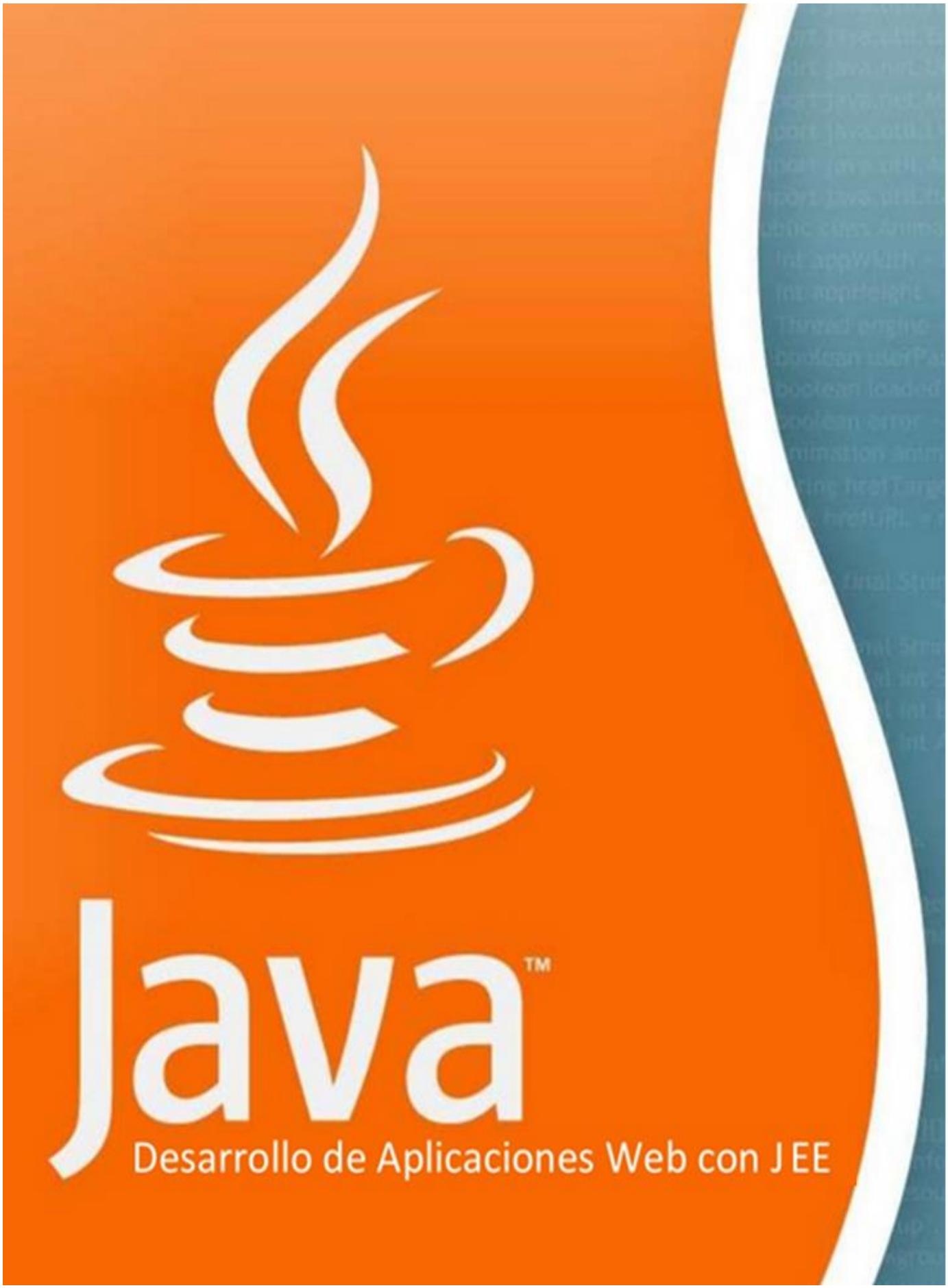

Java™
Desarrollo de Aplicaciones Web con J EE

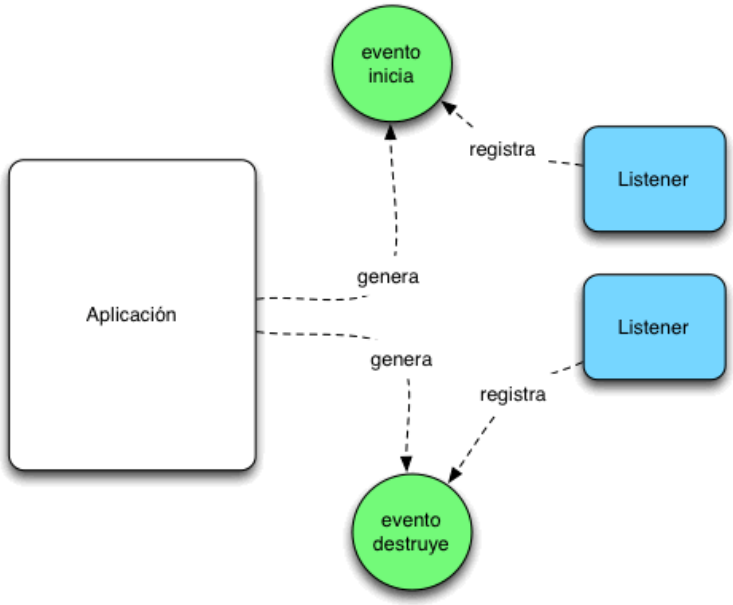
PARTE VII

EVENTOS Y FILTROS EN SERVLETS

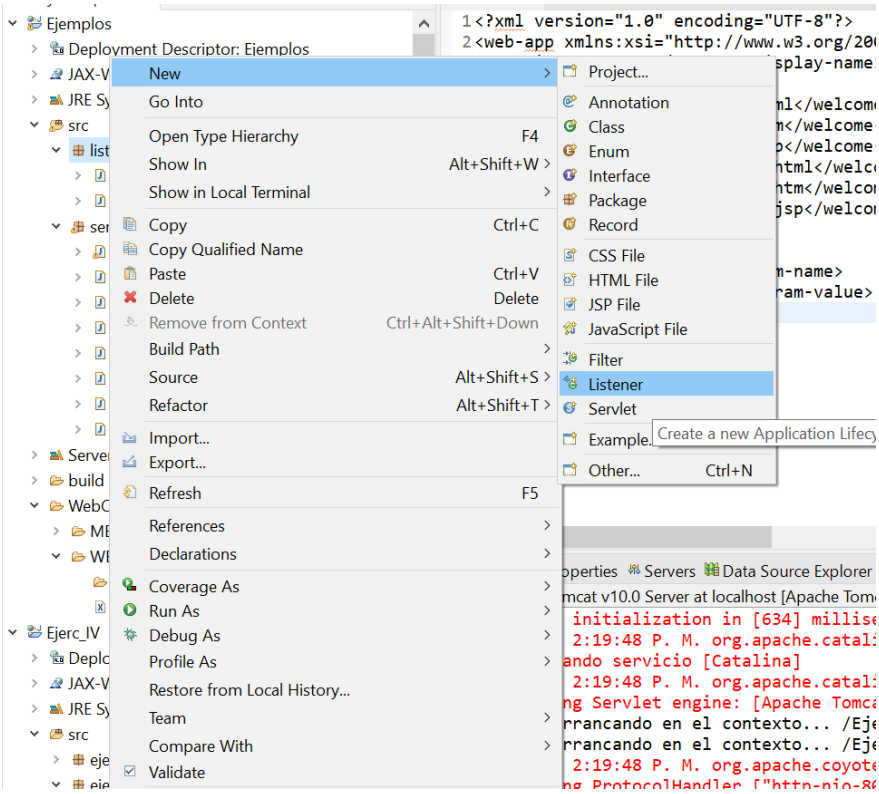
7. Eventos en servlets

En ocasiones, antes de arrancar nuestra aplicación web queremos realizar ciertas **tareas de inicialización**. Por ejemplo, quizás tengamos una lista de productos almacenado en la base de datos. Salvo que estemos tratando con cantidades muy grandes de productos, podría tener sentido cargar todos esos productos en memoria (por ejemplo, almacenándolos en el contexto de la aplicación) para que cuando consultemos la lista las operaciones vayan más rápido. O abrir un archivo con el cual vamos a trabajar en la aplicación web y cuando la aplicación web deje de ejecutarse cerrar el archivo. También resulta interesante a veces realizar algún tipo de acción cada vez que se crea una sesión del usuario, o simplemente cada vez que un determinado Servlet es usado. En este punto vamos a ver **cómo responder a este tipo de eventos del contenedor y a interceptar peticiones que llegan al servidor**. Los tipos de listener con los que podemos trabajar son:

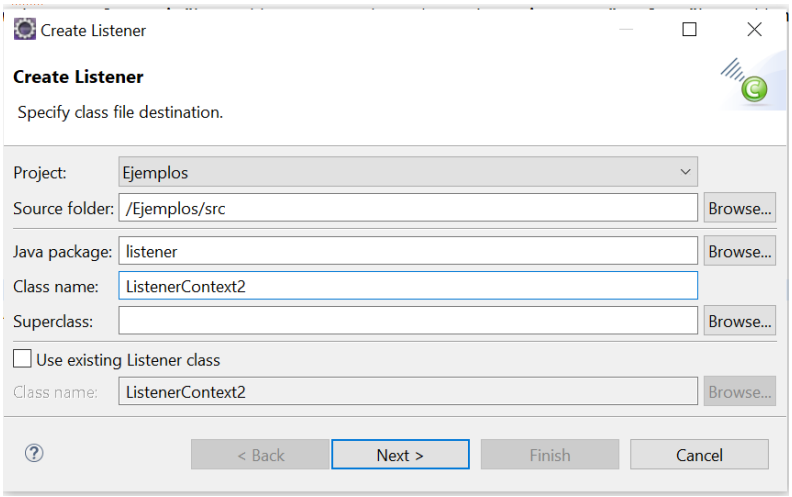
- ✓ **ServletContextListener**: Listener que se encarga de gestionar los eventos generales de la aplicación como son **arranque y parada de la aplicación**.
- ✓ **HttpSessionListener**: Listener que se encarga de gestionar los propios eventos de la sesión como **creación, invalidación y destrucción de sesiones**.
- ✓ **ServletRequestListener**: Listener que se encarga de los eventos de **creación y destrucción de peticiones.**



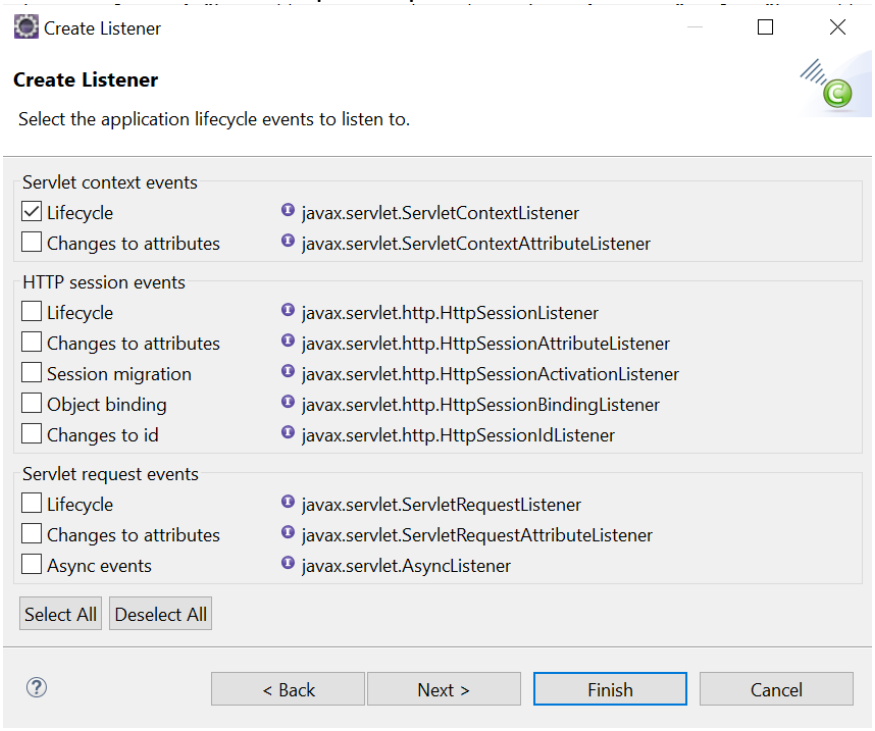
Para crear un Listener desde Eclipse en un proyecto creado se selecciona la opción de menú **New->Listener.**



A continuación, se indica el nombre del evento:



Por último se eligen los eventos desde los que se quiere escuchar:



7.1 ServletContextListener: arranque y parada de la aplicación

La interfaz **ServletContextListener** nos permite crear clases que pueden responder ante **eventos de arranque y parada de la aplicación web.** La interfaz ServletContextListener define dos métodos:

- ✓ **public void contextInitialized(ServletContextEvent sce):** este método es **invocado cuando la aplicación web es arrancada.** Este método será ejecutado **antes de que cualquier Servlet sea inicializado** (y por tanto, antes de que puedan responder a una petición de un usuario).
- ✓ **public void contextDestroyed(ServletContextEvent sce):** este método es **invocado cuando la aplicación va a ser detenida.** Cuando este método es invocado, **el método destroy() de todos los Servlets del contenedor ha sido ejecutado**.

Una vez hayamos creado una clase que implemente esta interfaz, tenemos que registrarla en el servidor de aplicaciones. Esto podemos hacerlo a través del descriptor de despliegue:

```
<listener>
<listener-class>
  ServletListenerContexto
</listener-class>
</listener>
```

También empleando la anotación @WebListener si estamos empleando un contenedor de Servlets que soporte la especificación 3.0 o posterior de los Servlets.

Es posible definir en una aplicación web cuantos listeners de contexto queramos. Si estos son declarados empleando el descriptor de despliegue, serán invocados en el mismo orden en el cual fueron definidos en el descriptor de despliegue cuando la aplicación arranque, y en orden inverso cuando la aplicación se detenga.

7.1.1 Ejemplo12

A continuación, mostramos un ejemplo de clase que implementa la interfaz. Esta clase se limita a mostrar mensajes en la consola cuando el servidor arranca, mostrando el contexto en el cual está arrancando la aplicación y el parámetro de configuración que se le ha pasado. Cuando la aplicación es detenida, mostrará otro mensaje en la consola y volverá a mostrar el contexto de la aplicación que se está deteniendo.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlr
    <display-name>Ejemplos</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <context-param>
        <param-name>bbdd</param-name>
        <param-value>datos</param-value>
    </context-param>
</web-app>
```

```java
public class ListenerContext implements ServletContextListener {

    /**
     * Default constructor.
     */
    public ListenerContext() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @see ServletContextListener#contextInitialized(ServletContextEvent)
     */
    public void contextInitialized(ServletContextEvent sce) {
        // TODO Auto-generated method stub
        System.out.println("Aplicacion arrancando en el contexto... " + sce.getServletContext().getContextPath()
                + " con el siguiente parámetro de configuracion: " + sce.getServletContext().getInitParameter("bbdd"));
    }

    /**
     * @see ServletContextListener#contextDestroyed(ServletContextEvent)
     */
    public void contextDestroyed(ServletContextEvent sce) {
        // TODO Auto-generated method stub
        System.out.println("Aplicacion del contexto " + sce.getServletContext().getContextPath() + " deteniéndose...");
    }
```

7.2 HttpSessionListener: escuchando eventos de sesión

También es posible dentro de una aplicación web escuchar los **eventos relativos a la creación y destrucción de las sesiones de los usuarios.** Para ello debemos implementar la interfaz HttpSessionListener que define los siguientes métodos:

- ✓ **public void sessionCreated(HttpSessionEvent se):** método que será **invocado cada vez que se cree una sesión en la aplicación.** A partir del evento que se le pasa como parámetro es posible obtener dicha sesión.
- ✓ **public void sessionDestroyed(HttpSessionEvent se):** método que será **invocado cada vez que se destruya una sesión en la aplicación.** A partir del evento que se le pasa como parámetro es posible obtener dicha sesión.

Al igual que sucedía con los listeners de contexto, es necesario registrar los listeners de sesión, o bien a través de la anotación @WebListener o en el descriptor de despliegue:

```
<listener>
 <listener-class>
  ServletListenerSesion
 </listener-class>
</listener>
```

Una de las utilidades de escuchar estas sesiones puede ser realizar loggin relativo a la creación y destrucción de sesiones.

7.2.1 Ejemplo13

Cada vez que se crea una sesión, muestra un mensaje en la consola indicando la hora en la cual fue creada la sesión, su identificador y su máximo tiempo de inactividad. Cada vez que se destruye una sesión, muestra otro mensaje en la consola mostrando el identificador de la sesión.

```
public class ListenerSession implements HttpSessionListener {

    /**
     * Default constructor.
     */
    public ListenerSession() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpSessionListener#sessionCreated(HttpSessionEvent)
     */
    public void sessionCreated(HttpSessionEvent se)  {
         // TODO Auto-generated method stub
        HttpSession session = se.getSession();
        System.out.print("A las " + new Date (System.currentTimeMillis()).toString() + " se creo la sesion con ID: " + session.getId()+"\n");
    }

    /**
     * @see HttpSessionListener#sessionDestroyed(HttpSessionEvent)
     */
    public void sessionDestroyed(HttpSessionEvent se)  {
         // TODO Auto-generated method stub
        HttpSession session = se.getSession();
        System.out.println("A las " + new Date (System.currentTimeMillis()).toString() + " se destruyo la sesion con ID: " + session.getId());
    }
```

```
public class Ejemplo13 extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Ejemplo13() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        HttpSession sesion = request.getSession(true);

        response.setContentType("text/plain");
        response.getWriter().append("Sesión creada:"+ sesion.getId());
        sesion.invalidate();
    }
```

También existen otros Listeners como **HttpSessionBindingListener**, que nos permite obtener notificaciones del contenedor cuando un objeto es ligado o desligado a una sesión. **HttpSessionAttributeListener** permite obtener notificaciones cuando hay cambios en los atributos contenidos en una sesión. **HttpSessionActivationListener** permite saber si el contenedor está haciendo pasiva una sesión, es esto es, la está eliminando de la memoria principal y pasándola a un dispositivo de almacenamiento secundario (habitualmente esto sucede cuando una sesión lleva bastante tiempo sin

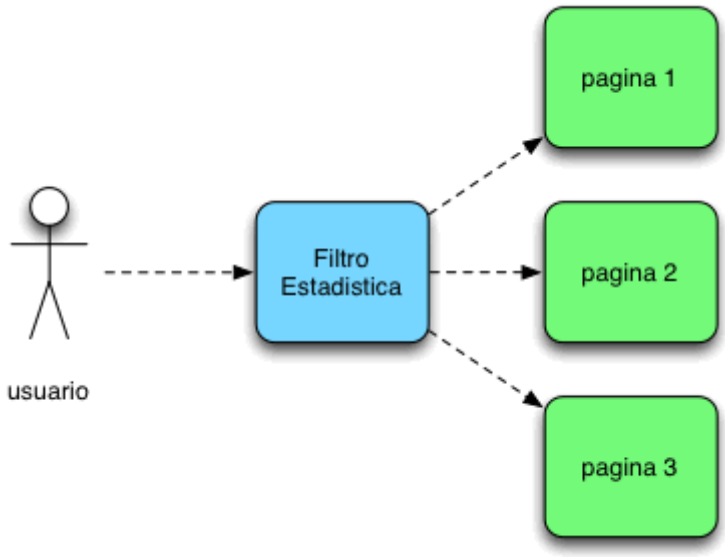
usarse, aunque no ha expirado, y se está acabando la memoria). Todas estas interfaces se encuentran en el paquete javax.servlet.http o jakarta.servlet.http.

7.3 Filtros en Serlvets

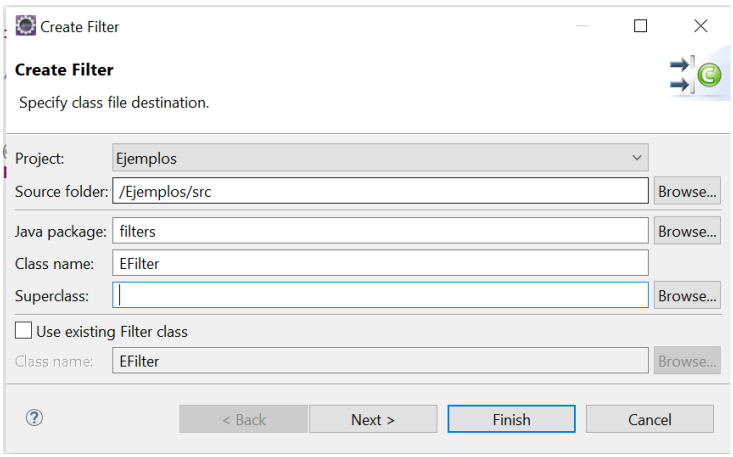
En una aplicación Java EE es posible definir una serie de **filtros que intercepten peticiones** que llegan desde el usuario. Un filtro de Java se encarga de añadir una nueva funcionalidad a la aplicación colocándose entre el usuario y las páginas. Estos filtros **definen patrones de las URLs de tal forma que cualquier petición que llegue al servidor empleando una URL que coincida con dicho patrón será pasado al filtro antes de llegar al recurso** estático o dinámico al cual fue dirigida.

En una aplicación web puede haber definidos múltiples filtros. Los filtros serán invocados en el orden en el cual fueron definidos en el descriptor de despliegue de la aplicación. Pero puede ser que dos o más filtros hayan indicado que quieren interceptar patrones de URL que en ocasiones son compatibles el uno con el otro, y por tanto deberán ser aplicados a la petición. En este caso la petición pasará primero por el filtro definido, el cual una vez haya hecho su trabajo deberá pasar la petición al siguiente filtro de la cadena, y así sucesivamente. El último filtro de cadena pasa la petición al recurso solicitado por el usuario.

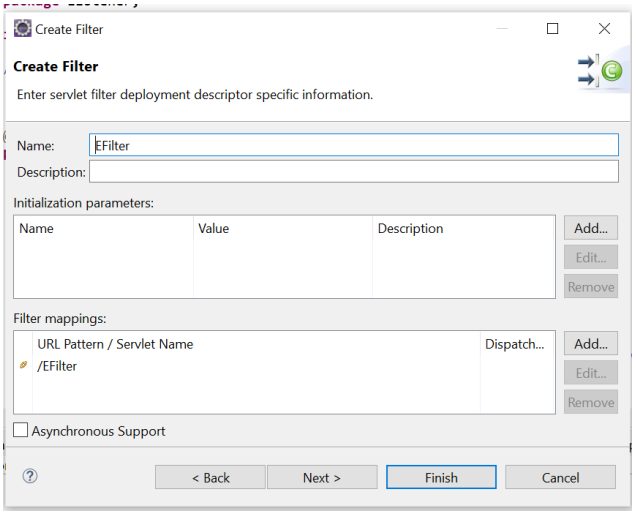
Podría darse la situación de que los filtros decidan que la petición no esté autorizada, o no debe de llegar a ese recurso por cualquier otro motivo, y deciden redirigir la petición a otro recurso o que ellos mismos generen la respuesta.

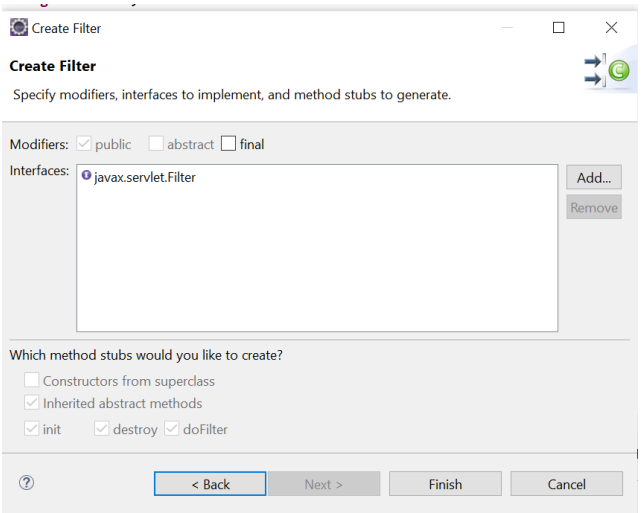


Para crear un filtro en Eclipse desde el proyecto donde estés trabajando accedes a la opción de menú **New->Filter.**



A continuación, en la siguiente ventana se pueden incluir parámetros de inicialización o el mapeo sobre los servlets a filtrar.





Para definir filtros debemos crear una clase que implemente la **interfaz javax.servlet.Filter o jakarta.servlet.Filter**. Esta clase define tres métodos:

- ✓ **public void init (filterconfig filterConfig):** este método es equivalente al método de inicialización de los servlets. Cuando el filtro es creado se invoca a este método para inicializarle. En el objeto que se le pasa como argumento es posible que encontremos información relativa a parámetros de configuración del filtro.
- ✓ **public void destroy():** este método es equivalente al método de destrucción de los servlets. Es invocado cuando el filtro se va a destruir.
- ✓ **public void doFilter(ServletRequest request,ServletResponse response, Filterchain chain):** este es el método que es invocado cada vez que ha llegado una petición compatible con los patrones de URL que desea interceptar el filtro. Los dos primeros objetos que se le pasan son los objetos de petición y respuesta asociados a la petición del usuario y el tercer parámetro es el objeto FilterChain, tiene un único método **doFilter (request, response)**, el cual deberemos invocar una vez este filtro haya terminado su trabajo para pasar la petición al siguiente filtro, o para enviarla finalmente al recurso que la debe recibir.

Podemos registrar un filtro empleando el descriptor de despliegue de la aplicación en especificaciones anteriores a Servlets 3.0 o en especificaciones posteriores a través de la anotación **@WebFilter.**

Los filtros pueden emplearse, por ejemplo, en autenticación, autorización, comprimir o descomprimir las peticiones o respuestas del servidor, lanzar un trigger ante determinados eventos, realizar analíticas detalladas del uso de la aplicación web, etc.

7.3.1. Ejemplo 14

El filtro intercepta todas las peticiones que llegan a nuestra aplicación e imprime en consola un mensaje indicando la IP de la máquina desde la cual el usuario ha realizado la petición. En el caso de que la aplicación se corresponda con un envío de un formulario, va a extraer todos los parámetros del formulario y también los va a mostrar por consola.

```java
//@WebFilter("/*")
//@WebFilter("/Ejemplo14")
@WebFilter(urlPatterns = { "/Ejemplo14", "/Ejemplo14b" })
public class EFilter extends HttpFilter implements Filter {

    /**
     * @see HttpFilter#HttpFilter()
     */
    public EFilter() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see Filter#destroy()
     */
    public void destroy() {
        // TODO Auto-generated method stub
    }


    /**
     * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
     */
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        StringBuilder buffer = new StringBuilder();
        Map<String, String[]> params = request.getParameterMap();
        for (Map.Entry<String, String[]> entry : params.entrySet()) {
            String key = entry.getKey();
            buffer.append(key);
            buffer.append("=");
            String[] val = entry.getValue();
            for (String string : val) {
                buffer.append(string);
                buffer.append("|");
            }
        }
        System.out.println("Recibida peticion Desde la IP: " + request.getRemoteAddr());
        if (!buffer.toString().equals("")) {
            System.out.println("\n\tla petición tiene los parametros: " + buffer);
        }

        // pass the request along the filter chain
        chain.doFilter(request, response);
    }

    /**
     * @see Filter#init(FilterConfig)
     */
    public void init(FilterConfig fConfig) throws ServletException {
        // TODO Auto-generated method stub
    }
```

```java
public class Ejemplo14 extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Ejemplo14() {
        super();
        // TODO Auto-generated constructor stub
    }

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

        response.setContentType("text/html;Charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<body>");
        out.println("<form action='Ejemplo14' method='post'>");
        out.println("Filtro1:<input type='text' name='filtro1' /><br>");
        out.println("<input type='submit' name='Enviar1' value='Enviar1' />");
        out.println("</form>");
        out.println("</body></html>");
        out.close();

    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     *      response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html;Charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("Filtro1: " + request.getParameter("filtro1"));

    }
```

```java
public class Ejemplo14b extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Ejemplo14b() {
        super();
        // TODO Auto-generated constructor stub
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;Charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<body>");
        out.println("<form action='Ejemplo14b' method='post'>");
        out.println("Filtro2:<input type='text' name='filtro2' /><br>");
        out.println("<input type='submit' name='Enviar2' value='Enviar2' />");
        out.println("</form>");
        out.println("</body></html>");
        out.close();
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println("Filtro2: " + request.getParameter("filtro2"));
    }
}
```