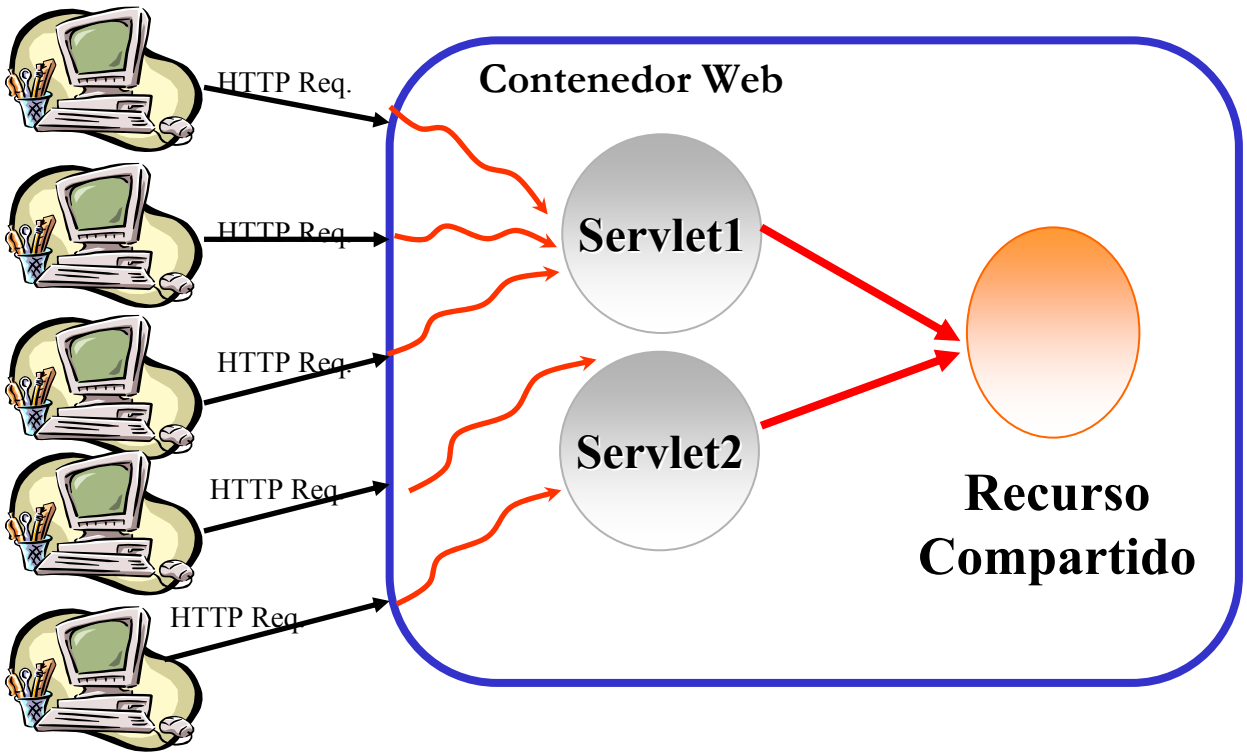
Invocar a Recursos Web

Problemas de Concurrencia en Aplicaciones Web

Múltiples requerimientos simultáneos a un servlet



- Un **Servlet** es un ***singleton***, esto significa que existe una única instancia del Servlet en memoria.
- Los **servidores web** son diseñados para manejar **múltiples requerimientos concurrentes** de los clientes, para ello **crean múltiples threads** y cada uno de ellos maneja un requerimiento.
- Cuando el servidor web recibe múltiples requerimientos para un servlet, todos los requerimientos son entregados a la instancia del Servlet sobre múltiples threads concurrentes, por ello el **Servlet** debe ser **escrito de una manera thread-safe**.

Alcances de los Datos de una Aplicación Web

Alcance	Thread-Safe	Ejemplo

Variables Instancia	NO	class Hola {
		private double d;
		}

Variables Clase	NO	class Hola {
		private static double d;
		}

Parámetros de Métodos	SI	void doGet(HttpServletRequest req, HttpServletResponse resp)

Variables Locales	SI	void doGet(HttpServletRequest req,
		HttpServletResponse resp) {
		double d;
		}

Datos de Sesión	NO	HttpSession se= req.getSession();
Datos de Aplicación	NO	ServletContext ctx= this.getServletContext();

Proteger Datos

¿Qué NO necesita ser protegido?

•**Variables Locales:** un thread tiene su propia pila de ejecución y las variables locales viven en la pila. Un único thread puede tocar dichos datos en un instante dado de tiempo.

•**Parámetros de Métodos:** son variables locales. En particular, está garantizado que los objetos **HttpServletRequest** y **HttpServletResponse** son **thread-safe**. El servidor web creará un nuevo objeto *request* y *response* por cada requerimeinto nuevo recibido de un cliente o el servidor *pooleará* los objetos *request* y *response*. En ambos casos, está garantizado que cada invocación a un método tomará su propio objeto *request* y *response*.

¿Qué necesita ser protegido?

•**Variables de Instancia:** son compartidas por todos los threads concurrentes que se ejecutan sobre la instancia del servlet.

•**Variables de Clase:** una clase de un Servlet, puede ser usada para instanciar múltiples servlets dentro de una aplicación web. Se tienen varias instancias de Servlets de la misma clase, cada una con su propio nombre. Las variables de clase son compartidas por todos los threads concurrentes que se ejecutan sobre dichas instancias de servlets.

•**Variables de Alcance Sesión:** existen mientras dure la sesión de usuario, es decir durante múltiples requerimientos realizados desde un mismo cliente. Aparentemente no hay problemas de concurrencia. Es posible que el usuario ejecute más de un *browser* desde el mismo cliente. En tal caso, threads concurrentes acceden a los datos de la sesión.

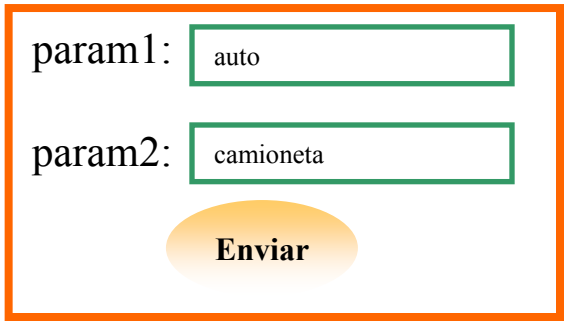
•**Variables de Alcance Aplicación:** existen mientras este levantada la aplicación y son compartidas por todas las componentes web de la aplicación.

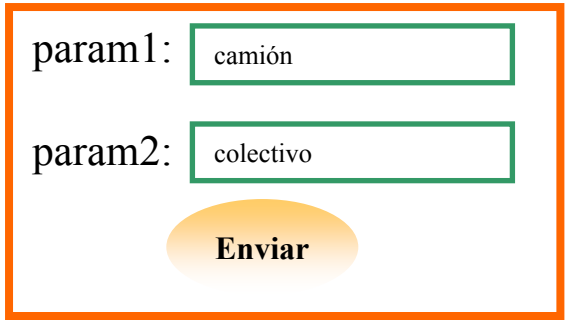
Ejemplo
Proteger Variables de Instancia

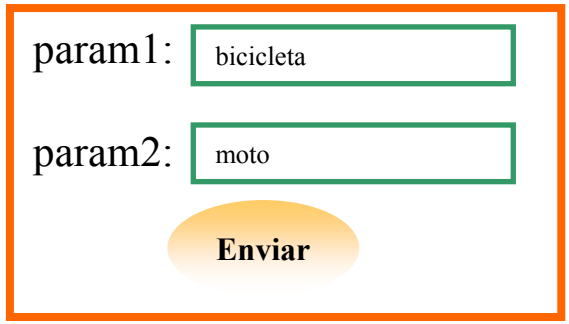
```
package misservlets.protegidos;

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

public class NoThreadSafe extends HttpServlet {
   private String valor1;
   private String valor2;
   public void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException {
   resp.setContentType("text/html");
   PrintWriter out=resp.getWriter();
   out.println("<html>");
   out.println("<body>");
   out.println("Los Valores de los Parámetros son: ");
   valor1=req.getParameter("param1");
   valor2=req.getParameter("param2");
   out.println("Parámetro 1: "+valor1);
   out.println("Parámetro 2: "+valor2);
   out.println("</body>");
   out.println("</html>");
   out.close();
   }
}// Fin de la clase
```

El Servlet muestra cualquier par de parámetros, los últimos guardados en las variables de instancia, NO necesariamente el par de parámetros dado por el cliente

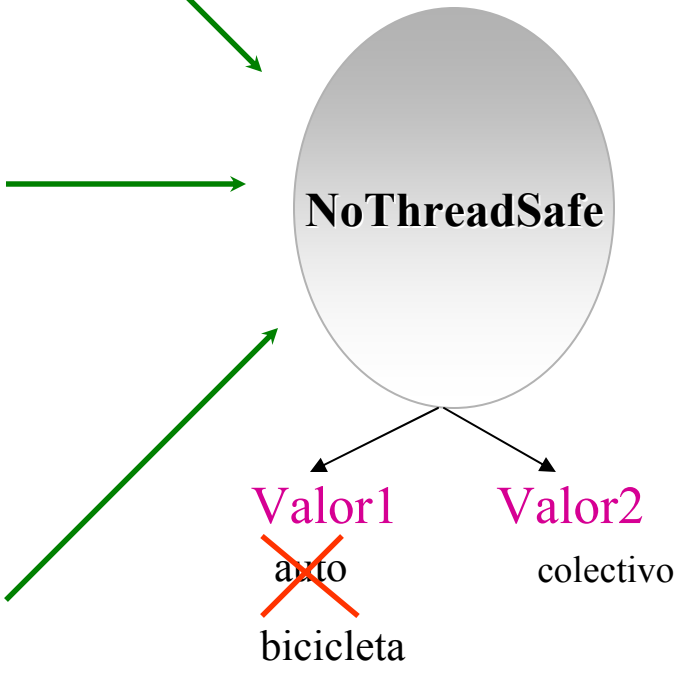
Ejemplo
Proteger Variables de Instancia







Los Valores de los Parámetros son:
Bicicleta
Colectivo



NO ESTA GARANTIZADO que el cliente modifica el valor de los parámetros e inmediatamente se ejecuta el código que los muestra. Un requerimiento de otro cliente podría modificar estos valores antes que sean mostrados.

Ejemplo
Proteger Variables de Instancia

```
package misservlets.protegidos;

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

public class ThreadSafe extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException {
   String valor1;
   String valor2;
   resp.setContentType("text/html");
   PrintWriter out=resp.getWriter();
   out.println("<html>");
   out.println("<body>");
   out.println("Los Valores de los Parámetros son: ");
   valor1=req.getParameter("param1");
   valor2=req.getParameter("param2");
   out.println("Parámetro 1: "+valor1);
   out.println("Parámetro 2: "+valor2);
   out.println("</body>");
   out.println("</html>");
   out.close();
   }
}// Fin de la clase
```

Usar variables locales siempre que sea posible

Ejemplo
Proteger Variables de Instancia

```
package misservlets.protegidos;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ThreadSafe extends HttpServlet {
   private String valor1;
   private String valor2;
   public synchronized void doGet(HttpServletRequest req, HttpServletResponse
   resp) throws ServletException, IOException {
   resp.setContentType("text/html");
   PrintWriter out=resp.getWriter();
   out.println("<html>");
   out.println("<body>");
   out.println("Los Valores de los Parámetros son: ");
   valor1=req.getParameter("param1");
   valor2=req.getParameter("param2");
   out.println("Parámetro 1: "+valor1);
   out.println("Parámetro 2: "+valor2);
   out.println("</body>");
   out.println("</html>");
   out.close();
   }
}// Fin de la clase
```

Declarar el método ***synchronized*** restringe al servlet a procesar de a un requerimiento a la vez.

Garantiza la Sincronización
Baja Performance

```
synchronized (this)
{
valor1=req.getParameter("param1");
valor2=req.getParameter("param2");
out.println("Parámetro 1: "+valor1);
out.println("Parámetro 2: "+valor2);
}
```

Redireccionar la Respuesta
Delegar el Requerimiento

Redireccionar la Respuesta

sendRedirect(String URL) de la interface javax.servlet.HttpServletResponse

Redirecciona el requerimiento del cliente a la URL especificada en el parámetro.

¿Cómo funciona? Usa el código de respuesta HTTP 302, que indica "El recurso que está buscando el cliente fue temporariamente movido". Toma el String que recibe como parámetro, que representa una URL y automáticamente le setea código HTTP 302. El *browser,* sin informar al usuario, direcciona el requerimiento a la URL enviada.

Delegar el Requerimiento

El proceso íntegro de delegación del requerimiento, se realiza del lado del servidor, a diferencia del redireccionamiento de la respuesta, no requiere de ninguna acción por parte del cliente ni del envío de información extra entre el cliente y el servidor.

Es necesario obtener el objeto **javax.servlet.RequestDispatcher**, invocando a alguno de los siguientes métodos del objeto **ServletRequest o ServletContext**:

getRequestDispatcher(String path) El path, si es relativo a la raíz del ServletContext comienza con "/"

getNamedDispatcher(String nom) nom es nombre de un servlet declarado en el web.xml

El objeto **RequestDispatcher** provee el método:

foward(javax.servlet.ServletRequest, javax.servlet.ServletResponse)

Delega el requerimiento y la respuesta al objeto RequestDisptacher. Permite pasar el requerimiento actual a otro servlet para que continúe el procesamiento y responda al cliente

Redireccionar la Respuesta
Delegar el Requerimiento

Redireccionar la Respuesta

sendRedirect(String URL) de la interface javax.servlet.HttpServletResponse

resp.sendRedirect("/Productos");

resp.sendRedirect("http://www.google.com.ar");

Usar URL' absolutas

Delegar el Requerimiento

foward(javax.servlet.ServletRequest, javax.servlet.ServletResponse)

```
RequestDispatcher dispatcher=request.getRequestDispatcher("header.html");
if (dispatcher!=null) dispatcher.forward(request,response);
```

```
ServletContext ctx=this.getServletContext()
RequestDispatcher dispatcher=ctx.getRequestDispatcher("/cliente/header.html");
if (dispatcher!=null) dispatcher.forward(request,response);
```

Código en **inicio.jsp**

/cliente/header.html
/cliente/inicio.jsp

Funcionan idénticos

Incluir Recursos

- Delegar el requerimiento frecuentemente se usa para dividir a un servlet en partes más pequeñas. Por ejemplo: un servlet que genera dinámicamente un encabezamiento que todas las páginas del sitio incluyen.
- El método **include()** del objeto RequestDispatcher o ServletContext permite incluir la salida de un Servlet en la salida de otro Servlet. Por ejemplo, incluir el Servlet que genera el encabezamiento en otro Servlet. De esta manera, cualquier cambio en el Servlet incluído, se reflejará automáticamente en todos los Servlets que lo incluyen. Puede incluirse cualquier recurso web, estático o dinámico.

include(javax.servlet.ServletRequest, javax.servlet.ServletResponse)

```
RequestDispatcher dispatcher=request.getRequestDispatcher("header.html");
if (dispatcher!=null) dispatcher.include(request,response);
```

También es un método de ServletContext