Práctica No. 6. Manejo de Eventos con Javascript

Preparación del Entorno

- Descargar de la página web http://academicos.azc.uam.mx/jfg/pags/tarea_taller_web.html los archivos del Servidor ***Apache Tomcat***
- Instalar, agregar y arrancar el servidor
- Descargar e importar el proyecto contenido en el archivo **ManejoEventosBase.zip**

Se creará un directorio en ***WEB-INF*** con el nombre ***js*** para agregar diversos archivos en *javascript*.

Para que se reconozcan las rutas de los archivos, se modificará el archivo de configuración de Spring.

controlador-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:context="http://www.springframework.org/schema/context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/mvc
         http://www.springframework.org/schema/mvc/spring-mvc.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd">

         <context:component-scan base-
package="uam.tdaw.eventos.controladores" />

         <bean id="viewResolver"

      class="org.springframework.web.servlet.view.UrlBasedViewResolver">
            <property name="viewClass"
                  value="org.springframework.web.servlet.view.JstlView" />
            <property name="prefix" value="/jsp/" />
         <property name="suffix" value=".jsp" />
      </bean>

      <mvc:resources mapping="/recursos/**" location="/WEB-INF/"/>
          <mvc:annotation-driven />

</beans>
```

Esto indica que cualquier invocación a la ruta ***recursos*** se redireccionará directamente a ***/WEB-INF/***

Probando la carga de un archivo javascript

Se probará que al seleccionar un *radio button*, se muestre una alerta. Para esto es necesario indicar la ruta de un archivo de Javascript

eventosRadioButtons.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="tag"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<script src="<c:url value="/recursos/js/alerta.js" />"></script>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>Eventos con Radio Buttons</h1>

<tag:form action="leerOpcionRadioButtons" method="POST"
modelAttribute="opcionRadioButton">

     <tag:radiobutton path="clave" value="${opcionDefecto.clave}"
     label="${opcionDefecto.nombre}" checked="checked"
onclick="unaAlertaSencilla()"/>
     <tag:radiobuttons items="${listaDeOpciones}" itemValue="clave"
itemLabel="nombre" path="clave"
     onclick="unaAlertaSencilla()"/>

     <br><br>
     <tag:button> Enviar Selección</tag:button>
</tag:form>
<br><br>

<a href="menuPrincipal">Regresar al Menú Principal</a>
</body>
</html>
```

Se agrega una etiqueta para indicar una url:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

Posteriormente, se agrega el evento `onclick` a cada radio button para que al dar clic en él, se invoque a la función *`unaAlertaSencilla()`* que se encuentra en el archivo ***alerta.js***

alerta.js

```
function unaAlertaSencilla() {

     alert("Esta es una alerta Sencilla");
}
```

Recuperando un valor seleccionado

A continuación se recuperará el valor de un elemento al dar clic en él, para eso se creará una nueva función en el archivo ***alerta.js***

alerta.js

```
function recuperandoElDato(seleccionado) {
     alert("El valor es " + seleccionado.value);
}
```

Aquí se recibe un parámetro `seleccionado` del cuál se obtiene el valor a través del atributo `value`

eventosRadioButton.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="tag"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<script src="<c:url value="/recursos/js/alerta.js" />"></script>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>Eventos con Radio Buttons</h1>

<tag:form action="leerOpcionRadioButtons" method="POST"
modelAttribute="opcionRadioButton">

     <tag:radiobutton path="clave" value="${opcionDefecto.clave}"
     label="${opcionDefecto.nombre}" checked="checked"
onclick="recuperandoElDato(this)"/>
```

```
      <tag:radiobuttons items="${listaDeOpciones}" itemValue="clave"
itemLabel="nombre" path="clave"
      onclick="recuperandoElDato(this)"/>
      <br><br>
      <tag:button> Enviar Selección</tag:button>

</tag:form>
<br><br>

<a href="menuPrincipal">Regresar al Menú Principal</a>
</body>
</html>
```

Se agregará una función similar al manejo con listas de selección, para esto se realizará en un nuevo archivo javascript llamado ***pasandoParametros.js.***

pasandoParametros.js

```
function recuperandoParametros(seleccionado) {
      alert("El valor es " + seleccionado.value);
}
```

Sin embargo, la mayoría de los navegadores, no reconoce la opción *onclick* en un elemento *option* dentro de *select*, por lo que se utilizará la propiedad `onchange` dentro de `select`

eventosListas.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
    <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="tag"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script src="<c:url value="/recursos/js/pasandoParametros.js"
/>"></script>
<title>Insert title here</title>
</head>
<body>
<h1>Eventos con Listas</h1>


<tag:form action="leerOpcionListas" method="POST"
modelAttribute="continente">
```

```
<tag:select path="claveContinente" onchange="recuperandoParametros(this)">

            <tag:options items="${listaContinentes}"
itemValue="claveContinente" itemLabel="nombreContinente"/>

      </tag:select>
      <br><br>
      <tag:button>Enviar Selección</tag:button>
</tag:form>
<br><br>
<a href="menuPrincipal">Regresar al Menú Principal</a>
</body>
</html>
```

Ahora se mandará este parámetro a un controlador sin tener que presionar el botón de enviar, para esto se modificará el archivo JS.

pasandoParametros.js

```
function recuperandoParametros(seleccionado) {
      document.getElementById("formaListas").submit();
}
```

Es necesario indicar el id de la forma en la vista correspondiente.

```
<tag:form              action="leerOpcionListas"              method="POST"
modelAttribute="continente" id="formaListas">
```

Esto envía los datos al método que atrapa la invocación al *action*. Será necesario realizar modificaciones para que presente la vista con el elemento seleccionado.

CapturarLecturasController.java

```
@RequestMapping("leerOpcionListas")
      public ModelAndView
capturarOpcionesListas(@ModelAttribute("continente") Continente
continente){
            System.out.println("Se registró el continente: " +
continente.getClaveContinente());

            ModelAndView modelo = new ModelAndView("eventosListas");
            AdministrarListas administrar = new AdministrarListas();
            LinkedList<Continente>lista = administrar.listaContinenentes();
            LinkedList<Continente>listaContinentes =
administrar.marcaSeleccionado(lista,continente.getClaveContinente());

            modelo.addObject("listaContinentes", listaContinentes);
            modelo.addObject("continente", continente);
            return modelo;
      }
```

Se agrega el método para “marcar” al elemento que ha sido seleccionado.

AdministrarListas.java

```
public
LinkedList<Continente>marcaSeleccionado(LinkedList<Continente>lista,
String clave){

            for(int i=0;i<lista.size();i++){
                  Continente aux = lista.get(i);
                  if(aux.getClaveContinente().compareTo(clave)==0){
                        lista.get(i).setSeleccionado(true);
                  }
            }

            return lista;
      }
```

También es necesario realizar modificaciones a la vista

eventosListas.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
    <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="tag"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script src="<c:url value="/recursos/js/pasandoParametros.js"
/>"></script>
<title>Insert title here</title>
</head>
<body>
<h1>Eventos con Listas</h1>

<tag:form action="leerOpcionListas" method="POST"
modelAttribute="continente" id="formaListas">

<tag:select path="claveContinente" onchange="recuperandoParametros(this)">
            <c:forEach items="${listaContinentes}" var="claveContinente">
                  <c:if test="${claveContinente.seleccionado==true}">
                  <tag:option value="${claveContinente.claveContinente}"
label="${claveContinente.nombreContinente}"
                  selected="true"/>
                  </c:if>
                  <c:if test="${claveContinente.seleccionado==false}">
```

```
                <tag:option value="${claveContinente.claveContinente}"
label="${claveContinente.nombreContinente}"/>
                </c:if>
            </c:forEach>
      </tag:select>
      <br><br>

      <tag:button>Enviar Selección</tag:button>
</tag:form>
<br><br>
<a href="menuPrincipal">Regresar al Menú Principal</a>
</body>
</html>
```

En la clase ***Continente*** se agrega el atributo para saber si es el elemento seleccionado

Continente.java

```
package uam.tdaw.eventos.clases;

public class Continente {

      private String nombreContinente;
      private String claveContinente;
      private boolean seleccionado;

      public String getNombreContinente() {
            return nombreContinente;
      }
      public void setNombreContinente(String nombreContinente) {
            this.nombreContinente = nombreContinente;
      }
      public String getClaveContinente() {
            return claveContinente;
      }
      public void setClaveContinente(String claveContinente) {
            this.claveContinente = claveContinente;
      }
      public boolean isSeleccionado() {
            return seleccionado;
      }
      public void setSeleccionado(boolean seleccionado) {
            this.seleccionado = seleccionado;
      }
}
```

Se agregará una manera de avanzar al presionar el botón para que no se esté cargando siempre la misma vista.

pasandoParametros.js

```
function recuperandoParametros(seleccionado) {
	document.getElementById("formaListas").submit();
}

function avanzarBoton(){
	document.getElementById("txtAvanzar").value=1;
}
```

Se agrega un nuevo manejador de eventos al botón

```
<tag:button onclick="avanzarBoton()">Enviar Selección</tag:button>
```

Es necesario agregar un nuevo atributo a la clase ***Continente*** para leer el parámetro a ser utilizado como una bandera.

Continente.java

```
package uam.tdaw.eventos.clases;

public class Continente {

	private String nombreContinente;
	private String claveContinente;
	private boolean seleccionado;
	private int avanzar;

	public String getNombreContinente() {
		return nombreContinente;
	}
	public void setNombreContinente(String nombreContinente) {
		this.nombreContinente = nombreContinente;
	}
	public String getClaveContinente() {
		return claveContinente;
	}
	public void setClaveContinente(String claveContinente) {
		this.claveContinente = claveContinente;
	}
	public boolean isSeleccionado() {
		return seleccionado;
	}
	public void setSeleccionado(boolean seleccionado) {
		this.seleccionado = seleccionado;
	}
	public int getAvanzar() {
```

```
            return avanzar;
      }
      public void setAvanzar(int avanzar) {
            this.avanzar = avanzar;
      }
}
```

Se agrega el manejo de la bandera en el controlador correspondiente.

CapturarLecturasController.java

```
      @RequestMapping("leerOpcionListas")
      public ModelAndView
capturarOpcionesListas(@ModelAttribute("continente") Continente
continente){

            System.out.println("Se registró el continente: " +
continente.getClaveContinente());

            if(continente.getAvanzar()==0){
                  ModelAndView modelo = new ModelAndView("eventosListas");
                  AdministrarListas administrar = new AdministrarListas();

                  LinkedList<Continente>lista =
administrar.listaContinenentes();
                  LinkedList<Continente>listaContinentes =
administrar.marcaSeleccionado(lista,continente.getClaveContinente());

                  modelo.addObject("listaContinentes", listaContinentes);
                  modelo.addObject("continente", continente);
                  return modelo;
            }
            else{
                  return null;
            }
      }
```

En este caso no se avanza a ninguna vista, pero podría llamarse a cualquiera de las vistas ya establecidas.

Solo falta ocultar la caja de texto para que no sea visible.

```
<tag:hidden path="avanzar" id="txtAvanzar"/>
```

Listas Dependientes

La primer lista, la de los Continentes, tendrá un manejo similar al de la vista de las listas, incluyendo el manejo de banderas en el controlador, por lo que se agregará esta

DestinoFrm.java

```
package uam.tdaw.eventos.formas;

public class DestinoFrm {

	private String claveContinente;
	private String clavePais;
	private int avanzar;

	public String getClaveContinente() {
		return claveContinente;
	}
	public void setClaveContinente(String claveContinente) {
		this.claveContinente = claveContinente;
	}

	public String getClavePais() {
		return clavePais;
	}
	public void setClavePais(String clavePais) {
		this.clavePais = clavePais;
	}
	public int getAvanzar() {
		return avanzar;
	}
	public void setAvanzar(int avanzar) {
		this.avanzar = avanzar;
	}
}
```

Se creará un nuevo archivo de *javascript* en el directorio ***js*** llamado ***listasDependientes.js***

listasDependientes.js

```
function recuperandoContinente() {
	document.getElementById("txtAvanzar").value=0;
	document.getElementById("formaListasDependientes").submit();
}

function avanzarBoton(){
	document.getElementById("txtAvanzar").value=1;
}
```

Se agrega la funcionalidad para manejar la primera lista en el controlador.

CapturarLecturasController.java

```
@RequestMapping("leerOpcionListasDependientes")
	public ModelAndView
capturarOpcionesListasDependientes(@ModelAttribute("datosDestino")
DestinoFrm destino){

		System.out.println("Se registró el continente: " +
destino.getClaveContinente());


		if(destino.getAvanzar()==0){
			ModelAndView modelo = new
ModelAndView("eventosListasDependientes");
			AdministrarListas administrar = new AdministrarListas();

			LinkedList<Continente>lista =
administrar.listaContinenentes();
			LinkedList<Continente>listaContinentes =
administrar.marcaSeleccionado(lista,destino.getClaveContinente());

			modelo.addObject("listaContinentes", listaContinentes);
			modelo.addObject("datosDestino", destino);
			return modelo;

		}
		else{
			return null;
		}
	}
```

Se agrega la funcionalidad para cargar los países dependiendo el continente seleccionado.

CapturarLecturasController.java

```
@RequestMapping("leerOpcionListasDependientes")
	public ModelAndView
capturarOpcionesListasDependientes(@ModelAttribute("datosDestino")
DestinoFrm destino){

		System.out.println("Se registró el continente: " +
destino.getClaveContinente());


		if(destino.getAvanzar()==0){
			ModelAndView modelo = new
ModelAndView("eventosListasDependientes");
			AdministrarListas administrar = new AdministrarListas();
```

```
                LinkedList<Continente>lista =
administrar.listaContinenentes();
                LinkedList<Continente>listaContinentes =
administrar.marcaSeleccionado(lista,destino.getClaveContinente());

                LinkedList<Pais> listaPaises =
administrar.listaPaises(destino.getClaveContinente());

                modelo.addObject("listaContinentes", listaContinentes);
                modelo.addObject("listaPaises", listaPaises);

                modelo.addObject("datosDestino", destino);
                return modelo;
          }
          else{
                return null;
          }
     }
```

Se agrega a la vista la presentación de la segunda lista.

EventosListasDependientes.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="tag"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script src="<c:url value="/recursos/js/listasDependientes.js"
/>"></script>
<title>Insert title here</title>
</head>
<body>
<h1>Eventos con Listas Dependientes</h1>


<tag:form action="leerOpcionListasDependientes" method="POST"
modelAttribute="datosDestino" id="formaListasDependientes">

<tag:select path="claveContinente" onchange="recuperandoContinente()">

          <c:forEach items="${listaContinentes}" var="claveContinente">
```

```
                <c:if test="${claveContinente.seleccionado==true}">
                <tag:option value="${claveContinente.claveContinente}"
label="${claveContinente.nombreContinente}"
                selected="true"/>
                </c:if>
                <c:if test="${claveContinente.seleccionado==false}">
                <tag:option value="${claveContinente.claveContinente}"
label="${claveContinente.nombreContinente}"/>
                </c:if>
            </c:forEach>

      </tag:select>

      <br><br>

      <tag:select path="clavePais">
            <tag:options items="${listaPaises}" var="datosPais"
itemValue="clave" itemLabel="nombre"/>
      </tag:select>

      <tag:hidden path="avanzar" id="txtAvanzar"/>
      <br><br>

      <tag:button onclick="avanzarBoton()">Enviar Selección</tag:button>

</tag:form>
<br><br>
<a href="menuPrincipal">Regresar al Menú Principal</a>

</body>
</html>
```

Con esto ya se tiene la funcionalidad de que al seleccionar un continente, se presentan los países que corresponden a cada uno de ellos, sin embargo, esto todavía puede mejorarse. Se manejarán dos opciones:

Se hará que al presentar la pantalla, se carguen los datos correspondientes al primer elemento, en este caso del continente Americano. Para esto es necesario realizar la carga en cuanto se presenta la pantalla, por lo que se agrega la siguiente instrucción a la vista.

eventosListasDependientes.jsp

```
<body onload="cargaInicial()">
```

Esto indica que al cargar el cuerpo, se invoque a la función `cargaInicial()`

listasDependientes.js

```
function cargaInicial(){
	document.getElementById("formaListasDependientes").submit();
}
```

Sin embargo esto ciclaría la invocación de la página, por lo que se agregará el manejo de una bandera que indique cuando es necesario realizar nuevamente la carga de la página y cuando no.

listasDependientes.js

```
function cargaInicial(){
	var bandera = document.getElementById("txtBandera").value;
	if(bandera == 0)
	document.getElementById("formaListasDependientes").submit();
}
```

Es necesario modificar la vista, el controlador y la forma.

eventosListasDependientes.js

```
<tag:hidden path="avanzar" id="txtAvanzar"/>
	<tag:input path="bandera" id="txtBandera" value="${txtBandera}"/>
```

DestinoFrm.java

```
package uam.tdaw.eventos.formas;

public class DestinoFrm {

	private String claveContinente;
	private String clavePais;
	private int avanzar;
	private int bandera;

	public String getClaveContinente() {
		return claveContinente;
	}
	public void setClaveContinente(String claveContinente) {
		this.claveContinente = claveContinente;
	}
	public String getClavePais() {
		return clavePais;
	}
	public void setClavePais(String clavePais) {
		this.clavePais = clavePais;
	}
	public int getAvanzar() {
		return avanzar;
	}
	public void setAvanzar(int avanzar) {
```

```
            this.avanzar = avanzar;
      }
      public int getBandera() {
            return bandera;
      }
      public void setBandera(int bandera) {
            this.bandera = bandera;
      }
}
```

CapturarLecturasController.java

```
@RequestMapping("leerOpcionListasDependientes")
      public ModelAndView
capturarOpcionesListasDependientes(@ModelAttribute("datosDestino")
DestinoFrm destino){

            System.out.println("Se registró el continente: " +
destino.getClaveContinente());


            if(destino.getAvanzar()==0){
                  ModelAndView modelo = new
ModelAndView("eventosListasDependientes");
                  AdministrarListas administrar = new AdministrarListas();

                  LinkedList<Continente>lista =
administrar.listaContinenentes();
                  LinkedList<Continente>listaContinentes =
administrar.marcaSeleccionado(lista,destino.getClaveContinente());

                  LinkedList<Pais> listaPaises =
administrar.listaPaises(destino.getClaveContinente());

                  modelo.addObject("listaContinentes", listaContinentes);
                  modelo.addObject("listaPaises", listaPaises);
                  modelo.addObject("txtBandera",1);
                  modelo.addObject("datosDestino", destino);
                  return modelo;

            }
            else{
                  System.out.println("Se registró el país: " +
destino.getClavePais());
                  return null;
            }

      }
```

En este caso se indica que si la bandera es igual a 0, se realiza la invocación, de otra manera, no se hace, por lo que cada carga tras seleccionar un Continente solo despliega la nueva vista una vez y se elimina el ciclo.

Solo resta ocultar la caja de texto de la bandera.

eventosListasDependientes.js

```
<tag:hidden path="bandera" id="txtBandera" value="${txtBandera}"/>
```

Esto permite que en cuanto se muestra la vista, se tiene seleccionado un elemento y se presentan los elementos correspondientes en la lista dependiente.

Otra opción es presentar una selección por defecto y ocultar la lista dependiente hasta el momento en que se seleccione una opción válida.

Para esto se agrega la opción al inicio de la lista.

eventosListaDependientes.js

```
<tag:option value="DEF" label="Selecciona el Continente"/>
            <c:forEach items="${listaContinentes}" var="claveContinente">
                  <c:if test="${claveContinente.seleccionado==true}">
                  <tag:option value="${claveContinente.claveContinente}"
label="${claveContinente.nombreContinente}"
                  selected="true"/>
                  </c:if>
                  <c:if test="${claveContinente.seleccionado==false}">
                  <tag:option value="${claveContinente.claveContinente}"
label="${claveContinente.nombreContinente}"/>
                  </c:if>
            </c:forEach>
```

Se identifica la lista principal.

eventosListasDependientes.js

```
<tag:select  path="claveContinente"  onchange="recuperandoContinente(this)"
id="listaContinente">
```

Se implementa la funcionalidad para solo enviar el valor al controlador si no se ha seleccionado la opción por defecto.

listaDependientes.js

```
function recuperandoContinente(seleccionado) {
      document.getElementById("txtAvanzar").value=0;

      var comparacion = seleccionado.value.localeCompare("DEF");

      if( comparacion != 0){
```

```
      document.getElementById("formaListasDependientes").submit();
      }else{
      document.getElementById('listaPaisesDIV').style.display = "none";
      }
}
```

Se ocultará el elemento utilizando una hoja de estilos, por lo que se creará el directorio ***css*** en ***WEB-INF***. Se creará un archivo llamado ***ocultar.css***

ocultar.css

```
#listaPaisesDIV{
      display:none;
}
```

Se invoca a un elemento llamado *listaPaisesDIV* que se ocultará de manera inicial, es necesario crear este contenedor en la vista que encierre a la lista y adicionalmente a una leyenda.

eventosListasDependientes.jsp

```
<div id="listaPaisesDIV">
      <h2>Selecciona el País</h2>
      <tag:select path="clavePais">

            <tag:options items="${listaPaises}" var="datosPais"
itemValue="clave" itemLabel="nombre"/>

      </tag:select>
</div>
```

Es necesario incluir el archivo de estilos en la vista.

eventosListasDependientes.js

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script src="<c:url value="/recursos/js/listasDependientes.js"
/>"></script>
<link rel="stylesheet" type="text/css" href="<c:url
value="/recursos/css/ocultar.css" />" />
<title>Insert title here</title>
</head>
```

Se agrega la funcionalidad para cambiar estilos en el archivo *js* dependiendo el valor seleccionado en la primer lista.

listasDependientes.js

```
function recuperandoContinente(seleccionado) {
	document.getElementById("txtAvanzar").value=0;

	var comparacion = seleccionado.value.localeCompare("DEF");
	if( comparacion != 0){
		document.getElementById("formaListasDependientes").submit();
	}else{
	document.getElementById('listaPaisesDIV').style.display = "none";
	}
}
```

listasDependientes.js

```
function cargaInicial(){

	var seleccionado = document.getElementById("listaContinente").value;
	var comparacion = seleccionado.localeCompare("DEF");

	if(comparacion!=0){
	document.getElementById("listaPaisesDIV").style.display = "block";
	}
}
```

De esta manera tras cada recarga, si no se ha selecciona la opción por defecto, se muestra la lista dependiente.