Anexo III. Planos de código

1. Introducción

Este capítulo presenta el código de las clases e interfaces Java que componen La Aplicación Web en primer lugar. Luego, presenta el código de las JSPs, para terminar con el código de las funciones Javascript, la biblioteca de etiquetas `util.tld` y los archivos de etiquetas "`tag`".

2. Clases e Interfaces Java

A continuación se muestra el código Java de las clases y las interfaces que forman parte de la Aplicación Web, ordenadas por paquete y subpaquete, y alfabéticamente.

2.1 Paquete `control`

Choice.java

```
package control;

import utilidades.*;
import xml.items.ChoiceXML;

import javax.servlet.http.*;
import javax.servlet.*;

import java.io.IOException;

/**
 * Componente de control implícito asociado a la página JSP "choice.jsp". Es la lógica
 * asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea
 * llamada la página. Su función es comprobar que todos los parámetros iniciales de la toma de
 * datos de una pregunta tipo "Choice" sean correctos, en cuyo caso redirecciona la respuesta
 * a la siguiente página de toma de datos de tipo "Choice". La clase implementa la interfaz
 * <code>Control</code>, implementando su método <code>doLogic</code>, que es el único que
 * contiene, y que es el encargado de realizar la lógica asociada a la página "choice.jsp".
 *
 * @author   David Domínguez
 * @see      Control
 *
 */
public class Choice implements Control{

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página "choice.jsp".
  * Será llamado por el filtro <code>ControlFilter</code> en el caso de que detecte que la
  * página requerida es "choice.jsp". Ya que es llamado desde un filtro, será ejecutado
  * siempre que llegue una petición de un cliente y antes de que la petición llegue al
  * Servlet (página JSP compilada a Servlet) que sirve la respuesta al cliente. Comprueba
  * uno por uno todos los parámetros, y, en caso de que sean correctos, crea un objeto <code>
  * ChoiceXML</code> para guardar los datos asociados al tipo de pregunta y lo guarda en el
  * ámbito de sesión para utilizarlos más adelante para terminar de crear el ítem.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
  * @see  control.ControlFilter
  * @see     javax.servlet.http.HttpServletRequest
```

```
 * @see     javax.servlet.http.HttpServletResponse
 * @see     utilidades.EstadoWeb
 * @see     utilidades.MensajeEstado
 * @see     utilidades.EnteroPositivo
 * @see     xml.items.ChoiceXML
 */
public boolean doLogic(HttpServletRequest request, HttpServletResponse response)
      throws IOException, ServletException {

   // Comprueba si es la primera vez que se visita la página
   if (request.getParameter("Form") != null){

      // Caracteres reservados no permitidos
      final String reservados = ";/?:@&=+$,\\*\"<>|";

      String identificador = request.getParameter("Identif"); // Identificador del ítem
      String titulo = request.getParameter("Tit");       // Título del ítem
      String instrucciones = request.getParameter("Instruc"); // Instrucciones del ítem
      String pregunta = request.getParameter("Preg");       // Pregunta del ítem
      String Shuff = request.getParameter("Shuffle");    // Mezclar las respuestas o no
      String numResp = request.getParameter("Num_resp");    // Número de respuestas

      // Estado de la petición web. Almacena si son correctos los parámetros, los
      // mensajes de aviso al usuario y dónde colocar el foco en la página
      EstadoWeb estado = new EstadoWeb();

      boolean shuffle = false;      // Valor por defecto del parámetro
      if ( Shuff != null){       // Interpreta el valor marcado en la página
         if (Shuff.equalsIgnoreCase("ON"))
            shuffle = true;
      }

      // Comprueba que estén introducidos todos los atributos obligatorios
      if (identificador != null && !identificador.equals("") &&
            titulo != null && !titulo.equals("") &&
            numResp != null && !numResp.equals("")){

         // Comprueba que los caracteres del identificador estén permitidos
           for(int i = 0; i < identificador.length() && estado.isEstado() == true; i++) {
            char c = identificador.charAt(i);   // Lee el siguiente carácter
            int marca = reservados.indexOf(c);
            if (marca >= 0 ){
               estado.añadeError("Error: El carácter '" + reservados.charAt(marca) +
                     "' en la posición " + (i+1) + " del Identificador no está " +
                     "permitido<br />", "Identif");
            }
           }

         // Comprueba que el número de respuestas sea un valor válido
         EnteroPositivo numRespuestas = new EnteroPositivo(numResp);
         MensajeEstado msg = numRespuestas.esPositivo();
         if(msg.isEstado()){     // Si es un número válido y mayor o igual que cero
            if (numRespuestas.getEnteroPositivo() <=  1)// Si es menor o igual que uno
               estado.añadeError("El número de respuestas debe ser mayor que 1<br />",
                     "Num resp");
         } else               // Si no es un entero positivo válido
            estado.añadeError("Error: el número de respuestas " + msg.getMensaje() +
                  "<br />", "Num_resp");

         // Comprueba que las instrucciones sean un código XHTML válido
         if (instrucciones != null && !instrucciones.equals("")){
            msg = ParserNeutro.compruebaInstruccionesXHTML(instrucciones);
            if (msg.isEstado() == false)
               estado.añadeError(msg.getMensaje(), "Instruc");
         }

         // Si el estado es todo correcto pasa a generar el objeto contenedor del ítem
         if ( estado.isEstado()){
            // Crea un objeto que contiene las características del ítem Choice a crear
            ChoiceXML chXML = new ChoiceXML(identificador, titulo, instrucciones,
                  pregunta, shuffle);
            // Guarda el objeto choiceXML en sesión para seguir utilizándolo luego
            HttpSession sesion = request.getSession();
            sesion.setAttribute("Choice", chXML);
            // Redirecciona a la página de introducir las respuestas indicando el
            // número de respuestas y si hay que mezclarlas aleatoriamente
```

```
            response.sendRedirect("respuestaschoice.jsp?Num_Resp=" + numResp +
                  "&Shuff=" + shuffle);
         }
      }
      // Si no están todos los atributos obligatorios comprueba los que fallan
      // para informar al usuario
      else{
         if (identificador.equals(""))
            estado.añadeError("Debe introducir un Identificador<br />", "Identif");
         if (titulo.equals(""))
            estado.añadeError("Debe introducir un Título<br />", "Tit");
         if (numResp.equals(""))
            estado.añadeError("Debe introducir un número de respuestas<br />",
                  "Num_resp");
      }

      // Si ha habido algún fallo pone los datos que ya ha introducido el usuario junto
      // a los mensajes de aviso y el foco en el primer fallo encontrado
      if (estado.isEstado()== false){
         request.setAttribute("Aviso", estado.getMensaje());
         request.setAttribute("Focus", estado.getFocusON());
         request.setAttribute("Identif", identificador);
         request.setAttribute("Tit", titulo);
         request.setAttribute("Instruc", instrucciones);
         request.setAttribute("Preg", pregunta);
         request.setAttribute("Num resp", numResp);
         request.setAttribute("Shuffle", new Boolean(shuffle));
      }
   } else {
      // Si es la primera vez que se visita la página borra el objeto choiceXML del
      // ámbito de sesión para iniciar desde cero el ítem y pone el foco en el primer
      // campo del formulario
      HttpSession sesion = request.getSession();
      sesion.removeAttribute("Choice");
      request.setAttribute("Focus", "Identif");
   }

   return true;
 }
}
```

Control.java

```
package control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import javax.servlet.ServletException;

/**
 * Interfaz de todos los componentes de control implícitos encargados de manejar la lógica
 * asociada a cada una de las páginas JSP. Su único método es el que ejecuta toda la lógica.
 * Todas las clases encargadas de manejar la lógica de las páginas JSP deben implementar esta
 * clase y su método para que pueda realizarse correctamente.
 *
 * @author David Domínguez
 *
 */
interface Control {

 /**
  * Realiza la lógica de control asociada a las JSPs. Deben implementarlo todas las clases
  * encargadas realizar esa lógica.
  *
  * @param   request          la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 boolean indicando si se deben seguir ejecutando el resto de
  *                         los filtros o no
  * @throws  IOException       si hay algún problema con el acceso a ficheros
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see     ControlFilter
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
```

```
  */
 public boolean doLogic(HttpServletRequest request, HttpServletResponse response)
       throws IOException, ServletException;
}
```

ControlFilter.java

```
package control;

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

/**
 * Filtro que se encarga de decidir qué componente de control se debe ejecutar en función de
 * la página JSP que esté siendo visitada. La clase implementa la interfaz <code>Filter</code>,
 * y como tal, se ejecutará antes de que la petición del cliente llegue al Servlet destino
 * (real o JSP compilado a Servlet). La clase implementa los tres métodos que tiene la interfaz
 * <code>Filter</code>. El método <code>doFilter</code> que se ejecuta con cada petición a
 * cualquier página del proyecto como está configurado en "web.xml", es el método de control de
 * la Aplicación Web, encargado de decidir qué clase contiene la lógica asociada a una página
 * JSP determinada.
 *
 * @author   David Domínguez
 * @see      javax.servlet.Filter
 */
public class ControlFilter implements Filter {

 /**
  * Objeto de configuración del Filtro.
  */
 protected FilterConfig config = null;

 /**
  * Método que arranca y configura el filtro. Es llamado sólo una vez cuando se carga el
  * filtro para usarlo con la aplicación web. Simplemente le da el valor inicial al objeto
  * <code>FilterConfig</code>.
  *
  * @param   filterConfig   Objeto con la configuración y datos de interés del filtro
  * @see  javax.servlet.Filter#init(javax.servlet.FilterConfig)
  */
 public void init(FilterConfig filterConfig) {
    config = filterConfig;
 }

 /**
  * Método centro de control de la Aplicación, que se encarga de decidir qué componente está
  * asociado a cada página JSP visitada (si lo tiene). Se ejecuta siempre que se haga una
  * petición al servidor, antes de que ésta llegue al Sevlet destino (o página JSP compilada
  * a Servlet). Este filtro busca una clase java que se llame igual que la dirección pedida
  * por el cliente (sin extensión) y con la primera letra en mayúsculas, que implemente la
  * interfaz <code>Control</code>. Si la encuentra crea una instancia y ejecuta el método
  * <code>doLogic</code>, encargado de manejar la lógica de negocio asociada a la página
  * visitada. Si <code>doLogic</code> devuelve true se siguen ejecutando los demás filtros
  * (si los hay) en la cadena hasta llegar al Servlet. Este método es el control del
  * proyecto, encargado de buscar la lógica a ejecutar asociada a cada página requerida. Al
  * final, si todo ha ido bien, se llama al siguiente filtro en la cadena pasándole la
  * petición y la respuesta.
  *
  * Si la dirección introducida no contiene un nombre con extensión se busca ejecutar el
  * método <code>doLogic</code> de una clase por defecto, "Default", que se puede definir
  * con un comportamiento por defecto para este tipo de direcciones.
  *
  * @param   req              petición del cliente. El método espera que sea una
  *                           petición http; en caso contrario lanza una excepción de
  *                           servlet
  * @param   res              respuesta a enviar al cliente
  * @param   chain            representa la cadena de filtros que se aplica a la
  *                           petición y a la respuesta actuales
  * @throws  IOException      si hay algún problema leyendo o escribiendo en disco. La
  *                           lanza el método "doLogic" de la clase que implementa la
  *                           interfaz "control" que se ejecute
  * @throws  ServletException  si el método "doLogic" de la clase que implementa la
  *                           interfaz "control" que se ejecute la lanza, o si la
```

```
  *                          petición recibida no es http, o si hay alguna excepción
  *                          creando la instancia adecuada de "control", que se lanza en
  *                          una excepción de este tipo
  * @see  javax.servlet.Filter#doFilter(javax.servlet.ServletRequest,
javax.servlet.ServletResponse, javax.servlet.FilterChain)
  * @see  Control#doLogic(HttpServletRequest, HttpServletResponse)
  */
 public void doFilter ( ServletRequest req, ServletResponse res,
       FilterChain chain) throws IOException, ServletException {

    // La petición debe ser de tipo HTTP para poder acceder a los métodos particulares de
    // ésta
    if (!(req instanceof HttpServletRequest)) {
       throw new ServletException("El Filtro requiere una petición HTTP.");
    }

    // Determina el nombre del compontente de control implícito, que será igual que el
    // nombre de la página sin extensión y con la primera letra en mayúsculas
    HttpServletRequest request = (HttpServletRequest)req;
    HttpServletResponse response = (HttpServletResponse)res;
    String uri = request.getRequestURI();
    int inicio = uri.lastIndexOf("/")+1;
    int fin = uri.lastIndexOf(".");
    String nombre = "default";       // Nombre por defecto si la uri no es adecuada
    if (inicio < fin) {
       nombre = uri.substring(inicio, fin);
    }
    if (Character.isLowerCase(nombre.charAt(0))){   // Pone la primera letra en mayúsculas
       nombre = Character.toUpperCase(nombre.charAt(0))+ nombre.substring(1);
    }
    boolean doFilter = true;      // Para seguir ejecutando la cadena de filtros si nada
                                  // va mal
    // intenta cargar y ejecutar un componente de control implícito.
    try {
       // Busca en el paquete control la clase asociada
       Object o = Class.forName("control." + nombre).newInstance();
       // Debe implementar la interfaz "Control"
       if (!(o instanceof Control)) {
          throw new ServletException("Clase control."+ nombre +
                " no implementa control.Control");
       }
       Control control = (Control)o;
       doFilter = control.doLogic(request, response); // Ejecuta la lógica asociada
    }
    catch (ClassNotFoundException e) {
       // Si la página jsp no tiene un componente de control implícito no hace nada.
       // Sigue adelante. Simplemente indica que no necesita la preparación lógica de los
       // datos que se presentan en la página
    }
    catch (InstantiationException e) {
       throw new ServletException(e);
    }
    catch (IllegalAccessException e) {
       throw new ServletException(e);
    }
    // sigue ejecutando los filtros subsiguientes si todo ha ido bien
    if (doFilter) {
       chain.doFilter(request, response);
    }
 }

 /**
  * Se llama cuando el filtro se descarga de la memoria, típicamente cuando la Aplicación
  * se descarga. En este caso no es necesario hacer nada al descargar el filtro.
  *
  * @see javax.servlet.Filter#destroy()
  */
 public void destroy() {
    // vacío
 }
}
```

Creartest.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import xml.test.*;

import utilidades.*;

/**
 * Componente de control implícito asociado a la página JSP "creartest.jsp". Es la lógica
 * asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea
 * llamada la página. Su función es comprobar que todos los parámetros iniciales de la toma de
 * datos para crear un nuevo test sean correctos, en cuyo caso redirecciona la respuesta
 * a la página de seleccionar los ítems a incluir en el test. La clase implementa la interfaz
 * <code>Control</code>, implementando su método <code>doLogic</code>, que es el único que
 * contiene, y que es el encargado de realizar la lógica asociada a la página "creartest.jsp".
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Creartest implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página "creartest.jsp".
  * Será llamado por el filtro <code>ControlFilter</code> en el caso de que detecte que la
  * página requerida es "creartest.jsp". Ya que es llamado desde un filtro, será ejecutado
  * siempre que llegue una petición de un cliente y antes de que la petición llegue al
  * Servlet (página JSP compilada a Servlet) que sirve la respuesta al cliente. Comprueba
  * uno por uno todos los parámetros, y, en caso de que sean correctos, crea un objeto <code>
  * AssessmentTest</code> para guardar los datos asociados al test y lo guarda en el ámbito
  * de sesión para utilizarlos más adelante para terminar de crear el test.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see control.Control#doLogic(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)
  * @see  control.ControlFilter
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see     utilidades.EstadoWeb
  * @see     utilidades.MensajeEstado
  * @see     utilidades.EnteroPositivo
  * @see     xml.test.AssessmentTest
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Comprueba si es la primera vez que se visita la página
    if (request.getParameter("Form") != null){

       // Caracteres reservados no permitidos
       final String reservados = ";/?:@&=+$,\\*\"<>|";

       String identificador = request.getParameter("Identif"); // Identificador del test
       String titulo = request.getParameter("Tit");       // Título dele test
       String instrucciones = request.getParameter("Instruc"); // Instrucciones del test
       // Permitir revisión del test al finalizarlo
       String rev = request.getParameter("allowReview");
       String selec = request.getParameter("Select");  // Número de ítems a mostrar
       String Shuff = request.getParameter("Shuffle");    // Mezclar los ítems o no

       // Estado de la petición web. Almacena si son correctos los parámetros, los
       // mensajes de aviso al usuario y dónde colocar el foco en la página
       EstadoWeb estado = new EstadoWeb();
```

```
boolean shuffle = false;      // Valor por defecto del parámetro
if ( Shuff != null){       // Interpreta el valor marcado en la página
   if (Shuff.equalsIgnoreCase("ON"))
      shuffle = true;
}

// Comprueba que estén introducidos todos los atributos obligatorios
if (identificador != null && !identificador.equals("") &&
      titulo != null && !titulo.equals("") &&
      rev != null && !rev.equals("")){

   // Comprueba que los caracteres del identificador estén permitidos
     for(int i = 0; i < identificador.length() && estado.isEstado() == true; i++) {
      char c = identificador.charAt(i);   // Lee el siguiente carácter
      int marca = reservados.indexOf(c);
      if (marca >= 0 ){
         estado.añadeError("Error: El carácter '" + reservados.charAt(marca) +
               "' en la posición " + (i+1) + " del Identificador no está " +
               "permitido<br />", "Identif");
      }
     }

     // Permitir o no la revisión del documento
   boolean revision = Boolean.parseBoolean(rev);

   // Comprueba que el número de ítems a seleccionar esté vacío o sea un valor
   // válido
   EnteroPositivo seleccion;
   if (selec == null || selec.equals("")){
      seleccion = new EnteroPositivo("0");
      seleccion.esPositivo();       // Para actualizar el valor entero
   }
   else{
      seleccion = new EnteroPositivo(selec);
      MensajeEstado msg = seleccion.esPositivo();
      if(!msg.isEstado()){  // Si no es un número válido y mayor o igual que cero
         estado.añadeError("Error: el número de ítems a seleccionar " +
               msg.getMensaje() + "<br />", "Select");
      }
   }

   // Comprueba que las instrucciones sean un código XHTML válido
   if (instrucciones != null && !instrucciones.equals("")){
      MensajeEstado msg = ParserNeutro.compruebaInstruccionesXHTML(instrucciones);
      if (msg.isEstado() == false)
         estado.añadeError(msg.getMensaje(), "Instruc");
   }

   // Si el estado es todo correcto pasa a generar el objeto contenedor del test
   if ( estado.isEstado()){
      // Método de ordenación
      Ordering orden = null;
      if (shuffle == true)
         orden = new Ordering (shuffle);
      // Método de seleccionar ítems
      Selection select = null;
      // Si se selecciona algún número de ítems del total, es sin reemplazamiento,
      // ya que los ítems son constantes, no contienen variables, y no tiene
      // sentido repetirlos en una sección
      if (seleccion.getEnteroPositivo() != 0)
         select = new Selection(seleccion.getEnteroPositivo(), false);
      // Se inicializa el objeto Sección del test excepto las referencias a los
      // ítems, con un identificador válido que es válido, por eso no se comprueba
      AssessmentSection seccion =
         new AssessmentSection(new Identificador("seccion1"), titulo, true,
               select, orden, instrucciones);
      // Objeto de control para permitir la revisión o no
      ItemSessionControl control = new ItemSessionControl(revision);
      // Se inicializa el objeto Parte de test con la Sección anterior y con un
      // identificador válido, que como sólo hay una parte, no se realiza ninguna
      // comprobación, el objeto control anterior, y estableciendo la navegación y
      // la presentación a esos valores que son los únicos que soporta esta
      // implementación
      TestPart parte = new TestPart(new Identificador("parte1"), "nonlinear",
            "simultaneous", control, seccion);
      // Objeto Test inicializado con los datos proporcionados por el usuario
```

```
            // y los objetos creados más arriba
            AssessmentTest test = new AssessmentTest(identificador, titulo, parte);
            // Guarda el objeto AsessmentTest en sesión para seguir utilizándolo luego
            HttpSession sesion = request.getSession();
            sesion.setAttribute("Test1", test);
            // Se borra de sesión el objeto con los ítems a añadir para empezar de cero
            // añadiendo los ítems en la página siguiente
            sesion.removeAttribute("Items A Añadir");
            // Redirecciona a la página de seleccionar los ítems a incluir en el Test,
            // pasando por la página de seleccionar la asignatura primero
            response.sendRedirect("seleccionasignatura.jsp?Destino=" +
                  "seleccionaritems.jsp");
         }
      }
      // Si no están todos los atributos obligatorios comprueba los que fallan
      // para informar al usuario
      else{
         if (identificador.equals(""))
            estado.añadeError("Debe introducir un Identificador<br />", "Identif");
         if (titulo.equals(""))
            estado.añadeError("Debe introducir un Título<br />", "Tit");
         if (rev == null || rev.equals(""))
            estado.añadeError("Debe marcar si desea permitir la revisión del test " +
                  "al finalizarlo<br />", "Identif");
      }

      // Si ha habido algún fallo pone los datos que ya ha introducido el usuario junto
      // a los mensajes de aviso y el foco en el primer fallo encontrado
      if (estado.isEstado()== false){
         request.setAttribute("Aviso", estado.getMensaje());
         request.setAttribute("Focus", estado.getFocusON());
         request.setAttribute("Identif", identificador);
         request.setAttribute("Tit", titulo);
         request.setAttribute("Instruc", instrucciones);
         request.setAttribute("allowReview", rev);
         request.setAttribute("Select", selec);
         request.setAttribute("Shuffle", new Boolean(shuffle));
      }
   } else {
      // Si es la primera vez que se visita la página borra el objeto AssessmentTest del
      // ámbito de sesión para iniciar desde cero el test y pone el foco en el primer
      // campo del formulario
      HttpSession sesion = request.getSession();
      sesion.removeAttribute("Test");
      request.setAttribute("Focus", "Identif");
   }

   return true;
 }
}
```

Explorador.java

```
package control;

import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.Vector;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import utilidades.ListaArchivos;
import utilidades.UrlUtils;

/**
 * Componente de control implícito asociado a la página JSP "explorador.jsp". Es la lógica
 * asociada a esa página JSP. Su método <code>doLogic</code> se ejecutará siempre que sea
 * llamada la página. La página es el frame de la mitad izquierda de la página
 * "buscadorimagenes.jsp", y es el explorador de imágenes y directorios para seleccionar una
 * imagen. Se encarga de la navegación por los directorios de imágenes, mostrando los
```

```
 * subdirectorios y las imágenes que se encuentran en el directorio actual, permitiendo entrar
 * en subdirectorios y volver atrás, previsualizando a su vez las imágenes en el frame derecho
 * de la página para elegir una.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Explorador implements Control {

 /**
  * Lógica control asociada a la página "explorador.jsp". Se ejecuta siempre que haya
  * alguna petición con destino esa página. Es el frame izquierdo de la página
  * "buscadorimagenes.jsp". Ya que es llamado desde un filtro, será ejecutado siempre que
  * llegue una petición de un cliente y antes de que la petición llegue al Servlet (página
  * JSP compilada a Servlet) que sirve la respuesta al cliente. Se encarga de mostrar al
  * usuario todos los directorios que se encuentren por debajo del de las Imágenes por
  * defecto (nunca subir más arriba), permitiéndole ver las imágenes de los tipos que se
  * encuentren definidos en el método <code>ListaImagenes</code> en esos directorios en el
  * frame derecho ("imagen.html") y navegar por el árbol de subdirectorios por debajo del
  * directorio inicial de imágenes, sin permitir subir más arriba de éste.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se le envíe la respuesta correctamente
  * @throws  IOException       si hay algún problema en el acceso a las imágenes o los
  *                         directorios
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see     utilidades.ListaArchivos
  * @see     utilidades.UrlUtils#encodeURLReserved(String)
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Contexto del Servlet
    ServletContext sc =  request.getSession().getServletContext();
    // Directorio a mostrar en este momento
    String dirImagenName = request.getParameter("DirectorioActual");
    // Directorio base de trabajo (guardado en "web.xml")
    String dirTrabajo = sc.getInitParameter("Directorio Trabajo");
    // Directorio de asignatura si venimos de la página de selección de asignatura
    String dirAsignatura = request.getParameter("DirectorioAsignatura");
    // Directorio base de las imágenes. El parámetro está definido en "web.xml"
    String dirInicial = sc.getInitParameter("Directorio Imágenes");
    // Directorio padre del actual (null si estamos en el base)
    String dirSuperior = null;

    if (dirImagenName == null){
       // Es la primera vez que entra en la página, así que va al directorio de imágenes
       // inicial
       dirImagenName = "/" + dirAsignatura + dirInicial;
    }

    // Busca la segunda localización del carácter '/' para ver si estamos en el inicio
    int loc = dirImagenName.indexOf('/', 1);
    // Si longitud de la dirección actual es distinta de la que hay del inicio al segundo
    // carácter '/' más la del directorio inicial de los ítems no estamos en el base, y por
    // lo tanto mostramos la opción de ir al directorio superior
    if (loc + dirInicial.length() != dirImagenName.length())
       // Si no estamos en el inicial guarda el nombre del directorio padre para volver
       // atrás
       dirSuperior = dirImagenName.substring(0, dirImagenName.lastIndexOf('/'));

    // Guarda en request los nombres del directorio actual y el del directorio padre
    request.setAttribute("DirectorioActual", dirImagenName);
    request.setAttribute("DirectorioSuperior", dirSuperior);

    // Comprueba que existe el directorio base de las imágenes
    File dirImagenes = new File(sc.getRealPath(dirTrabajo) + "/" + dirImagenName);
    if (!dirImagenes.exists())
```

```
        if (!dirImagenes.mkdir())
           throw new IOException("Error: No se pudo crear el directorio base de las " +
                 "Imágenes");

    // Lista los nombres de los subdirectorios del directorio y lo guarda en request
    Vector /* String */ dirNames = ListaArchivos.ListaDirectorios(dirImagenes);
    request.setAttribute("NombresDirectorios",dirNames);

    // Lista los nombres de las imágenes del directorio y lo guarda en request
    Vector /* String */ imagNames = ListaArchivos.ListaImagenes(dirImagenes);
    // Guarda una versión de los nombres de los archivos de imágenes con todos los
    // caracteres ilegales codificados
    Vector /* String */ imagNamesCodif = new Vector();
    for (int i = 0; i < imagNames.size(); i++){
       imagNamesCodif.add(UrlUtils.encodeURLReserved((String)imagNames.get(i)));
    }
    // URL absoluta del directorio de las imágenes ya codificado
    String dirURL;
    // string con la url que ha realizado el request
    String urlS = request.getRequestURL().toString();
    // url sin el nombre de la página actual, hasta el último carácter '/'
    URL url = new URL(urlS.substring(0, urlS.lastIndexOf('/')));
    // Directorio de las imágenes desde el directorio base del proyecto
    String dir = "/" + dirTrabajo + dirImagenName + "/";
    // Dirección url absoluta del directorio a mostrar
    String urlAbs = url.toString() + dir;
    // Se codifican los caracteres ilegales de la url (excepto los reservados como los
    // separadores, que se dejan igual)
    dirURL = UrlUtils.encodeURL(urlAbs);
    // Se guarda la URL codificada del directorio de las imágenes y los nombres de las
    // imágenes codificadas y sin codificar
    request.setAttribute("NombresImagenes", imagNames);
    request.setAttribute("NombresImagenesCodif", imagNamesCodif);
    request.setAttribute("URLImagenes", dirURL);

    return true;
 }
}
```

Fileupload.java

```
package control;

import java.io.*;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.commons.fileupload.servlet.*;
import org.apache.commons.fileupload.*;
import org.apache.commons.fileupload.disk.*;
import org.apache.commons.io.FilenameUtils;

import utilidades.ListaArchivos;

/**
 * Componente de control implícito asociado a la página JSP "fileupload.jsp". Es la lógica
 * asociada a esa página JSP. Su método <code>doLogic</code> se ejecutará siempre que sea
 * llamada la página. Su función es la de permitir al usuario seleccionar un directorio en el
 * servidor al que subir las imágenes seleccionadas en la página.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Fileupload implements Control {

 /**
  * Lógica de control asociada a la página "fileupload.jsp", que se encarga de subir un
  * archivo al servidor. Será llamado por el filtro <code>ControlFilter</code> en el caso de
  * que detecte que la página requerida es "fileupload.jsp". Ya que es llamado desde un
  * filtro, será ejecutado siempre que llegue una petición de un cliente y antes de que la
  * petición llegue al Servlet (página JSP compilada a Servlet) que sirve la respuesta al
```

```
 * cliente. Se encarga de listar los subdirectorios que hay en el directorio de imágenes y
 * permitir entrar en ellos y volver atrás. Puede recibir información del resultado del
 * intento de crear un nuevo directorio para informar en "fileupload.jsp". Una vez
 * seleccionado el directorio debe subir el archivo, comprobando que sea de imagen, elegido
 * por el usuario y almacenarlo en el susodicho directorio, avisando al usuario del éxito o
 * fracaso de tal acción.
 *
 * @param   request           la petición http enviada por el cliente
 * @param   response       la respuesta http a enviar desde el servidor
 * @return                 true para que se sigan ejecutando los demás filtros del
 *                         servidor y se le envíe la respuesta correctamente
 * @throws  IOException       si hay algún problema con el acceso a los directorios o
 *                         guardando en disco las imágenes
 * @throws  ServletException  si hay algún problema accediendo a request o response
 * @see
control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
 * @see     javax.servlet.http.HttpServletRequest
 * @see     javax.servlet.http.HttpServletResponse
 * @see  control.ControlFilter
 * @see     utilidades.ListaArchivos
 * @see     org.apache.commons.fileupload.servlet.ServletRequestContext
 * @see     org.apache.commons.fileupload.servlet.ServletFileUpload
 * @see     org.apache.commons.fileupload.disk.DiskFileItemFactory
 * @see     org.apache.commons.io.FilenameUtils
 */
public boolean doLogic(HttpServletRequest request,
      HttpServletResponse response) throws IOException, ServletException {

   // Contexto del Servlet
   ServletContext sc =  request.getSession().getServletContext();
   // Directorio base de trabajo (guardado en "web.xml")
   String dirTrabajo = sc.getInitParameter("Directorio Trabajo");
   // Directorio de asignatura si venimos de la página de selección de asignatura
   String dirAsignatura = request.getParameter("DirectorioAsignatura");
   // Directorio a mostrar en este momento
   String dirImagenName = request.getParameter("DirectorioActual");
   // Directorio inicial de las imágenes (guardado en "web.xml")
   String dirInicial = sc.getInitParameter("Directorio Imágenes");
   // Aviso recibido de otra página a mostrar en esta
   String aviso = request.getParameter("Aviso");
   // Informacion recibida de otra página a mostrar en esta
   String informacion = request.getParameter("Informacion");
   // Directorio superior del actual (null si estamos en el inicial)
   String dirSuperior = null;

   if (dirImagenName == null){
      // Es la primera vez que entra en la página, así que va al directorio inicial de
      // imágenes
      dirImagenName = "/" + dirAsignatura + dirInicial;
   }

   // Busca la segunda localización del carácter '/' para ver si estamos en el inicio
   int loc = dirImagenName.indexOf('/', 1);
   // Si longitud de la dirección actual es distinta de la que hay del inicio al segundo
   // carácter '/' más la del directorio inicial de los ítems no estamos en el base, y por
   // lo tanto mostramos la opción de ir al directorio superior
   if (loc + dirInicial.length() != dirImagenName.length())
      // Si no estamos en el inicial guarda el nombre del directorio padre para volver
      // atrás
      dirSuperior = dirImagenName.substring(0, dirImagenName.lastIndexOf('/'));

   // Guarda en request los nombres del directorio actual y el del directorio padre
   request.setAttribute("DirectorioActual", dirImagenName);
   request.setAttribute("DirectorioSuperior", dirSuperior);

   // Comprueba que existe el directorio base de las imágenes
   File dirImagenes = new File(sc.getRealPath(dirTrabajo) + "/" + dirImagenName);
   if (!dirImagenes.exists())
      if (!dirImagenes.mkdir())
         throw new IOException("Error: No se pudo crear el directorio base de las " +
               "Imágenes");

   // Busca los directorios del directorio actual
   Vector /* String */ dirNames = ListaArchivos.ListaDirectorios(dirImagenes);
   request.setAttribute("NombresDirectorios",dirNames);
```

```
    // Comprobamos que tenemos una petición de subida de archivo (petición multipart)
    ServletRequestContext rqctxt = new  ServletRequestContext(request);
    boolean isMultipart = ServletFileUpload.isMultipartContent(rqctxt);

    if (isMultipart){
       // Crea un factory para archivos que guardar en disco, dejando por defecto el
       // límite de tamaño para archivos en disco y el directorio temporal el del sistema
       DiskFileItemFactory factory = new DiskFileItemFactory();

       // Crea un manejador de archivos subidos
       ServletFileUpload upload = new ServletFileUpload(factory);

       try {
          // Crea un List con los archivos subidos (debe de ser sólo uno)
          List /* FileItem */ items = upload.parseRequest(request);

          Iterator iter = items.iterator();
          while (iter.hasNext()) {
             FileItem fItem = (FileItem) iter.next();
             // Si el ítem no es un campo de formulario (será un archivo)
             if (!fItem.isFormField()) {
                String fName = FilenameUtils.getName(fItem.getName());
                // Comprueba que sea un archivo de imagen soportado
                String fExtension = FilenameUtils.getExtension(fName);
                if (fExtension.equalsIgnoreCase("bmp") ||
                      fExtension.equalsIgnoreCase("gif") ||
                      fExtension.equalsIgnoreCase("jpg") ||
                      fExtension.equalsIgnoreCase("jpeg") ||
                      fExtension.equalsIgnoreCase("png")){
                   File uploadedFile = new File(dirImagenes,fName);
                   // Comprueba que no exista ya el nombre de archivo
                   if (!uploadedFile.exists()){
                      fItem.write(uploadedFile);
                      informacion = "Imagen subida con éxito";
                   }else{
                      aviso = "Ese nombre de archivo ya existe";
                   }
                }
                else{
                   aviso = "Tipo de archivo incorrecto";
                }
             }
          }
       }
       catch (FileUploadException e){
          // Si la clase ServletFileUpload no ha podido procesar la petición
          throw new ServletException("Error procesando la petición", e);
       }
       catch (Exception e){
          // Si hay error al escribir el archivo o procesando la respuesta
          throw new ServletException ("Error guardando el archivo de imagen", e);
       }
    }

    // Avisos e información para el usuario de la página
    request.setAttribute("Aviso", aviso);
    request.setAttribute("Informacion", informacion);

    return true;
 }
}
```

Fin.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
```

```
 * Componente de control implícito asociado a la página JSP "fin.jsp". Es la lógica asociada a
 * esa página JSP. Su método <code>doLogic</code> se ejecutará siempre que sea llamada la
 * página. Sólo se encarga de disponer en la página los mensajes de éxito o fracaso creando el
 * ítem o el test XML del QTI.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Fin implements Control {

 /**
  * Será llamado por el filtro "ControlFilter" en el caso de que detecte que la página
  * requerida es "fin.jsp". Ya que es llamado desde un filtro, será ejecutado siempre que
  * llegue una petición de un cliente y antes de que la petición llegue al Servlet (página
  * JSP compilada a Servlet) que sirve la respuesta al cliente. Se encarga de avisar al
  * usuario del éxito o fracaso de la creación del ítem XML.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se le envíe la respuesta correctamente
  * @throws  IOException       no debe de darse el caso en esta implementación de Control
  * @throws  ServletException  no debe de darse el caso en esta implementación de Control
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Recoge los valores del estado y el mensaje para el usuario, que será información o
    // aviso de error en función del estado
    String mensaje = request.getParameter("Mensaje");
    boolean estado = Boolean.valueOf(request.getParameter("Estado")).booleanValue();

    if (estado){
       // Si todo ha ido bien creando el ítem
       request.setAttribute("Informacion", mensaje);
       request.setAttribute("Titulo", "Éxito");
    } else {
       // Si ha habido un error se avisa al usuario
       request.setAttribute("Aviso", mensaje);
       request.setAttribute("Titulo", "Error");
    }

    return true;
 }
}
```

Finitem.java

```
package control;

import java.io.File;
import java.io.IOException;
import java.util.Vector;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.*;

import xml.items.AssessmentItem;

import utilidades.*;

/**
 * Componente de control implícito asociado a la página JSP "finitem.jsp". Es la lógica
 * asociada a esa página JSP. Su método <code>doLogic</code> se ejecutará siempre que sea
 * llamada la página. Permite al usuario elegir el directorio dentro del de los ítems donde
 * almacenar el ítem XML y lo intenta crear y almacenar en disco.
 *
```

```
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Finitem implements Control {

 /**
  * Será llamado por el filtro <code>ControlFilter</code> en el caso de que detecte que la
  * página requerida es "fin.jsp". Ya que es llamado desde un filtro, será ejecutado
  * siempre que llegue una petición de un cliente y antes de que la petición llegue al
  * Servlet (página JSP compilada a Servlet) que sirve la respuesta al cliente. Trata de
  * crear el ítem XML almacenado en sesión. Saca el objeto, que debe ser subclase de <code>
  * AssessmentItem</code>, de sesión y lo creará en la carpeta que elija el usuario. Si ya
  * existe ese nombre de archivo entonces permite modificar el identificador del ítem e
  * introducir un nuevo nombre. Permite al usuario crear nuevos directorios y navegar por
  * los directorios. Puede recibir información de la página de crear nuevos directorios.
  * Luego redirecciona la respuesta a "fin.jsp" que informa al usuario del éxito o fracaso
  * del proceso.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se le envíe la respuesta correctamente
  * @throws  IOException       si hay algún problema accediendo a los directorios o
  *                         leyendo su contenido o al escribir el archivo en disco
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  java.io.File
  * @see  xml.items.AssessmentItem
  * @see  utilidades.MensajeEstado
  * @see     utilidades.ListaArchivos
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Contexto del Servlet
    ServletContext sc =  request.getSession().getServletContext();
    // Directorio base de trabajo (guardado en "web.xml")
    String dirTrabajo = sc.getInitParameter("Directorio Trabajo");
    // Directorio de asignatura si venimos de la página de selección de asignatura
    String dirAsignatura = request.getParameter("DirectorioAsignatura");
    // Directorio seleccionado a mostrar en este momento
    String dirItemActual = request.getParameter("DirectorioActual");
    // Directorio inicial de los ítems (guardado en "web.xml")
    String dirInicial = sc.getInitParameter("Directorio Items");
    // Directorio superior del actual (null si estamos en el inicial)
    String dirSuperior = null;
    // Aviso recibido de otra página a mostrar en esta. Se utiliza también para los avisos
    // de la propia página. Se sobreescribe porque los avisos no se pueden dar a la vez.
    String aviso = request.getParameter("Aviso");
    // Informacion recibida de otra página a mostrar en esta
    String informacion = request.getParameter("Informacion");

    // Si vamos a otra página o seguimos en la misma
    boolean salida = false;
    // Si se permite introducir un nuevo identificador
    boolean muestraNuevoIdent = false;

    if (dirItemActual == null){
       // Primera vez que se entra en la página, así que establece el directorio de inicio
       dirItemActual = "/" + dirAsignatura + dirInicial;
       // Comprueba si existe el directorio base de los ítems, y si no existe intenta
       // crearlo
       File dirItems = new File(sc.getRealPath(dirTrabajo) + "/" + dirAsignatura +
             dirInicial);
       if (!dirItems.exists())
          if (!dirItems.mkdir())
             throw new IOException("Error: No se pudo crear el directorio base de " +
                   "los Items");
    }

    // Si "Form" es distinto de null se ha pulsado el botón de crear ítem
```

```
if (request.getParameter("Form") != null){

   HttpSession sesion = request.getSession();
   // Si el objeto sesión no es nuevo la sesión no ha expirado
   if (!sesion.isNew()){
      // Saca de sesión el Assessment ítem creado para crear el archivo XML del ítem
      AssessmentItem aItem = (AssessmentItem)sesion.getAttribute("AItem");
      String nombre = aItem.getIdentificador(); // Nombre del archivo XML a crear

      // Estado de la comprobación del identificador del ítem
      boolean correcto = true;

      // Si se ha introducido un nuevo identificador crea el archivo con ese nombre,
      // comprobando antes que sea un nombre correcto
      String identificador = request.getParameter("Identif");
      if (identificador != null && !identificador.equals("")){

         // Caracteres reservados no permitidos
         final String reservados = ";/?:@&=+$,\\*\"<>|";

         // Comprueba que los caracteres del nuevo identificador estén permitidos
           for(int i = 0; i < identificador.length() && correcto == true; i++) {
            char c = identificador.charAt(i);   // Lee el siguiente carácter
            int marca = reservados.indexOf(c);
            if (marca >= 0 ){
               correcto = false;
               aviso = "Error: El carácter '" + reservados.charAt(marca) +
                     "' en la posición " + (i+1) + " del Identificador no " +
                     "está permitido<br />";
               muestraNuevoIdent = true;// Permitir introducir en un nuevo nombre
               // Guarda en request si se muestra el campo de introducir un nuevo
               // identificador
               request.setAttribute("Nuevo", Boolean.toString(muestraNuevoIdent));

            }
           }
           if (correcto == true)
            nombre = identificador;
      }

      // Si el nombre del archivo es correcto se intenta crear el archivo del ítem
      if (correcto == true){
         File item = new File(sc.getRealPath(dirTrabajo) + dirItemActual + "/" +
               nombre + ".xml");
         if (item.exists()){        // Si ya existe un ítem con ese nombre
            aviso = "Ya existe un ítem con ese identificador, debe " +
                  "seleccionar otro<br />";
            // Permitir introducir en la página un nuevo nombre
            muestraNuevoIdent = true;
            // Guarda en request si se muestra el campo de introducir un nuevo
            // identificador
            request.setAttribute("Nuevo", Boolean.toString(muestraNuevoIdent));
         } else {
            // Si no existe crea el ítem XML y va a fin.jsp para mostrar el
            // resultado
            aItem.setIdentificador(nombre);
            MensajeEstado msg;
            String url = request.getRequestURL().toString();
            // URL base del proyecto web
            String urlBase = url.substring(0, url.lastIndexOf('/'));
            msg = aItem.creaItemXML(sc.getRealPath(dirTrabajo) + dirItemActual +
                  "/", sc, urlBase);
            response.sendRedirect("fin.jsp?Estado=" + msg.isEstado() +
                  "&Mensaje=" + msg.getMensaje());
            salida = true;          // Sale de esta página
         }
      }

   } else{        // Si la sesión ha expirado
      response.sendRedirect("errorfinsesion.jsp");
      salida = true;             // Sale de esta página
   }
}

// Si se sigue en esta página muestra los directorios para elegir uno
if (salida != true){
```

```
        // Busca la segunda localización del carácter '/' para ver si estamos en el inicio
        int loc = dirItemActual.indexOf('/', 1);
        // Si longitud de la dirección actual es distinta de la que hay del inicio al
        // segundo  carácter '/' más la del directorio inicial de los ítems no estamos en
        // el base, y por lo tanto mostramos la opción de ir al directorio superior
        if (loc + dirInicial.length() != dirItemActual.length())
           // Si no estamos en el inicial guarda el nombre del directorio padre para
           // poder volver atrás
           dirSuperior = dirItemActual.substring(0, dirItemActual.lastIndexOf('/'));

        // Guarda en request los nombres del directorio actual y el del directorio padre
        request.setAttribute("DirectorioActual", dirItemActual);
        request.setAttribute("DirectorioSuperior", dirSuperior);

        // Lista los nombres de los directorios del directorio actual y los guarda en
        // request
        File dirActual = new File(sc.getRealPath(dirTrabajo) + dirItemActual);
        Vector /* String */ dirNames = ListaArchivos.ListaDirectorios(dirActual);
        request.setAttribute("NombresDirectorios",dirNames);
    }

    // Avisos e información para el usuario de la página
    request.setAttribute("Aviso", aviso);
    request.setAttribute("Informacion", informacion);

    return true;
 }
}
```

Fintest.java

```
package control;

import java.io.File;
import java.io.IOException;
import java.util.Vector;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import xml.test.AssessmentTest;

import utilidades.*;

/**
 * Componente de control implícito asociado a la página JSP "fintest.jsp". Es la lógica
 * asociada a esa página JSP. Su método <code>doLogic</code> se ejecutará siempre que sea
 * llamada la página. Permite al usuario elegir el directorio dentro del de test donde
 * almacenar el test XML y lo intenta crear y almacenar en disco.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Fintest implements Control {

 /**
  * Será llamado por el filtro <code>ControlFilter</code> en el caso de que detecte que la
  * página requerida es "fintest.jsp". Ya que es llamado desde un filtro, será ejecutado
  * siempre que llegue una petición de un cliente y antes de que la petición llegue al
  * Servlet (página JSP compilada a Servlet) que sirve la respuesta al cliente. Trata de
  * crear el test XML almacenado en sesión. Saca el objeto, que debe ser instancia de <code>
  * AssessmentTest</code>, de sesión y lo creará en la carpeta que elija el usuario. Si ya
  * existe ese nombre de archivo entonces permite modificar el identificador del test e
  * introducir un nuevo nombre. Permite al usuario crear nuevos directorios y navegar por
  * los directorios. Puede recibir información de la página de crear nuevos directorios.
  * Luego redirecciona la respuesta a "fin.jsp" que informa al usuario del éxito o fracaso
  * del proceso.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
```

```
  * @return                true para que se sigan ejecutando los demás filtros del
  *                         servidor y se le envíe la respuesta correctamente
  * @throws  IOException       si hay algún problema accediendo a los directorios o
  *                         leyendo su contenido o al escribir el archivo en disco
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  java.io.File
  * @see  xml.test.AssessmentTest
  * @see  utilidades.MensajeEstado
  * @see     utilidades.ListaArchivos
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Contexto del Servlet
    ServletContext sc =  request.getSession().getServletContext();
    // Directorio base de trabajo (guardado en "web.xml")
    String dirTrabajo = sc.getInitParameter("Directorio Trabajo");
    // Directorio de asignatura si venimos de la página de selección de asignatura
    String dirAsignatura = request.getParameter("DirectorioAsignatura");
    // Directorio seleccionado a mostrar en este momento
    String dirItemActual = request.getParameter("DirectorioActual");
    // Directorio inicial de los tests (guardado en "web.xml")
    String dirInicial = sc.getInitParameter("Directorio Tests");
    // Directorio superior del actual (null si estamos en el inicial)
    String dirSuperior = null;
    // Aviso recibido de otra página a mostrar en esta. Se utiliza también para los avisos
    // de la propia página. Se sobreescribe porque los avisos no se pueden dar a la vez.
    String aviso = request.getParameter("Aviso");
    // Informacion recibida de otra página a mostrar en esta
    String informacion = request.getParameter("Informacion");

    // Si vamos a otra página o seguimos en la misma
    boolean salida = false;
    // Si se permite introducir un nuevo identificador
    boolean muestraNuevoIdent = false;

    if (dirItemActual == null){
       // Primera vez que se entra en la página, así que establece el directorio de inicio
       dirItemActual = "/" + dirAsignatura + dirInicial;
       // Comprueba si existe el directorio base de los tests, y si no existe intenta
       // crearlo
       File dirItems = new File(sc.getRealPath(dirTrabajo) + "/" + dirAsignatura +
             dirInicial);
       if (!dirItems.exists())
          if (!dirItems.mkdir())
             throw new IOException("Error: No se pudo crear el directorio base de " +
                   "los Tests");
    }

    // Si "Form" es distinto de null se ha pulsado el botón de crear test
    if (request.getParameter("Form") != null){

       HttpSession sesion = request.getSession();
       // Si el objeto sesión no es nuevo la sesión no ha expirado
       if (!sesion.isNew()){
          // Saca de sesión el AssessmentTest creado para crear el archivo XML del test
          AssessmentTest test = (AssessmentTest)sesion.getAttribute("Test2");
          String nombre = test.getIdentificador();  // Nombre del archivo XML a crear

          // Estado de la comprobación del identificador del ítem
          boolean correcto = true;

       // Si se ha introducido un nuevo identificador crea el archivo con ese nombre,
          // comprobando antes que sea un nombre correcto
          String identificador = request.getParameter("Identif");
          if (identificador != null && !identificador.equals("")){

             // Caracteres reservados no permitidos
             final String reservados = ";/?:@&=+$,\\*\"<>|";

             // Comprueba que los caracteres del nuevo identificador estén permitidos
```

```
            for(int i = 0; i < identificador.length() && correcto == true; i++) {
             char c = identificador.charAt(i);   // Lee el siguiente carácter
             int marca = reservados.indexOf(c);
             if (marca >= 0 ){
                correcto = false;
                aviso = "Error: El carácter '" + reservados.charAt(marca) +
                      "' en la posición " + (i+1) + " del Identificador no " +
                      "está permitido<br />";
                muestraNuevoIdent = true;// Permitir introducir en un nuevo nombre
                // Guarda en request si se muestra el campo de introducir un nuevo
                // identificador
                request.setAttribute("Nuevo", Boolean.toString(muestraNuevoIdent));

             }
            }
            if (correcto == true)
             nombre = identificador;
      }

      // Si el nombre del archivo es correcto se intenta crear el archivo del test
      if (correcto == true){
         File testFile = new File(sc.getRealPath(dirTrabajo) + dirItemActual + "/" +
               nombre + ".xml");
         if (testFile.exists()){       // Si ya existe un ítem con ese nombre
            aviso = "Ya existe un test con ese identificador, debe " +
                  "seleccionar otro<br />";
            // Permite introducir en la página un nuevo nombre
            muestraNuevoIdent = true;
            // Guarda en request si se muestra el campo de introducir un nuevo
            // identificador
            request.setAttribute("Nuevo", Boolean.toString(muestraNuevoIdent));
         } else {
            // Si no existe crea el test XML y va a fin.jsp para mostrar el resultado
            test.setIdentificador(nombre);
            test.getTestPart().getSeccion().creaRefItems(testFile);
            MensajeEstado msg = test.creaAssessmentTest(sc.getRealPath(dirTrabajo) +
                  dirItemActual + "/", sc);
            response.sendRedirect("fin.jsp?Estado=" + msg.isEstado() + "&Mensaje=" +
                  msg.getMensaje());
            salida = true;          // Sale de esta página
         }
      }

   } else{        // Si la sesión ha expirado
      response.sendRedirect("errorfinsesion.jsp");
      salida = true;             // Sale de esta página
   }
}

// Si se sigue en esta página muestra los directorios para elegir uno
if (salida != true){
   // Busca la segunda localización del carácter '/' para ver si estamos en el inicio
   int loc = dirItemActual.indexOf('/', 1);
   // Si longitud de la dirección actual es distinta de la que hay del inicio al
   // segundo  carácter '/' más la del directorio inicial de los ítems no estamos en
   // el base, y por lo tanto mostramos la opción de ir al directorio superior
   if (loc + dirInicial.length() != dirItemActual.length())
      // Si no estamos en el inicial guarda el nombre del directorio padre para
      // poder volver atrás
      dirSuperior = dirItemActual.substring(0, dirItemActual.lastIndexOf('/'));

   // Guarda en request los nombres del directorio actual y el del directorio padre
   request.setAttribute("DirectorioActual", dirItemActual);
   request.setAttribute("DirectorioSuperior", dirSuperior);

   // Lista los nombres de los directorios del directorio actual y los guarda en
   // request
   File dirActual = new File(sc.getRealPath(dirTrabajo) + dirItemActual);
   Vector /* String */ dirNames = ListaArchivos.ListaDirectorios(dirActual);
   request.setAttribute("NombresDirectorios",dirNames);
}

// Avisos e información para el usuario de la página
request.setAttribute("Aviso", aviso);
request.setAttribute("Informacion", informacion);
```

```
    return true;
 }
}
```

Gapmatch.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import xml.items.GapmatchXML;
import utilidades.*;

/**
 * Componente de control implícito asociado a la página JSP "gapmatch.jsp". Es la lógica
 * asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea
 * llamada la página. Su función es comprobar que todos los parámetros iniciales de la toma de
 * datos de una pregunta tipo "Gap Match" sean correctos, en cuyo caso redirecciona la
 * respuesta a la siguiente página de toma de datos de tipo "Gap Match". La clase implementa
 * la interfaz <code>Control</code>, implementando su método <code>doLogic</code>, que es el
 * único que contiene, y que es el encargado de realizar la lógica asociada a la página
 * "gapmatch.jsp".
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Gapmatch implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página "gapmatch.jsp".
  * Será llamado por el filtro "ControlFilter" en el caso de que detecte que la página
  * requerida es "gapmatch.jsp". Ya que es llamado desde un filtro, será ejecutado siempre
  * que llegue una petición de un cliente y antes de que la petición llegue al Servlet
  * (página JSP compilada a Servlet) que sirve la respuesta al cliente. Comprueba uno por
  * uno todos los parámetros, y, en caso de que sean correctos, crea un objeto "GapmatchXML"
  * para guardar los datos asociados al tipo de pregunta y lo guarda en el ámbito de sesión
  * para utilizarlos más adelante para terminar de crear el ítem.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     utilidades.MensajeEstado
  * @see     utilidades.EnteroPositivo
  * @see     xml.items.GapmatchXML
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Comprueba que no sea la primera vez que se visita la página
    if (request.getParameter("Form") != null){

       String identificador = request.getParameter("Identif"); // Identificador del ítem
       String titulo = request.getParameter("Tit");       // Título del ítem
       String instrucciones = request.getParameter("Instruc"); // Instrucciones del ítem
       String pregunta = request.getParameter("Preg");       // Pregunta del ítem
       String Shuff = request.getParameter("Shuffle");       // Mezclar respuestas o no
       String numOpc = request.getParameter("Num_opc");      // Número de opciones
       String numHuec = request.getParameter("Num_huec");    // Número de huecos
```

```
// Caracteres reservados no permitidos
final String reservados = ";/?:@&=+$,\\*\"<>|";

// Estado de la petición web. Almacena si son correctos los parámetros, los
// mensajes de aviso al usuario y dónde colocar el foco en la página
EstadoWeb estado = new EstadoWeb();

boolean shuffle = false;   //valor por defecto
if ( Shuff != null){    // Interpreta el valor marcado en la página
   if (Shuff.equalsIgnoreCase("ON"))
      shuffle = true;
}

// Comprueba que estén introducidos todos los atributos obligatorios
if (identificador != null && !identificador.equals("") &&
      titulo != null && !titulo.equals("") &&
      numOpc != null && !numOpc.equals("") &&
      numHuec != null && !numHuec.equals("")){

   // Comprueba que los caracteres del identificador estén permitidos
     for(int i = 0; i < identificador.length() && estado.isEstado() == true; i++) {
      char c = identificador.charAt(i);   // Lee el siguiente carácter
      int marca = reservados.indexOf(c);
      if (marca >= 0 ){
         estado.añadeError("Error: El carácter '" + reservados.charAt(marca) +
               "' en la posición " + (i+1) + " del Identificador no está " +
               "permitido<br />", "Identif");
      }
     }

   // Comprueba que el número de opciones sea un valor válido
   EnteroPositivo opciones = new EnteroPositivo(numOpc);
   MensajeEstado msg = opciones.esPositivo();
   if (msg.isEstado()){    // Si es un entero válido y mayor que cero
      if (opciones.getEnteroPositivo() <= 1)    // Si es menor o igual que uno
         estado.añadeError("El número de opciones debe ser mayor que uno" +
               "<br />", "Num_opc");
   } else                  // Si no es un entero positivo válido
      estado.añadeError("Error: el número de opciones " + msg.getMensaje() +
            "<br />", "Num_opc");

   // Comprueba que el número de huecos sea un valor válido
   EnteroPositivo huecos = new EnteroPositivo(numHuec);
   msg = huecos.esPositivo();
   if (msg.isEstado()){    // Si es un entero válido y mayor que cero
      if (huecos.getEnteroPositivo() < 1)       // Si es menor que uno
         estado.añadeError("El número de huecos debe ser mayor o igual que 1" +
               "<br />", "Num huec");
   } else                  // Si no es un entero positivo válido
      estado.añadeError("Error: el número de huecos es un " + msg.getMensaje() +
            "<br />", "Num_huec");

   // Comprueba que las instrucciones sean un código XHTML válido
   if (instrucciones != null && !instrucciones.equals("")){
      msg = ParserNeutro.compruebaInstruccionesXHTML(instrucciones);
      if (msg.isEstado() == false)
         estado.añadeError(msg.getMensaje(), "Instruc");
   }

   // Si el estado es todo correcto pasa a generar el objeto contenedor del ítem
   if (estado.isEstado() == true){
      // Crea un objeto que contiene las características del ítem Gap Match a
      // crear
      GapmatchXML gmXML = new GapmatchXML(identificador, titulo, instrucciones,
            pregunta, shuffle);
      // Guarda el objeto Gapmatch en sesión para seguir utilizándolo luego
      HttpSession sesion = request.getSession();
      sesion.setAttribute("GapMatch", gmXML);
      // Redireccionamos a la página de introducir las respuestas indicando el
      // número de opciones, el de huecos y si hay que mezclar
      response.sendRedirect("respuestasgapmatch.jsp?NumOpc=" + numOpc +
            "&NumHuec=" + numHuec + "&Shuffle=" + shuffle);
   }
}
// si no están todos los atributos obligatorios comprueba los que faltan
```

```
        // para informar al usuario
        else{
           if (identificador.equals(""))
              estado.añadeError("Debe introducir un Identificador<br />", "Identif");
           if (titulo.equals(""))
              estado.añadeError("Debe introducir un Título<br />", "Tit");
           if (numOpc.equals(""))
              estado.añadeError("Debe introducir un número de opciones<br />",
                    "Num_opc");
           if (numHuec.equals(""))
              estado.añadeError("Debe introducir un número de huecos<br />", "Num_huec");
        }

        // Si ha habido algún fallo
        if (estado.isEstado()== false){
           request.setAttribute("Aviso", estado.getMensaje());
           request.setAttribute("Focus", estado.getFocusON());
           request.setAttribute("Identif", identificador);
           request.setAttribute("Tit", titulo);
           request.setAttribute("Instruc", instrucciones);
           request.setAttribute("Preg", pregunta);
           request.setAttribute("Num_opc", numOpc);
           request.setAttribute("Num_huec", numHuec);
           request.setAttribute("Shuffle", new Boolean(shuffle));
        }
     } else {
        // Si es la primera vez que se visita la página borra el objeto GapMatchXML del
        // ámbito de sesión para iniciar desde cero el ítem y pone el foco en el primer
        // campo del formulario
        HttpSession sesion = request.getSession();
        sesion.removeAttribute("GapMatch");
        request.setAttribute("Focus", "Identif");
     }

     return true;
  }
}
```

Hottext.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import utilidades.*;

import xml.items.HottextXML;

/**
 * Componente de control implícito asociado a la página JSP "hottext.jsp". Es la lógica
 * asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea
 * llamada la página. Su función es comprobar que todos los parámetros iniciales de la toma de
 * datos de una pregunta tipo "Hot Text" sean correctos, en cuyo caso redirecciona la
 * respuesta a la siguiente página de toma de datos de tipo "Hot Text". La clase implementa la
 * interfaz <code>Control</code>, implementando su método <code>doLogic</code>, que es el
 * único que contiene, y que es el encargado de realizar la lógica asociada a la página
 * "hottext.jsp".
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Hottext implements Control {

  /**
   * Realiza la comprobación de los parámetros que se recogen en la página "hottext.jsp".
   * Será llamado por el filtro <code>ControlFilter</code> en el caso de que detecte que la
   * página requerida es "hottext.jsp". Ya que es llamado desde un filtro, será ejecutado
   * siempre que llegue una petición de un cliente y antes de que la petición llegue al
```

```
 * Servlet (página JSP compilada a Servlet) que sirve la respuesta al cliente. Comprueba
 * uno por uno todos los parámetros, y, en caso de que sean correctos, crea un objeto
 * "HottextXML" para guardar los datos asociados al tipo de pregunta y lo guarda en el
 * ámbito de sesión para utilizarlos más adelante para terminar de crear el ítem.
 *
 * @param   request           la petición http enviada por el cliente
 * @param   response       la respuesta http a enviar desde el servidor
 * @return                 true para que se sigan ejecutando los demás filtros del
 *                         servidor y se envíe la respuesta correctamente
 * @throws  IOException       no debe de producirse
 * @throws  ServletException  si hay algún problema accediendo a request o response
 * @see
control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
 * @see     javax.servlet.http.HttpServletRequest
 * @see     javax.servlet.http.HttpServletResponse
 * @see  control.ControlFilter
 * @see     utilidades.EstadoWeb
 * @see     utilidades.MensajeEstado
 * @see     utilidades.EnteroPositivo
 * @see     xml.items.HottextXML
 */
public boolean doLogic(HttpServletRequest request,
      HttpServletResponse response) throws IOException, ServletException {

   // Comprueba que no sea la primera vez que se visita la página
   if (request.getParameter("Form") != null){

      // Caracteres reservados no permitidos
      final String reservados = ";/?:@&=+$,\\*\"<>|";

      String identificador = request.getParameter("Identif"); // Identicador del ítem
      String titulo = request.getParameter("Tit");       // Título del ítem
      String instrucciones = request.getParameter("Instruc"); // Instrucciones del ítem
      String pregunta = request.getParameter("Preg");       // Pregunta del ítem
      String numHot = request.getParameter("Num_hot");      // Número de hottexts

      // Estado de la petición web. Almacena si son correctos los parámetros, los
      // mensajes de aviso al usuario y dónde colocar el foco en la página
      EstadoWeb estado = new EstadoWeb();

      // Comprueba que estén introducidos todos los atributos obligatorios
      if (identificador != null && !identificador.equals("") &&
            titulo != null && !titulo.equals("") &&
            numHot != null && !numHot.equals("")){

         // Comprueba que los caracteres del identificador estén permitidos
           for(int i = 0; i < identificador.length() && estado.isEstado() == true; i++) {
            char c = identificador.charAt(i);   // Lee el siguiente carácter
            int marca = reservados.indexOf(c);
            if (marca >= 0 ){
               estado.añadeError("Error: El carácter '" + reservados.charAt(marca) +
                     "' en la posición " + (i+1) + " del Identificador no está " +
                     "permitido<br />", "Identif");
            }
           }

         // Comprueba que el número de hottext sea un valor válido
         EnteroPositivo numhottexts= new EnteroPositivo(numHot);
         MensajeEstado msg = numhottexts.esPositivo();
         if (msg.isEstado()){    // Si es un número válido y mayor o igual que cero
            if (numhottexts.getEnteroPositivo() <= 1)    // Si es menor o igual que uno
               estado.añadeError("El número de hottexts debe ser mayor que uno" +
                     "<br />", "Num_hot");
         } else                  // Si no es un entero positivo válido
            estado.añadeError("Error: el número de hottext " + msg.getMensaje() +
                  "<br />", "Num_hot");

         // Comprueba que las instrucciones sean un código XHTML válido
         if (instrucciones != null && !instrucciones.equals("")){
            msg = ParserNeutro.compruebaInstruccionesXHTML(instrucciones);
            if (msg.isEstado() == false)
               estado.añadeError(msg.getMensaje(), "Instruc");
         }

         // Si todo es correcto
```

```
            if (estado.isEstado() == true){
               // Crea un objeto HottextXML con las características del ítem Hot Text a
               // crear
               HottextXML htXML = new HottextXML(identificador, titulo, instrucciones,
                     pregunta);
               // Guarda el objeto HottextXML en sesión para seguir utilizándolo luego
               HttpSession sesion = request.getSession();
               sesion.setAttribute("HotText", htXML);
               // Redirecciona a la página de introducir los hottexts y los textos
               // indicando el número de hottexts
               response.sendRedirect("respuestashottext.jsp?NumHot=" + numHot);
            }
         }
         // Si no están todos los atributos obligatorios comprueba los que fallan
         // para informar al usuario
         else{
            if (identificador.equals(""))
               estado.añadeError("Debe introducir un Identificador<br />", "Identif");
            if (titulo.equals(""))
               estado.añadeError("Debe introducir un Título<br />", "Tit");
            if (numHot.equals(""))
               estado.añadeError("Debe introducir un número de hottexts<br />",
                     "Num_hot");
         }

         // Si ha habido algún fallo pone los datos que ya ha introducido el usuario junto
         // a los mensajes de aviso y el foco en el primer fallo encontrado
         if (estado.isEstado()== false){
            request.setAttribute("Aviso", estado.getMensaje());
            request.setAttribute("Focus", estado.getFocusON());
            request.setAttribute("Identif", identificador);
            request.setAttribute("Tit", titulo);
            request.setAttribute("Instruc", instrucciones);
            request.setAttribute("Preg", pregunta);
            request.setAttribute("Num_hot", numHot);
         }
      } else {
         // Si es la primera vez que se visita la página borra el objeto hottextXML del
         // ámbito de sesión para iniciar desde cero el ítem y pone el foco en el primer
         // campo del formulario
         HttpSession sesion = request.getSession();
         sesion.removeAttribute("HotText");
         request.setAttribute("Focus", "Identif");
      }

      return true;
   }
}
```

Index.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 * Componente de control implícito asociado a la página JSP "index.jsp". Es la lógica
 * asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea
 * llamada la página. Sólo se preocupa de inicializar la sesión del usuario con una nueva para
 * borrar cualquier objeto antiguo almacenado en sesión. La clase implementa la interfaz <code>
 * Control</code>, implementando su método <code>doLogic</code>, que es el único que contiene,
 * y que es el encargado de realizar la lógica asociada a la página "index.jsp".
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Index implements Control {
```

```
 /**
  * Reinicia el objeto sesión para eliminar cualquier objeto a desechar almacenado ahí. Será
  * llamado por el filtro "ControlFilter" en el caso de que detecte que la página requerida
  * es "index.jsp". Ya que es llamado desde un filtro, será ejecutado siempre que llegue una
  * petición de un cliente y antes de que la petición llegue al Servlet (página JSP compilada
  * a Servlet) que sirve la respuesta al cliente.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    HttpSession sesion = request.getSession();
    sesion.invalidate();
    sesion = request.getSession(true);

    return true;
 }

}
```

Inlinechoice.java

```
package control;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import xml.items.InlinechoiceXML;

import utilidades.*;

/**
 * Componente de control implícito asociado a la página JSP "inlinechoice.jsp". Es la lógica
 * asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea
 * llamada la página. Su función es comprobar que todos los parámetros iniciales de la toma de
 * datos de una pregunta tipo "Inline Choice" sean correctos, en cuyo caso redirecciona la
 * respuesta a la siguiente página de toma de datos de tipo "Inline Choice". La clase
 * implementa la interfaz <code>Control</code>, implementando su método <code>doLogic</code>,
 * que es el único que contiene, y que es el encargado de realizar la lógica asociada a la
 * página "inlinechoice.jsp".
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Inlinechoice implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página
  * "inlinechoice.jsp". Será llamado por el filtro <code>ControlFilter</code> en el caso de
  * que detecte que la página requerida es "inlinechoice.jsp". Ya que es llamado desde un
  * filtro, será ejecutado siempre que llegue una petición de un cliente y antes de que la
  * petición llegue al Servlet (página JSP compilada a Servlet) que sirve la respuesta al
  * cliente. Comprueba uno por uno todos los parámetros, y, en caso de que sean correctos,
  * crea un objeto <code>InlinechoiceXML</code> para guardar los datos asociados al tipo de
  * pregunta y lo guarda en el ámbito de sesión para utilizarlos más adelante para terminar
  * de crear el ítem.
  *
  * @param   request           la petición http enviada por el cliente
```

```
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.MensajeEstado
  * @see     utilidades.EnteroPositivo
  * @see     utilidades.EstadoWeb
  * @see     xml.items.InlinechoiceXML
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Comprueba que no sea la primera vez que se visita la página
    if (request.getParameter("Form") != null){

       // Caracteres reservados no permitidos
       final String reservados = ";/?:@&=+$,\\*\"<>|";

       String identificador = request.getParameter("Identif"); // Identificador del ítem
       String titulo = request.getParameter("Tit");       // Título del ítem
       String instrucciones = request.getParameter("Instruc"); // Instrucciones del ítem
       String Shuff = request.getParameter("Shuffle");       // Mezclar las opciones
       String numOpc = request.getParameter("Num_opc");      // Número de opciones

       // Estado de la petición web. Almacena si son correctos los parámetros, los
       // mensajes de aviso al usuario y dónde colocar el foco en la página
       EstadoWeb estado = new EstadoWeb();

       boolean shuffle = false;   //valor por defecto
       if ( Shuff != null){    // Interpreta el valor marcado en la página
          if (Shuff.equalsIgnoreCase("ON"))
             shuffle = true;
       }

       // Comprueba que estén introducidos todos los atributos obligatorios
       if (identificador != null && !identificador.equals("") &&
             titulo != null && !titulo.equals("") &&
             numOpc != null && !numOpc.equals("")){

          // Comprueba que los caracteres del identificador estén permitidos
            for(int i = 0; i < identificador.length() && estado.isEstado() == true; i++) {
             char c = identificador.charAt(i);   // Lee el siguiente carácter
             int marca = reservados.indexOf(c);
             if (marca >= 0 ){
                estado.añadeError("Error: El carácter '" + reservados.charAt(marca) +
                      "' en la posición " + (i+1) + " del Identificador no está " +
                      "permitido<br />", "Identif");
             }
            }

          // Comprueba que el número de opciones sea un valor válido
          EnteroPositivo opciones = new EnteroPositivo(numOpc);
          MensajeEstado msg = opciones.esPositivo();
          if(msg.isEstado()){     // Si es un número válido y mayor o igual que cero
             if ( opciones.getEnteroPositivo() <= 1)      // Si es menor o igual que uno
                estado.añadeError("El número de opciones debe ser mayor que 1<br />",
                      "Num_opc");
          }else                   // Si no es un entero positivo válido
             estado.añadeError("Error: el número de opciones " + msg.getMensaje() +
                   "<br />", "Num_opc");

          // Comprueba que las instrucciones sean un código XHTML válido
          if (instrucciones != null && !instrucciones.equals("")){
             msg = ParserNeutro.compruebaInstruccionesXHTML(instrucciones);
             if (msg.isEstado() == false)
                estado.añadeError(msg.getMensaje(), "Instruc");
          }

          // Si el estado es todo correcto pasa a generar el objeto contenedor del ítem
```

```
            if (estado.isEstado() == true){
               // Crea un objeto que contiene las características del ítem Inline Choice
               // a crear
               InlinechoiceXML icXML = new InlinechoiceXML(identificador, titulo,
                     instrucciones, shuffle);
               // Guarda el objeto Inlinechoice en sesión para seguir utilizándolo luego
               HttpSession sesion = request.getSession();
               sesion.setAttribute("InlineChoice", icXML);
               // Redirecciona a la página de introducir las opciones indicando el
               // número de opciones y si hay que mezclar
               response.sendRedirect("respuestasinlinechoice.jsp?Num_Opc=" + numOpc +
                     "&Shuff=" + shuffle);
            }
         }
         // si no están todos los atributos obligatorios comprueba los que fallan
         // para informar al usuario
         else{
            if (identificador.equals(""))
               estado.añadeError("Debe introducir un Identificador<br />", "Identif");
            if (titulo.equals(""))
               estado.añadeError("Debe introducir un Título<br />", "Tit");
            if (numOpc.equals(""))
               estado.añadeError("Debe introducir un número de respuestas<br />",
                     "Num_opc");
         }

         // Si ha habido algún fallo pone los datos que ya ha introducido el usuario junto
         // a los mensajes de aviso y el foco en el primer fallo encontrado
         if (estado.isEstado() == false){
            request.setAttribute("Aviso",estado.getMensaje());
            request.setAttribute("Focus", estado.getFocusON());
            request.setAttribute("Identif",identificador);
            request.setAttribute("Tit",titulo);
            request.setAttribute("Instruc",instrucciones);
            request.setAttribute("Num_opc",numOpc);
            request.setAttribute("Shuffle",new Boolean(shuffle));
         }
      } else {
         // Si es la primera vez que se visita la página borra inlinechoiceXML del ámbito
         // de sesión para iniciar desde cero el ítem y pone el foco en el primer campo del
         // formulario
         HttpSession sesion = request.getSession();
         sesion.removeAttribute("InlineChoice");
         request.setAttribute("Focus", "Identif");
      }

      return true;
   }
}
```

Introducirpesos.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import utilidades.EstadoWeb;
import utilidades.ResumenItem;
import utilidades.Identificador;
import utilidades.Comprobaciones;

import xml.test.*;

/**
 * Componente de control implícito asociado a la página JSP "introducirpesos.jsp". Es la lógica
 * asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea
 * llamada la página. Su función es comprobar que los datos introducidos en la página de
 * introducción de los pesos de los ítems del test sean correctos. La clase implementa la
 * interfaz <code>Control</code>, implementando su método <code>doLogic</code>, que es el
```

```
 * encargado de realizar la lógica asociada a la página "introducirpesos.jsp". Contiene también
 * un método para leer la tabla de pesos de la página y otro método para rellenar los pesos que
 * se dejen vacíos.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Introducirpesos implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página
  * "introducirpesos.jsp". Será llamado por el filtro <code>ControlFilter</code> en el caso
  * de que detecte que la página requerida es "introducirpesos.jsp". Ya que es llamado desde
  * un filtro, será ejecutado siempre que llegue una petición de un cliente y antes de que
  * la petición llegue al Servlet (página JSP compilada a Servlet) que sirve la respuesta al
  * cliente. Comprueba uno por uno todos los parámetros, y, en caso de que sean correctos,
  * crea el array de referencias a los ítems y guarda el objeto test actualizado en el
  * ámbito de sesión para utilizarlos más adelante para terminar de crear el test.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     xml.test.AssessmentItemRef
  * @see     xml.test.AssessmentTest
  * @see     xml.test.AssessmentSection
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Nombre base para los identificadores de los ítems por defecto
    final String identificadorBase = "item_";
    request.setAttribute("IdentificadorBase", identificadorBase);

    // Objeto sesión para acceder a los objetos ahí almacenados
    HttpSession sesion = request.getSession();
    // Si el objeto sesión no es nuevo la sesión no ha expirado
    if (!sesion.isNew()){
       // Array con los ítems seleccionados para añadir al test
       ResumenItem[] items = (ResumenItem[])sesion.getAttribute("Items A Añadir");
       // Variable para almacenar los errores de la web y dónde colocar el foco
       EstadoWeb estado = new EstadoWeb();
       // Array con los identificadores
       Identificador[] identificadores = new Identificador[items.length];
       // Array de valores de los pesos en caso de ítem correcto
       Double[] pesoOk = new Double[items.length];
       // Array de valores de los pesos en caso de ítem incorrecto
       Double[] pesoNoOk = new Double[items.length];
       // Se leen los datos de si se mezclan aleatoriamente las respuestas y de si se
       // seleccionan sólo algunos de los ítems
       AssessmentTest test = (AssessmentTest)sesion.getAttribute("Test1");
       boolean shuffle = false;
       boolean select = false;
       // Si el objeto Ordering es distinto de null se mezclan aleatoriamente
       if (test.getTestPart().getSeccion().getOrden() != null)
          shuffle = true;
       // Si el objeto Selection es distinto de null hay una selección entre los ítems
       if (test.getTestPart().getSeccion().getSeleccion() != null)
          select = true;
       // Array de boolean indicando si se dejan en posición fija los ítems
       boolean[] fija = new boolean[items.length];
       // Array de boolean indicando si los ítems son requeridos
       boolean[] requerido = new boolean[items.length];
       // Si se sigue en esta página o se pasa a la siguiente
       boolean salida = false;
```

```
// Si se ha pulsado el botón de crear test
if(request.getParameter("Crear") != null){
   // Lee los identificadores de los ítems comprobando que sean correctos y no
   // haya ninguno repetido, actualizando el estado de la web tras la comprobación
   Identificador.testIdentificadores(identificadores, "Identif_", estado, request,
         "");

   // Lee los pesos en caso de respuesta correcta comprobándolos
   leePesos(request, "Ok_", pesoOk, estado, "correcto");
   // Los pesos en caso de respuesta correcta deben ser mayores que cero
   for (int i = 0; i < pesoOk.length; i++) {
      if (pesoOk[i] != null && pesoOk[i].doubleValue() <= 0)
         estado.añadeError("El peso correcto del ítem " + i + " debe ser un " +
               "número mayor que cero<br />", "Ok_" + i);
   }

   // lee los pesos en caso de respuesta incorrecta comprobándolos
   leePesos(request, "NoOk_", pesoNoOk, estado, "incorrecto");
   // Los pesos en caso de respuesta incorrecta deben ser iguales o menores que
   // cero
   for (int i = 0; i < pesoNoOk.length; i++) {
      if (pesoNoOk[i] != null && pesoNoOk[i].doubleValue() > 0)
         pesoNoOk[i] = new Double(-pesoNoOk[i].doubleValue());
   }

   // Lee los ítems en posición fija si se mezclan aleatoriamente
   if (shuffle == true){
      Comprobaciones.leeCheckBoxes(fija, "Fija_", request);
   }

   // Lee los ítems requeridos si hay una selección
   if (select == true){
      Comprobaciones.leeCheckBoxes(requerido, "Requerido_", request);
      // Comprueba que no se hayan marcado como requeridos más ítems que los que
      // se deben seleccionar del test
      int numRequeridos = 0;
      for (int i = 0; i < requerido.length; i++)
         if (requerido[i] == true)
            numRequeridos++;
      int seleccionar =
         test.getTestPart().getSeccion().getSeleccion().getSeleccion();
      if (seleccionar < numRequeridos)
         estado.añadeError("Debe seleccionar menos ítems requeridos que el " +
               "número de ítems a seleccionar para el test, " + seleccionar,
               "Requerido_0");
   }

   // Si todos los pesos introducidos son números decimales válidos y están todos
   // los identificadores y son válidos
   if (estado.isEstado() == true){
      // Se rellenan los pesos correctos vacíos con 1 y incorrectos vacíos con 0
      rellenaPesos(pesoOk, 1);
      rellenaPesos(pesoNoOk, 0);
      // Se crea el array de referencias a los ítems para el test
      AssessmentItemRef[] itemArray = new AssessmentItemRef[items.length];
      for (int i = 0; i < itemArray.length; i++){
         Weight wOK = new Weight(new Identificador("WeightOK"),
               pesoOk[i].doubleValue());
         Weight wNOOK = new Weight(new Identificador("WeightNOK"),
               pesoNoOk[i].doubleValue());
         itemArray[i] = new AssessmentItemRef(identificadores[i], requerido[i],
               fija[i], items[i].getArchivo(), wOK, wNOOK);
      }
      // Almacena el array de referencias a los ítems en la sección del test
      test.getTestPart().getSeccion().setItemRefArray(itemArray);
      // Almacena en sesión el test para usarlo más adelante
      sesion.setAttribute("Test2", test);
      // Se redirecciona la respuesta a la siguiente página para crear el test
      response.sendRedirect("seleccionasignatura.jsp?Destino=fintest.jsp");
      salida = true;
   }
}
// Si no se ha pulsado el botón de crear test es la primera vez que se entra en la
// página y se inicializan los identificadores de los ítems a unos valores por
// defecto y se coloca el foco en el primer campo del formulario de peso correcto
else {
```

```
            for (int i = 0; i < items.length; i++)
               if (identificadores[i] == null)
                  identificadores[i] = new Identificador(identificadorBase + (i+1));

            request.setAttribute("Focus", "Ok_1");
         }

         // Si se sigue en esta página se guardan en request todos los parámetros requeridos
         // y los ya introducidos si se ha introducido alguno
         if (salida == false){
            request.setAttribute("Shuffle", new Boolean(shuffle));
            request.setAttribute("Select", new Boolean(select));
            request.setAttribute("Fija", fija);
            request.setAttribute("Requerido", requerido);
            request.setAttribute("Aviso", estado.getMensaje());
            request.setAttribute("Identif", identificadores);
            request.setAttribute("Ok", pesoOk);
            request.setAttribute("NoOk", pesoNoOk);
            request.setAttribute("Focus", estado.getFocusON());
            sesion.setAttribute("Items_A_Añadir", items);
         }

      } else      // Si la sesión ha expirado
         response.sendRedirect("errorfinsesion.jsp");

      return true;
   }

   /**
    * Lee los pesos introducidos en la página de introducción de pesos del objeto
    * <code>HttpServletRequest</code>, con el nombre común de los campos dado por el <code>
    * String</code> <code>origen</code> y rellena el array de <code>Double</code> pesos,
    * comprobando que sean números decimales correctos, y actualizando el valor del objeto
    * <code>estado</code> en caso de error.
    *
    * @param   request     httpServletRequest de donde leer los pesos introducidos por el
    *                   usuario
    * @param   origen      string con el nombre base común de los campos que contienen los
    *                   pesos a leer
    * @param   pesos    array de Double en el que almacenar los pesos leídos si son números
    *                   decimales correctos
    * @param   estado      estado de la comprobación para avisar en caso de error
    * @param   tipoPeso string indicando si el peso es tipo correcto o incorrecto para el
    *                   aviso al usuario
    */
   public void leePesos (HttpServletRequest request, String origen, Double[] pesos,
         EstadoWeb estado, String tipoPeso){

      for (int i = 0; i < pesos.length; i++){
         try {
            // Valor del peso en caso de respuesta correcta
            String peso = request.getParameter(origen + i);
            // Si el peso se ha rellenado con un valor
            if (peso != null && !peso.equals(""))
               pesos[i] = new Double(peso);
         }catch (NumberFormatException e){
            estado.añadeError("El peso " + tipoPeso + " del ítem " + i + " es incorrecto" +
                  "<br />", origen + i);
         }
      }
   }

   /**
    * Rellena las posiciones vacías del array de <code>Double pesos</code> con el
    * valor indicado por el parámetro <code>relleno</code>.
    *
    * @param   pesos    array de objetos Double a terminar de rellenar
    * @param   relleno     double con el valor por defecto para rellenar los huecos vacíos
    */
   public void rellenaPesos (Double[] pesos, double relleno){
      for (int i = 0; i < pesos.length; i++)
         if (pesos[i] == null)
            pesos[i] = new Double(relleno);
   }

}
```

Match.java

```
package control;

import utilidades.*;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import xml.items.MatchXML;

/**
 * Componente de control implícito asociado a la página JSP "match.jsp". Es la lógica asociada
 * a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea llamada la
 * página. Su función es comprobar que todos los parámetros iniciales de la toma de datos de
 * una pregunta tipo "Match" sean correctos, en cuyo caso redirecciona la respuesta a la
 * siguiente página de toma de datos de tipo "Match". La clase implementa la interfaz <code>
 * Control</code>, implementando su método <code>doLogic</code>, que es el único que contiene,
 * y que es el encargado de realizar la lógica asociada a la página "match.jsp".
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Match implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página "match.jsp". Será
  * llamado por el filtro <code>ControlFilter</code> en el caso de que detecte que la
  * página requerida es "match.jsp". Ya que es llamado desde un filtro, será ejecutado
  * siempre que llegue una petición de un cliente y antes de que la petición llegue al
  * Servlet (página JSP compilada a Servlet) que sirve la respuesta al cliente. Comprueba
  * uno por uno todos los parámetros, y, en caso de que sean correctos, crea un objeto <code>
  * MatchXML</code> para guardar los datos asociados al tipo de pregunta y lo guarda en el
  * ámbito de sesión para utilizarlos más adelante para terminar de crear el ítem.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     utilidades.MensajeEstado
  * @see     utilidades.EnteroPositivo
  * @see     xml.items.MatchXML
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Comprueba que no sea la primera vez que se entra en la página
    if (request.getParameter("Form") != null){

       // Caracteres reservados no permitidos
       final String reservados = ";/?:@&=+$,\\*\"<>|";

       String identificador = request.getParameter("Identif"); // Identificador del ítem
       String titulo = request.getParameter("Tit");       // Título del ítem
       String instrucciones = request.getParameter("Instruc"); // Instrucciones del ítem
       String pregunta = request.getParameter("Preg");       // Pregunta del ítem
       // Número de elementos del grupo 1
       String grupo1 = request.getParameter("NumGrupo1");
       // Número de elementos del grupo 2
       String grupo2 = request.getParameter("NumGrupo2");
       // Mezclar las preguntas o no
       String Shuff = request.getParameter("Shuffle");
```

```
// Estado de la petición web. Almacena si son correctos los parámetros, los
// mensajes de aviso al usuario y dónde colocar el foco en la página
EstadoWeb estado = new EstadoWeb();

boolean shuffle = false;         //valor por defecto
if ( Shuff != null){          // Interpreta el valor marcado en la página
   if (Shuff.equalsIgnoreCase("ON"))
         shuffle = true;
}

// Comprueba que estén introducidos todos los atributos obligatorios
if (identificador != null && !identificador.equals("") &&
   titulo != null && !titulo.equals("") &&
   grupo1 != null && !grupo1.equals("") &&
   grupo2 != null && !grupo2.equals("")){

   // Comprueba que los caracteres del identificador estén permitidos
     for(int i = 0; i < identificador.length() && estado.isEstado() == true; i++) {
      char c = identificador.charAt(i);   // Lee el siguiente carácter
      int marca = reservados.indexOf(c);
      if (marca >= 0 ){
         estado.añadeError("Error: El carácter '" + reservados.charAt(marca) +
               "' en la posición " + (i+1) + " del Identificador no está " +
               "permitido<br />", "Identif");
      }
     }

   // Recoge los valores del número de elementos en cada grupo
   EnteroPositivo numGrupo1 = new EnteroPositivo(grupo1);
   EnteroPositivo numGrupo2 = new EnteroPositivo(grupo2);
   // Comprueba el número de elementos del grupo 1
   MensajeEstado msg = numGrupo1.esPositivo();
   if(msg.isEstado()){     // Si es un número válido y mayor o igual que cero
      if(numGrupo1.getEnteroPositivo() < 1)  // Si es menor o igual que uno
         estado.añadeError("Error: El número de elementos del grupo 1 debe " +
               "ser un número mayor que cero<br />", "NumGrupo1");
   } else               // Si no es un entero positivo válido
      estado.añadeError("Error: el número de elementos del grupo 1 " +
            msg.getMensaje() + "<br />", "NumGrupo1");
   // Comprueba el número de elementos del grupo 2
   msg = numGrupo2.esPositivo();
   if(msg.isEstado()){     // Si es un número válido y mayor o igual que cero
      if(numGrupo2.getEnteroPositivo() < 1)  // Si es menor o igual que uno
         estado.añadeError("Error: El número de elementos del grupo 2 debe " +
               "ser un número mayor que cero<br />", "NumGrupo2");
   } else               // Si no es un entero positivo válido
      estado.añadeError("Error: el número de elementos del grupo 2 " +
            msg.getMensaje() + "<br />", "NumGrupo2");

   // Comprueba que las instrucciones sean un código XHTML válido
   if (instrucciones != null && !instrucciones.equals("")){
      msg = ParserNeutro.compruebaInstruccionesXHTML(instrucciones);
      if (msg.isEstado() == false)
         estado.añadeError(msg.getMensaje(), "Instruc");
   }

   // Si el estado es todo correcto pasa a generar el objeto contenedor del ítem
   if (estado.isEstado()== true){
      // Crea un objeto que contiene las características del ítem Match a crear
      MatchXML maXML = new MatchXML(identificador, titulo, instrucciones,
            pregunta, shuffle);
      // Guarda el objeto Match en sesión para seguir utilizándolo en la sesión
      HttpSession sesion = request.getSession();
      sesion.setAttribute("Match", maXML);
      // Redirecciona a la página de introducir las respuestas indicando los
      // números de elementos de cada grupo y si hay que mezclar
      response.sendRedirect("respuestasmatch.jsp?NumGrupo1=" + grupo1 +
            "&NumGrupo2=" +   grupo2 + "&Shuffle=" + shuffle);
   }
}
// si no están todos los atributos obligatorios comprueba los que fallan
// para informar al usuario.
else{
   if (identificador.equals(""))
      estado.añadeError("Debe introducir un Identificador<br />", "Identif");
   if (titulo.equals(""))
```

```
            estado.añadeError("Debe introducir un Título<br />", "Tit");
         if (grupo1.equals(""))
            estado.añadeError("Debe introducir el número de elementos del grupo 1" +
                  "<br />", "NumGrupo1");
         if (grupo2.equals(""))
            estado.añadeError("Debe introducir el número de elementos del grupo 2" +
                  "<br />", "NumGrupo2");
      }
      // Si ha habido algún fallo pone los datos que ya ha introducido el usuario junto
      // a los mensajes de aviso y el foco en el primer fallo encontrado
      if (estado.isEstado() == false){
         request.setAttribute("Identif", identificador);
         request.setAttribute("Tit", titulo);
         request.setAttribute("Instruc", instrucciones);
         request.setAttribute("Preg", pregunta);
         request.setAttribute("NumGrupo1",grupo1);
         request.setAttribute("NumGrupo2",grupo2);
         request.setAttribute("Shuffle",new Boolean(shuffle));
         request.setAttribute("Aviso",estado.getMensaje());
         request.setAttribute("Focus",estado.getFocusON());
      }
   } else {
      // Si es la primera vez que se visita la página borra el objeto matchXML del
      // ámbito de sesión para iniciar desde cero el ítem y pone el foco en el primer
      // campo del formulario
      HttpSession sesion = request.getSession();
      sesion.removeAttribute("Match");
      request.setAttribute("Focus", "Identif");
   }

   return true;
 }
}
```

Nuevodirectorio.java

```
package control;

import java.io.File;
import java.io.IOException;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Componente de control implícito asociado a la página JSP "nuevodirectorio.jsp". Es la
 * lógica asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que
 * sea llamada la página. Se encarga de crear un nuevo directorio.
 *
 * @author   David Domínguez
 * @see      Control
 *
 */
public class Nuevodirectorio implements Control {

 /**
  * Se encarga de crear un nuevo subdirectorio con el nombre indicado por <code>DirNombre
  * </code> en el directorio <code>DirectorioActual</code>, parámetros ambos que recibe en
  * el request. Será llamado por el filtro <code>ControlFilter</code> en el caso de que
  * detecte que la página requerida es "nuevodirectorio.jsp". Ya que es llamado desde un
  * filtro, será ejecutado siempre que llegue una petición de un cliente y antes de que la
  * petición llegue al Servlet (página JSP compilada a Servlet) que sirve la respuesta al
  * cliente. Comprueba que no exista ningún subdirectorio con ese nombre ya en el directorio
  * actual, e intenta crearlo. Avisa al usuario en caso de que ocurra algún fallo. Vuelve a
  * la página indicada en el parámetro <code>VolverA</code> recibido en el request.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       si hay algún problema accediendo al directorio o al crearlo
  * @throws  ServletException  si hay algún problema accediendo a request o response
```

```
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Caracteres reservados no permitidos
    final String reservados = ";/?:@&=+$,\\*\"<>|";

    ServletContext sc =  request.getSession().getServletContext();
    String dirActual = request.getParameter("DirectorioActual");// Directorio actual
    String dirNuevo = request.getParameter("DirNombre");     // Directorio a crear
    String origen = request.getParameter("VolverA");         // Página a donde volver
    String aviso = new String("");                          // Errores al crearlo
    String informacion = new String("");                    // Información de éxito
    boolean correcto = true;               // Si es correcto el nombre del directorio

    // Pone en request el directorio actual y a donde volver por si hay algún fallo que
    // siga estando disponible o al pulsar cancelar
    request.setAttribute("DirectorioActual", dirActual);
        String atributos;   // Atributos que se pasan en la URL para la página siguiente
        if (origen.indexOf('?') == -1) // Si el origen no tiene '?' se pone uno el primero
          atributos = "?DirectorioActual=" + dirActual;
        else                    // Si ya tiene el '?', sigue añadiendo datos
          atributos = "&DirectorioActual=" + dirActual;

    request.setAttribute("VolverA", origen + atributos);
    // Si no es la primera vez que visita la página y se ha introducido el nombre del
    // directorio
    if (dirNuevo != null && !dirNuevo.equals("")){
       // Comprueba que los caracteres del nombre del directorio estén permitidos
         for(int i = 0; i < dirNuevo.length() && correcto == true; i++) {
          char c = dirNuevo.charAt(i);  // Lee el siguiente carácter
          int marca = reservados.indexOf(c);
          if (marca >= 0 ){
             aviso = "Error: El carácter '" + reservados.charAt(marca) + "' en la " +
                   "posición " + (i+1) + " no está permitido<br />";
             correcto = false;
          }
         }

         if (correcto == true){
          String dirTrabajo = sc.getInitParameter("Directorio Trabajo");
          File dir = new File( sc.getRealPath(dirTrabajo)+ dirActual + "/" + dirNuevo);
          if (!dir.exists()) {    // Si no existe el directorio intenta crearlo
             if (dir.mkdir()){
                informacion = "Directorio creado con éxito<br />";
             } else{            // Si el directorio no pudo ser creado avisa al usuario
                aviso = "El directorio no pudo ser creado<br />";
             }
          } else {              // Si el directorio ya existe avisa al usuario
             aviso = "El directorio ya existe<br />";
          }
         }
         // Redirecciona la respuesta a la página desde la que se llama a esta, pasándole
         // el directorio actual, y los avisos o información de éxito relacionadas con la
         // creación del directorio
         if (origen.indexOf('?') == -1) // Si el origen no tiene '?' se pone uno el primero
          atributos = "?Aviso=" + aviso + "&Informacion=" + informacion +
                "&DirectorioActual=" + dirActual;
         else                       // Si ya tiene el '?', sigue añadiendo datos
          atributos = "&Aviso=" + aviso + "&Informacion=" + informacion +
                "&DirectorioActual=" + dirActual;

       response.sendRedirect(origen + atributos);
    }

    return true;
 }
}
```

Respuestaschoice.java

```
package control;

import utilidades.*;

import xml.items.ChoiceXML;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.*;

/**
 * Componente de control implícito asociado a la página JSP "respuestaschoice.jsp". Es la
 * lógica asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que
 * sea llamada la página. Su función es comprobar que todos los parámetros finales de la toma
 * de datos de una pregunta tipo "Choice" sean correctos, en cuyo caso redirecciona la
 * respuesta a la página "seleccionasignatura.jsp" para elegir donde guardar el ítem. La clase
 * implementa la interfaz <code>Control</code>, implementando su método <code>doLogic</code>,
 * que es el encargado de realizar la lógica asociada a la página "respuestaschoice.jsp".
 * Tiene otro método que simplemente se ejecuta si los parámetros son correctos para
 * redireccionar la respuesta y terminar de crear el objeto <code>ChoiceXML</code>.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Respuestaschoice implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página
  * "respuestaschoice.jsp". Será llamado por el filtro <code>ControlFilter</code> en el
  * caso de que detecte que la página requerida es "respuestaschoice.jsp". Ya que es
  * llamado desde un filtro, será ejecutado siempre que llegue una petición de un cliente y
  * antes de que la petición llegue al Servlet (página JSP compilada a Servlet) que sirve la
  * respuesta al cliente. Comprueba uno por uno todos los parámetros, y, en caso de que sean
  * correctos, llama al método <code>terminaChoice</code> para terminar de crear el objeto
  * del ítem y redireccionar la respuesta.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     utilidades.EnteroPositivo
  * @see     utilidades.Identificador
  * @see     utilidades.Comprobaciones
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Nombre base para los identificadores de las respuestas por defecto
    final String identificadorBase = "Respuesta";
    request.setAttribute("IdentificadorBase", identificadorBase);

    // Estos elementos siempre estarán presentes en request al venir de la página anterior
    String num_Resp = request.getParameter("Num_Resp");   // Número de respuestas del ítem
    int numResp = Integer.parseInt(num_Resp);       // Número de respuestas en un entero
    String shuff = request.getParameter("Shuff");      // Mezclar respuestas sí o no

    // Array con los identificadores
    Identificador[] identificadores = new Identificador[numResp];

    //Comprueba si es la primera vez que se entra en la página
    if(request.getParameter("Form") != null){

       String[] respuestas = new String [numResp];  // Array con las respuestas
       // Estado de la petición web. Almacena si son correctos los parámetros, los
```

```
      // mensajes de aviso al usuario y dónde colocar el foco en la página
      EstadoWeb estado = new EstadoWeb();

      int i;
      // Lee las respuestas introducidas, y si alguna está vacía añade el código de error
      for(i = 0; i < numResp; i++){
         respuestas[i] = request.getParameter("Respuesta"+i);
         if (respuestas[i] == null || respuestas[i].equals(""))
            estado.añadeError("Error: la respuesta " + (i+1) + " no puede quedar " +
                  "vacía<br />", "Respuesta" + i);
      }

      // Lee los identificadores de las respuestas comprobando que sean correctos y no
      // haya ninguno repetido, devolviendo el estado de la web tras la comprobación
      Identificador.testIdentificadores(identificadores, "Identificador", estado,
            request, "");

      //Lee los checkbox de respuesta en posición fija
      boolean[] respFija = new boolean[numResp];
      Comprobaciones.leeCheckBoxes(respFija, "Fija", request);

      // Comprueba los checkbox de respuesta correcta que al menos uno esté marcado
      // y suma el número total de respuestas correctas
      int numCorrectas = 0;                  // Número total de respuesas correctas
      boolean[] respCorrecta = new boolean[numResp];  // Respuesta correcta o no
      numCorrectas = Comprobaciones.sumaCorrectas(respCorrecta, "Correcto", request);
      if (numCorrectas == 0)  // Sin ningún checkbox de respuesta correcta está marcado
         estado.añadeError("Debe marcar al menos una respuesta correcta<br />",
               "Correcto0");

      // Lee el valor introducido del número máximo de elecciones
      String max = request.getParameter("MaxChoices");
      EnteroPositivo maxChoices = new EnteroPositivo(max);
      // Comprueba que el valor introducido sea un valor correcto, mayor que el número
      // de respuestas correctas o cero, y menor que el número de respuestas posibles
      Comprobaciones.testMaximo(maxChoices, estado, "MaxChoices", numCorrectas, numResp);

      // Si todo ha ido bien llama al método que finaliza el ítem y redirecciona al
      // cliente
      if (estado.isEstado() == true){
         terminaChoice(request, response, respuestas, identificadores, maxChoices,
               respCorrecta, respFija);
      } else {
         // Si ha habido algún fallo pone los datos que ya ha introducido el usuario
         // junto a los mensajes de aviso y el foco en el primer fallo encontrado
         request.setAttribute("Respuestas",respuestas);
         request.setAttribute("Identificador",identificadores);
         request.setAttribute("MaxChoices",max);
         request.setAttribute("Correcto",respCorrecta);
         request.setAttribute("Fija",respFija);
         request.setAttribute("Aviso",estado.getMensaje());
         request.setAttribute("Focus",estado.getFocusON());
      }
   } else {
      // Si es la primera vez que se visita la página pone el foco en el primer campo
      request.setAttribute("Focus", "MaxChoices");
      // Se inicializan los identificadores de las respuestas a unos valores por defecto
      for (int i = 0; i < identificadores.length; i++){
         identificadores[i] = new Identificador(identificadorBase + (i+1));
         request.setAttribute("Identificador",identificadores);
      }
   }

   // El número de respuestas y si se mezclan las respuestas aleatoriamente hacen falta
   // siempre, sea la primera vez que visita la página o no y pone el foco en el primer
   // campo del formulario
   request.setAttribute("Num Resp", num_Resp);
   request.setAttribute("Shuff",shuff);

   return true;
}

/**
 * Este método se ejecuta una vez que se comprueba que todos los parámetros son correctos.
 * Usa el objeto <code>ChoiceXML</code> creado anteriormente y almacenado en sesión para
 * guardar el resto de los datos asociados al ítem, redireccionando la respuesta a
```

```
 * la página "seleccionasignatura.jsp" para que el usuario seleccione una asignatura en la
 * que guardar el ítem XML. Si el objeto <code>ChoiceXML</code> no se ha podido recuperar
 * de sesión porque ésta ha expirado, redirecciona la respuesta a una página para informar
 * al usuario.
 *
 * @param   req            petición del cliente para acceder a la sesión
 * @param   res            respuesta para el cliente para redireccionarlo a otra página
 * @param   respuestas     array de string con cada una de las respuestas
 * @param   identif        array de identificador con los identificadores de las
 *                    respuestas
 * @param   max            enteroPositivo con el número máximo de respuestas elegibles
 * @param   correct        array de boolean marcando las respuestas que son correctas
 * @param   fija        array de boolean marcando las respuestas que son fijas
 * @throws  IOException    si hay algún error al redireccionar la respuesta
 * @see     xml.items.ChoiceXML
 * @see     javax.servlet.http.HttpSession
 * @see     javax.servlet.http.HttpServletResponse#sendRedirect(java.lang.String)
 */
private void terminaChoice(HttpServletRequest req, HttpServletResponse res,
      String[] respuestas, Identificador[] identif, EnteroPositivo max,
      boolean [] correct,  boolean[] fija) throws IOException{

   HttpSession sesion = req.getSession();
   // Si el objeto sesión no es nuevo la sesión no ha expirado
   if (!sesion.isNew()){
      // Rellena los campos que faltan en el objeto ChoiceXML que se creó en la
      // primera parte de la toma de datos
      ChoiceXML chXML = (ChoiceXML)sesion.getAttribute("Choice");
      chXML.setRespuestas(respuestas);
      chXML.setIdentificadores(identif);
      chXML.setMaxChoices(max.getEnteroPositivo());
      chXML.setCorrecto(correct);
      chXML.setFija(fija);
      // Almacena el item modificado en sesión para utilizarlo más adelante
      sesion.setAttribute("AItem", chXML);
      // Redirecciona la respuesta a "seleccionasignatura.jsp" indicándole la dirección
      // a la que debe ir después
      res.sendRedirect("seleccionasignatura.jsp?Destino=finitem.jsp");
   } else         // Si la sesión ha expirado
      res.sendRedirect("errorfinsesion.jsp");

}
}
```

Respuestasgapmatch.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import utilidades.*;

import xml.items.GapmatchXML;

/**
 * Componente de control implícito asociado a la página JSP "respuestasgapmatch.jsp". Es la
 * lógica asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que
 * sea llamada la página. Su función es comprobar que todos los parámetros finales de la toma
 * de datos de una pregunta tipo "Gap Match" sean correctos, en cuyo caso redirecciona la
 * respuesta a la página "seleccionasignatura.jsp" para elegir donde guardar el ítem. La clase
 * implementa la interfaz <code>Control</code>, implementando su método <code>doLogic</code>,
 * que es el encargado de realizar la lógica asociada a la página "respuestaschoice.jsp".
 * Tiene otro método que simplemente se ejecuta si los parámetros son correctos para
 * redireccionar la respuesta y terminar de crear el objeto <code>GapmatchXML</code>.
 *
 * @author   David Domínguez
 * @see      Control
 *
 */
```

```
public class Respuestasgapmatch implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página
  * "respuestasgapmatch.jsp". Será llamado por el filtro <code>ControlFilter</code> en el
  * caso de que detecte que la página requerida es "respuestasgapmatch.jsp". Ya que es
  * llamado desde un filtro, será ejecutado siempre que llegue una petición de un cliente y
  * antes de que la petición llegue al Servlet (página JSP compilada a Servlet) que sirve la
  * respuesta al cliente. Comprueba uno por uno todos los parámetros, y, en caso de que sean
  * correctos, llama al método <code>terminaGapmatch</code> para terminar de crear el objeto
  * del ítem y redireccionar la respuesta.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     utilidades.EnteroPositivo
  * @see     utilidades.Identificador
  * @see     utilidades.Comprobaciones
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Nombre base para los identificadores de los huecos por defecto
    final String identificadorBaseHueco = "Hueco";
    request.setAttribute("IdentificadorBaseHueco", identificadorBaseHueco);
    // Nombre base para los identificadores de las opciones por defecto
    final String identificadorBaseOpcion = "Opcion";
    request.setAttribute("IdentificadorBaseOpcion", identificadorBaseOpcion);

    // Estos elementos siempre estarán presentes en request al venir de la página anterior
    String numOpciones = request.getParameter("NumOpc"); // Número de opciones del ítem
    String numHuecos = request.getParameter("NumHuec");      // Número de huecos del ítem
    String shuff = request.getParameter("Shuffle");       // Mezclar respuestas sí o no
    int numOpc = Integer.parseInt(numOpciones);     // Número de opciones y
    int numHuec = Integer.parseInt(numHuecos);      // de huecos en un entero
    // Identificadores de las opciones
    Identificador[] identOpciones = new Identificador[numOpc];
    // Identificadores de los huecos
    Identificador[] identHuecos = new Identificador[numHuec];

    //Comprueba si es la primera vez que se entra en la página
    if(request.getParameter("Form") != null){
       String[] opciones = new String [numOpc];     // Opciones disponibles

       // Estado de la petición web. Almacena si son correctos los parámetros, los
       // mensajes de aviso al usuario y dónde colocar el foco en la página
       EstadoWeb estado = new EstadoWeb();

       int i;
       // Lee los fragmentos de texto que acompañan a los huecos
       String[] fragmentosTexto = new String[numHuec + 1];
       for (i = 0; i < (numHuec + 1); i++)
          fragmentosTexto[i] = request.getParameter("Texto" + i);

       // Lee las opciones introducidas, y si alguna está vacía añade el código de error
       for(i = 0; i < numOpc; i++){
          opciones[i] = request.getParameter("Opcion"+i);
          if (opciones[i] == null || opciones[i].equals(""))
             estado.añadeError("Error: la opción " + (i+1) + " está vacía<br />",
                   "Opcion"+i);
       }

       // Lee los identificadores de las opciones comprobando que sean correctos y no
       // haya ninguno repetido, devolviendo el estado de la web tras la comprobación
       Identificador.testIdentificadores(identOpciones, "IdentOpc", estado, request,
             "En las opciones ");
```

```
      // Lee los identificadores de los huecos comprobando que sean correctos y no haya
      // ninguno repetido, devolviendo el estado de la web tras la comprobación
      Identificador.testIdentificadores(identHuecos, "IdentHuec", estado, request,
            "En los huecos ");

      //Lee los checkbox de opciones en posición fija
      boolean[] respFijaOpc = new boolean[numOpc];
      Comprobaciones.leeCheckBoxes(respFijaOpc, "Fija", request);

      // Comprueba los botones de respuesta correcta que uno esté marcado en cada fila
      String[] respCorrecta = new String[numHuec];// Contendrá el número de la columna
                                          // de la opción que es correcta
      // También suma el número de elementos correctos en cada columna
      int[] numCorrectasOpc = new int[numOpc];  // Número de correctos en cada opción
      for (int j = 0; j < numOpc; j++)
         numCorrectasOpc[j] = 0;    // Inicializa el número de correctas en la columna
      for (i = 0; i < numHuec; i++){
         respCorrecta[i] = request.getParameter("Correcto"+i);
         if (respCorrecta[i] == null)
            // Si no se ha marcado ninguna correcta en esa fila pone el foco en el
            // identificador del hueco correspondiente ya que no se puede en el botón
            // de radio
            estado.añadeError("Debe marcar la opción correcta para  el hueco " +
                  (i+1) +  "<br />", "IdentHuec" + i);
         else  // Aumenta en uno el número de correctas de la columna marcada correcta
            numCorrectasOpc[Integer.parseInt(respCorrecta[i])]++;
      }

      // Lee los valores de la tabla de valores máximos de elección de las opciones
      EnteroPositivo[] maxChoicesOpc = new EnteroPositivo[numOpc];
      String[] max = new String[numOpc];
      for (i = 0; i< numOpc; i++)
         max[i] = request.getParameter("Max"+i);
      Comprobaciones.testTablaMaximos(maxChoicesOpc, "Max", estado, max,
            "de las opciones", "columna", numCorrectasOpc, numHuec);

      //Si todo está bien llama al método que finaliza el ítem y redirecciona al cliente
      if (estado.isEstado() == true){
         terminaGapmatch(request, response, fragmentosTexto, opciones, identHuecos,
               identOpciones, maxChoicesOpc, respCorrecta, respFijaOpc);
      } else {
         // Si ha habido algún fallo pone los datos que ya ha introducido el usuario
         // junto a los mensajes de aviso y el foco en el primer fallo encontrado
         request.setAttribute("Texto", fragmentosTexto);
         request.setAttribute("Opcion", opciones);
         request.setAttribute("IdentHuec", identHuecos);
         request.setAttribute("IdentOpc", identOpciones);
         request.setAttribute("Max", max);
         request.setAttribute("Correcto", respCorrecta);
         request.setAttribute("Fija", respFijaOpc);
         request.setAttribute("Aviso", estado.getMensaje());
         request.setAttribute("Focus",estado.getFocusON());
      }
   } else {
      // Si es la primera vez que se visita la página pone el foco en el primer campo
      request.setAttribute("Focus", "Texto0");
      // Se inicializan los identificadores de las opciones y los huecos a unos valores
      // por defecto
      for (int i = 0; i < identOpciones.length; i++)
         identOpciones[i] = new Identificador(identificadorBaseOpcion + (i+1));
      for (int i = 0; i < identHuecos.length; i++)
         identHuecos[i] = new Identificador(identificadorBaseHueco + (i+1));
      request.setAttribute("IdentHuec", identHuecos);
      request.setAttribute("IdentOpc", identOpciones);
   }

   // El número de huecos, el número de opciones y si se mezclan las respuestas
   // aleatoriamente hacen falta siempre, sea la primera vez que visita la página o no y
   // pone el foco en el primer campo del formulario
   request.setAttribute("NumOpc",numOpciones);
   request.setAttribute("NumHuec",numHuecos);
   request.setAttribute("Shuffle",shuff);

   return true;
}
```

```
/**
 * Este método se ejecuta una vez que se comprueba que todos los parámetros son correctos.
 * Usa el objeto <code>GapmatchXML</code> creado anteriormente y almacenado en sesión
 * para guardar el resto de los datos asociados a la pregunta, redireccionando la
 * respuesta a la página "seleccionasignatura.jsp" para que el usuario seleccione una
 * asignatura en la que guardar el ítem XML. Si el objeto <code>GapmatchXML</code> no se
 * ha podido recuperar de sesión porque ésta ha expirado, redirecciona la respuesta a una
 * página para informar al usuario.
 *
 * @param   req            petición del cliente para acceder a la sesión
 * @param   res            respuesta para el cliente para redireccionarlo a otra página
 * @param   textos         array con los textos que rodean a los huecos
 * @oaram   opc            array de opciones disponibles
 * @param   iHuec       array de identificador con los identificadores de los huecos
 * @param   iOpc        array de identificador con los identificadores de las opciones
 * @param   maxOpc         array de enteroPositivo con el número máximo de opciones que
 *                      puede seleccionar cada hueco
 * @param   correct        array de string con el número de la opción correcta de cada
 *                      hueco
 * @param   fija        array de boolean marcando las opciones que son fijas
 * @throws  IOException    si hay algún error al redireccionar la respuesta
 * @see     xml.items.GapmatchXML
 * @see     javax.servlet.http.HttpSession
 * @see     javax.servlet.http.HttpServletResponse#sendRedirect(java.lang.String)
 */
private void terminaGapmatch(HttpServletRequest req, HttpServletResponse res,
      String[] textos, String[] opc, Identificador[] iHuec, Identificador[] iOpc,
      EnteroPositivo[] maxOpc, String[] correct, boolean[] fija) throws IOException{
   HttpSession sesion = req.getSession();
   // Si el objeto sesión no es nuevo la sesión no ha expirado
   if (!sesion.isNew()){
      // Rellena los campos que faltan en el objeto ChoiceXML que se creó en la
      // primera parte de la toma de datos
      GapmatchXML gmXML = (GapmatchXML)sesion.getAttribute("GapMatch");
      gmXML.setTextos(textos);
      gmXML.setOpciones(opc);
      gmXML.setIdentHuecos(iHuec);
      gmXML.setIdentOpciones(iOpc);
      gmXML.setMaxOpciones(maxOpc);
      gmXML.setRespCorrecta(correct);
      gmXML.setFija(fija);
      // Almacena el ítem modificado en sesión para utilizarlo más adelante
      sesion.setAttribute("AItem", gmXML);
      // Redirecciona la respuesta a "seleccionasignatura.jsp" indicándole la dirección
      // a la que debe ir después
      res.sendRedirect("seleccionasignatura.jsp?Destino=finitem.jsp");
   } else         // Si la sesión ha expirado
      res.sendRedirect("errorfinsesion.jsp");

}
}
```

Respuestashottext.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.*;

import xml.items.HottextXML;

import utilidades.*;

/**
 * Componente de control implícito asociado a la página JSP "respuestashottext.jsp". Es la
 * lógica asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que
 * sea llamada la página. Su función es comprobar que todos los parámetros finales de la toma
 * de datos de una pregunta tipo "Hot Text" sean correctos, en cuyo caso redirecciona la
 * respuesta a la página "seleccionasignatura.jsp" para elegir donde guardar el ítem. La clase
 * implementa la interfaz <code>Control</code>, implementando su método <code>doLogic</code>,
 * que es el encargado de realizar la lógica asociada a la página "respuestashottext.jsp".
```

```
 * Tiene otro método que simplemente se ejecuta si los parámetros son correctos para
 * redireccionar la respuesta y terminar de crear el objeto <code>HottextXML</code>.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Respuestashottext implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página
  * "respuestashottext.jsp". Será llamado por el filtro <code>ControlFilter</code> en el
  * caso de que detecte que la página requerida es "respuestashottext.jsp". Ya que es
  * llamado desde un filtro, será ejecutado siempre que llegue una petición de un cliente y
  * antes de que la petición llegue al Servlet (página JSP compilada a Servlet) que sirve la
  * respuesta al cliente. Comprueba uno por uno todos los parámetros, y, en caso de que sean
  * correctos,  llama al método <code>terminaHottext</code> para terminar de crear el objeto
  * del ítem y redireccionar la respuesta.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see     javax.servlet.http.HttpSession
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     utilidades.EnteroPositivo
  * @see     utilidades.Identificador
  * @see     utilidades.Comprobaciones
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Nombre base para los identificadores de los hottexts por defecto
    final String identificadorBase = "Hottext";
    request.setAttribute("IdentificadorBase", identificadorBase);

    // Este elemento siempre estará presente en request porque viene de la página anterior
    String numHottext= request.getParameter("NumHot"); // Número de hottexts
    int numHot = Integer.parseInt(numHottext);      // número de hottexts en entero
    // Array con los identificadores de los hottexts
    Identificador[] identHottexts = new Identificador[numHot];

    //Comprueba si es la primera vez que se entra en la página
    if(request.getParameter("Form") != null){

       String[] hottexts = new String [numHot];     // Array con los hottexts

       // Estado de la petición web. Almacena si son correctos los parámetros, los
       // mensajes de aviso al usuario y dónde colocar el foco en la página
       EstadoWeb estado = new EstadoWeb();

       int i;
       // Lee los fragmentos de texto que acompañan a los huecos
       String[] fragmentosTexto = new String[numHot + 1];
       for (i = 0; i < (numHot + 1); i++)
          fragmentosTexto[i] = request.getParameter("Texto"+i);

       // Lee los hottexts introducidos, y si alguno está vacío añade el código de error
       for(i = 0; i < numHot; i++){
          hottexts[i] = request.getParameter("Hottext"+i);
          if (hottexts[i] == null || hottexts[i].equals(""))
             estado.añadeError("El hottext " + (i+1)+ " no puede quedar vacío<br />",
                   "Hottext" + i);
       }

       // Lee los identificadores de los hottexts comprobando que sean correctos y no
       // haya ninguno repetido, devolviendo el estado de la web tras la comprobación
       Identificador.testIdentificadores(identHottexts, "IdentHot", estado, request, "");
```

```
      // Comprueba los checkbox de hottext correcto que al menos uno esté marcado y suma
      // el total de respuestas marcadas como correctas
      boolean[] correcta = new boolean[numHot];    // Respuestas correcta o no
      int numCorrectas = Comprobaciones.sumaCorrectas(correcta, "Correcto", request);
      if (numCorrectas == 0)  // Sin ningún checkbox de respuesta correcta está marcado
         estado.añadeError("Debe marcar al menos una opción correcta<br />",
               "Correcto0");

      // Lee el número máximo de elecciones introducido por el usuario
      String max = request.getParameter("Max");
      EnteroPositivo maxChoices = new EnteroPositivo(max);
      // Comprueba que el valor introducido sea un valor correcto, mayor que el número
      // de respuestas correctas o cero, y menor que el número de respuestas posibles
      Comprobaciones.testMaximo(maxChoices, estado, "Max", numCorrectas, numHot);

      //Si todo está bien llama al método que finaliza el ítem y redirecciona al cliente
      if (estado.isEstado() == true){
         terminaGapmatch(request, response, fragmentosTexto, hottexts, identHottexts,
               maxChoices, correcta);
      } else {
         // Si ha habido algún fallo pone los datos que ya ha introducido el usuario
         // junto a los mensajes de aviso y el foco en el primer fallo encontrado
         request.setAttribute("Texto", fragmentosTexto);
         request.setAttribute("Hottexts", hottexts);
         request.setAttribute("IdentHot", identHottexts);
         request.setAttribute("Max", max);
         request.setAttribute("Correcto", correcta);
         request.setAttribute("Aviso", estado.getMensaje());
         request.setAttribute("Focus", estado.getFocusON());
      }
   } else {
      // Si es la primera vez que se visita la página pone el foco en el primer campo
      request.setAttribute("Focus", "Max");
      // Se inicializa los identificadores a unos valores por defecto
      for (int i = 0; i < identHottexts.length; i++)
         identHottexts[i] = new Identificador(identificadorBase + (i+1));
      request.setAttribute("IdentHot", identHottexts);
   }

   // El número de hottexts hace falta siempre, sea la primera vez que visita la página o
   // no y pone el foco en el primer campo del formulario
   request.setAttribute("NumHot",numHottext);

   return true;
}
/**
 * Este método se ejecuta una vez que se comprueba que todos los parámetros son correctos.
 * Usa el objeto <code>HottextXML</code> creado anteriormente y almacenado en sesión para
 * guardar el resto de los datos asociados a la pregunta, redireccionando la respuesta a
 * la página "seleccionasignatura.jsp" para que el usuario seleccione una asignatura en la
 * que guardar el ítem XML. Si el objeto <code>HottextXML</code> no se ha podido recuperar
 * de sesión porque ésta ha expirado, redirecciona la respuesta a una página para informar
 * al usuario.
 *
 * @param   req            petición del cliente para acceder a la sesión
 * @param   res            respuesta para el cliente para redireccionarlo a otra página
 * @param   textos         array con los textos que rodean a los hottexts
 * @oaram   hott        array con los hottexts
 * @param   iHott       array de identificador con los identificadores de los hottexts
 * @param   maxOpc         número máximo de hottexts que se pueden seleccionar
 * @param   correct        array de boolean indicando si el hottext es correcto
 * @throws  IOException    si hay algún error al redireccionar la respuesta
 * @see     xml.items.HottextXML
 * @see     javax.servlet.http.HttpSession
 * @see     javax.servlet.http.HttpServletResponse#sendRedirect(java.lang.String)
 */
private void terminaGapmatch(HttpServletRequest req, HttpServletResponse res,
      String[] textos, String[] hott, Identificador[] iHott,EnteroPositivo max,
      boolean[] correct) throws IOException{

   // Objeto sesión para acceder a los objetos ahí almacenados
   HttpSession sesion = req.getSession();
   // Si el objeto sesión no es nuevo la sesión no ha expirado
   if (!sesion.isNew()){
      // Rellena los campos que faltan en el objeto ChoiceXML que se creó en la
```

```
        // primera parte de la toma de datos
        HottextXML htXML = (HottextXML)sesion.getAttribute("HotText");
        htXML.setTextos(textos);
        htXML.setHottexts(hott);
        htXML.setIdentHottexts(iHott);
        htXML.setMaxChoices(max.getEnteroPositivo());
        htXML.setCorrecto(correct);
        // Almacena el ítem modificado en sesión para utilizarlo más adelante
        sesion.setAttribute("AItem", htXML);
        // Redirecciona la respuesta a "seleccionasignatura.jsp" indicándole la dirección
        // a la que debe ir después
        res.sendRedirect("seleccionasignatura.jsp?Destino=finitem.jsp");
    } else          // Si la sesión ha expirado
        res.sendRedirect("errorfinsesion.jsp");
 }

}
```

Respuestasinlinechoice.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.*;

import xml.items.InlinechoiceXML;

import utilidades.*;

/**
 * Componente de control implícito asociado a la página JSP "respuestasinlinechoice.jsp". Es
 * la lógica asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre
 * que sea llamada la página. Su función es comprobar que todos los parámetros finales de la
 * toma de datos de una pregunta tipo "Inline Choice" sean correctos, en cuyo caso redirecciona
 * la respuesta a la página "seleccionasignatura.jsp" para elegir donde guardar el ítem. La
 * clase implementa la interfaz <code>Control</code>, implementando su método <code>doLogic
 * </code>, que es el encargado de realizar la lógica asociada a la página
 * "respuestasinlinechoice.jsp".Tiene otro método que simplemente se ejecuta si los parámetros
 * son correctos para redireccionar la respuesta y terminar de crear el objeto <code>
 * InlinechoiceXML</code>.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Respuestasinlinechoice implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página
  * "respuestasinlinechoice.jsp". Será llamado por el filtro <code>ControlFilter</code> en
  * el caso de que detecte que la página requerida es "respuestasinlinechoice.jsp". Ya que
  * es llamado desde un filtro, será ejecutado siempre que llegue una petición de un
  * cliente y antes de que la petición llegue al Servlet (página JSP compilada a Servlet)
  * que sirve la respuesta al cliente. Comprueba uno por uno todos los parámetros, y, en
  * caso de que sean correctos, llama al método <code>terminaInlinechoice</code> para
  * terminar de crear el objeto del ítem y redireccionar la respuesta.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     utilidades.Identificador
  * @see     utilidades.Comprobaciones
```

```
 */
public boolean doLogic(HttpServletRequest request,
      HttpServletResponse response) throws IOException, ServletException {

   // Nombre base para los identificadores de las opciones por defecto
   final String identificadorBase = "Opcion";
   request.setAttribute("IdentificadorBase", identificadorBase);

   // Estos elementos siempre estarán presentes en request al venir de la página anterior
   String num_Opc = request.getParameter("Num_Opc");     // Número de respuestas
   String shuff = request.getParameter("Shuff");         // Mezclar respuestas sí o no
   int numOpc = Integer.parseInt(num_Opc);         // Número de opciones en entero
   // Array con los identificadores de las opciones de respuesta
   Identificador[] identificadores = new Identificador[numOpc];

   //Comprueba si es la primera vez que se entra en la página
   if(request.getParameter("Form") != null){

      // Array con las opciones posibles de respuesta
      String[] opciones = new String [numOpc];

      // Estado de la petición web. Almacena si son correctos los parámetros, los
      // mensajes de aviso al usuario y dónde colocar el foco en la página
      EstadoWeb estado = new EstadoWeb();

      // Recoge el valor de los dos campos de texto que pueden estar vacíos
      String texto1 = request.getParameter("Texto1");
      String texto2 = request.getParameter("Texto2");

      int i;
      // Lee las opciones introducidas, y si alguna está vacía añade el código de error
      for(i = 0; i < numOpc; i++){
         opciones[i] = request.getParameter("Opciones"+i);
         if (opciones[i] == null || opciones[i].equals(""))
            estado.añadeError("Error: la opción " + (i+1) + " no puede ser vacía" +
                  "<br />", "Opciones" + i);
      }

      // Lee los identificadores de las opciones comprobando que sean correctos y no
      // haya ninguno repetido, devolviendo el estado de la web tras la comprobación
      Identificador.testIdentificadores(identificadores, "Identificador", estado,
            request, "");

      // Comprueba los botones de radio de respuesta correcta que uno esté marcado
      String respCorrecta = request.getParameter("Correcto");
      if (respCorrecta == null)
         estado.añadeError("Debe marcar una respuesta como correcta<br />",
               "Identificador0");

      //Lee los checkbox de respuesta en posición fija
      boolean[] respFija = new boolean[numOpc];
      Comprobaciones.leeCheckBoxes(respFija, "Fija", request);

      //Si todo está bien llama al método que finaliza el ítem y redirecciona al cliente
      if (estado.isEstado() == true){
         terminaInlinechoice(request, response, opciones, identificadores,
               respCorrecta, respFija, texto1, texto2);
      } else {
         // Si ha habido algún fallo pone los datos que ya ha introducido el usuario
         // junto a los mensajes de aviso y el foco en el primer fallo encontrado
         request.setAttribute("Opciones",opciones);
         request.setAttribute("Identificador",identificadores);
         request.setAttribute("Correcto",respCorrecta);
         request.setAttribute("Fija",respFija);
         request.setAttribute("Texto1", texto1);
         request.setAttribute("Texto2", texto2);
         request.setAttribute("Aviso",estado.getMensaje());
         request.setAttribute("Focus",estado.getFocusON());
      }
   } else {
      // Si es la primera vez que se visita la página pone el foco en el primer campo
      request.setAttribute("Focus", "Texto1");
      // Inicializa los identificadores a unos valores por defecto
      for (int i = 0; i < identificadores.length; i++)
         identificadores[i] = new Identificador(identificadorBase + (i+1));
      request.setAttribute("Identificador",identificadores);
```

```
    }

    // El número de respuestas y si se mezclan las respuestas aleatoriamente hacen falta
    // siempre, sea la primera vez que visita la página o no y pone el foco en el primer
    // campo del formulario
    request.setAttribute("Num_Opc",num_Opc);
    request.setAttribute("Shuff",shuff);

    return true;
 }
 /**
  * Este método se ejecuta una vez que se comprueba que todos los parámetros son correctos.
  * Usa el objeto <code>InlinechoiceXML</code> creado anteriormente y almacenado en sesión
  * para guardar el resto de los datos asociados a la pregunta, redireccionando la
  * respuesta a la página "seleccionasignatura.jsp" para que el usuario seleccione una
  * asignatura en la que guardar el ítem XML. Si el objeto <code>InlinechoiceXML</code> no
  * se ha podido recuperar de sesión porque ésta ha expirado, redirecciona la respuesta a
  * una página para informar al usuario.
  *
  * @param   req            petición del cliente para acceder a la sesión
  * @param   res            respuesta para el cliente para redireccionarlo a otra página
  * @param   respuestas     array de string con cada una de las respuestas
  * @param   identif        array de identificador con los identificadores de las
  *                      respuestas
  * @param   correct        string con el número de la opción que es correcta
  * @param   fija        array de boolean marcando las respuestas que son fijas
  * @param   texto1         string con el texto antes de las opciones
  * @param   texto2         string con el texto después de las opciones
  * @throws  IOException    si hay algún error al redireccionar la respuesta
  * @see     xml.items.InlinechoiceXML
  * @see     javax.servlet.http.HttpSession
  * @see     javax.servlet.http.HttpServletResponse#sendRedirect(java.lang.String)
  */
 private void terminaInlinechoice(HttpServletRequest req, HttpServletResponse res,
       String[] respuestas, Identificador[] identif, String correct, boolean[] fija,
       String texto1, String texto2) throws IOException{
    HttpSession sesion = req.getSession();
    // Si el objeto sesión no es nuevo la sesión no ha expirado
    if (!sesion.isNew()){
       // Rellena los campos que faltan en el objeto ChoiceXML que se creó en la
       // primera parte de la toma de datos
       InlinechoiceXML icXML = (InlinechoiceXML)sesion.getAttribute("InlineChoice");
       icXML.setRespuestas(respuestas);
       icXML.setIdentificadores(identif);
       icXML.setCorrecto(Integer.parseInt(correct));
       icXML.setFija(fija);
       icXML.setTextos(texto1, texto2);
       // Almacena el ítem modificado en sesión para utilizarlo más adelante
       sesion.setAttribute("AItem", icXML);
       // Redirecciona la respuesta a "seleccionasignatura.jsp" indicándole la dirección
       // a la que debe ir después
       res.sendRedirect("seleccionasignatura.jsp?Destino=finitem.jsp");
    } else         // Si la sesión ha expirado
       res.sendRedirect("errorfinsesion.jsp");

 }
}
```

Respuestasmatch.java

```
package control;

import utilidades.*;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.*;

import xml.items.MatchXML;

/**
 * Componente de control implícito asociado a la página JSP "respuestasmatch.jsp". Es la
 * lógica asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que
```

```
 * sea llamada la página. Su función es comprobar que todos los parámetros finales de la toma
 * de datos de una pregunta tipo "Match" sean correctos, en cuyo caso redirecciona la respuesta
 * a la página "seleccionasignatura.jsp" para elegir donde guardar el ítem. La clase implementa
 * la interfaz <code>Control</code>, implementando su método <code>doLogic</code>, que es el
 * encargado de realizar la lógica asociada a la página "respuestasmatch.jsp". Tiene otro
 * método que simplemente se ejecuta si los parámetros son correctos para redireccionar la
 * respuesta y terminar de crear el objeto <code>MatchXML</code>.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Respuestasmatch implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página
  * "respuestasmatch.jsp". Será llamado por el filtro <code>ControlFilter</code> en el caso
  * de que detecte que la página requerida es "respuestasmatch.jsp". Ya que es llamado
  * desde un filtro, será ejecutado siempre que llegue una petición de un cliente y antes de
  * que la petición llegue al Servlet (página JSP compilada a Servlet) que sirve la
  * respuesta al cliente. Comprueba uno por uno todos los parámetros, y, en caso de que sean
  * correctos,  llama al método <code>terminaMatch</code> para terminar de crear el objeto
  * del ítem y redireccionar la respuesta.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     utilidades.EnteroPositivo
  * @see     utilidades.Identificador
  * @see     utilidades.Comprobaciones
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Nombre base para los identificadores de las opciones del grupo 2 por defecto
    final String identificadorBaseGrupo2 = "Grupo2_";
    request.setAttribute("IdentificadorBaseGrupo2", identificadorBaseGrupo2);
    // Nombre base para los identificadores de las opciones del grupo 1 por defecto
    final String identificadorBaseGrupo1 = "Grupo1_";
    request.setAttribute("IdentificadorBaseGrupo1", identificadorBaseGrupo1);

    // Estos elementos siempre estarán presentes en request al venir de la página anterior
    // Número de respuestas del grupo 1
    String numGrupo1= request.getParameter("NumGrupo1");
    // Número de respuestas del grupo 2
    String numGrupo2= request.getParameter("NumGrupo2");
    // Mezclar respuestas sí o no
    String shuff = request.getParameter("Shuffle");
    int numGrup1 = Integer.parseInt(numGrupo1);     // Números de respuestas
    int numGrup2 = Integer.parseInt(numGrupo2);     // de cada grupo en un entero
    // Arrays con los identificadores del grupo 1 y del grupo 2
    Identificador[] identificadores1 = new Identificador[numGrup1];
    Identificador[] identificadores2 = new Identificador[numGrup2];

    //Comprueba si es la primera vez que se entra en la página
    if(request.getParameter("Form") != null){

       // Arrays con las respuestas del grupo 1 y del grupo 2
       String[] respuestas1 = new String [numGrup1];
       String[] respuestas2 = new String [numGrup2];

       // Estado de la petición web. Almacena si son correctos los parámetros, los
       // mensajes de aviso al usuario y dónde colocar el foco en la página
       EstadoWeb estado = new EstadoWeb();

       int i;
```

```
// Lee las respuestas del grupo 1, y si alguna está vacía añade el código de error
for(i = 0; i < respuestas1.length; i++){
   respuestas1[i] = request.getParameter("Grupo1 "+i);
   if (respuestas1[i] == null || respuestas1[i].equals(""))
      estado.añadeError("Debe rellenar la respuesta " + (i+1) + " del grupo 1" +
            "<br />", "Grupo1_" + i);
}
// Lee las respuestas del grupo2, y si alguna está vacía añade el código de error
for(i = 0; i < respuestas2.length; i++){
   respuestas2[i] = request.getParameter("Grupo2_"+i);
   if (respuestas2[i] == null || respuestas2[i].equals(""))
      estado.añadeError("Debe rellenar la respuesta " + (i+1) + " del grupo 2" +
            "<br />", "Grupo2_" + i);
}

// Lee los identificadores del grupo 1 comprobando que sean correctos y no haya
// ninguno repetido, devolviendo el estado de la web tras la comprobación
Identificador.testIdentificadores(identificadores1, "Identificador1_", estado,
      request, "En el grupo 1 ");

// Lee los identificadores del grupo 2 comprobando que sean correctos y no haya
// ninguno repetido, devolviendo el estado de la web tras la comprobación
Identificador.testIdentificadores(identificadores2, "Identificador2_", estado,
      request, "En el grupo 2 ");

// Comprueba los checkbox de respuesta correcta de cada fila que al menos uno esté
// marcado en cada fila y guarda el número total de correctas en cada fila
// Número de respuesas correctas en cada fila
int[] numCorrectas1 = new int[numGrup1];
// Matriz de combinaciones correctas
boolean[][] respCorrecta = new boolean[numGrup1][numGrup2];
for (i = 0; i < numGrup1; i++){
   numCorrectas1[i] = Comprobaciones.sumaCorrectas(respCorrecta[i], "Correcto" +
         i + "_", request);
   if  (numCorrectas1[i] == 0)      // Si el número de correctas de la fila es cero
      estado.añadeError("Error: no ha marcado ninguna respuesta correcta en " +
            "la fila " + (i+1) + "<br />", "Correcto" + i + "_0");
}
// Suma el número de elementos correctos en cada columna para comprobar los
// números máximos de elección del grupo 2
int[] numCorrectas2 = new int[numGrup2];  // Número de correctas en cada columna
for (int j = 0; j < numGrup2; j++){
   numCorrectas2[j] = 0;      // Inicializa el número de correctas en la columna
   for (i = 0; i < numGrup1; i++)
      if (respCorrecta[i][j])
         numCorrectas2[j]++;
}


// Recoge el valor del número máximo de elecciones comprobando que sea válido
// Valores del número máximo de elecciones de cada elemento del grupo 1
EnteroPositivo[] maxChoices1 = new EnteroPositivo[numGrup1];
// Valores del número máximo de elecciones de cada elemento del grupo 2
EnteroPositivo[] maxChoices2 = new EnteroPositivo[numGrup2];

// Lee los valores de los números máximos de elecciones introducidos en la página
// Valores introducidos para el grupo 1
String[] max1 = new String[maxChoices1.length];
for (i = 0; i < maxChoices1.length; i++)
   max1[i] = request.getParameter("Max1_" + i);
// Valores introducidos para el grupo 2
String[] max2 = new String[maxChoices2.length];
for (i = 0; i < maxChoices2.length; i++)
   max2[i] = request.getParameter("Max2_" + i);
// Comprueba el valor introducido para el grupo 1 que sea mayor que el número de
// respuestas correctas o cero, y menor que el número total de respuestas posibles
Comprobaciones.testTablaMaximos(maxChoices1, "Max1_", estado, max1, "del grupo 1",
      "fila", numCorrectas1, numGrup2);
// Comprueba el valor introducido para el grupo 2 que sea mayor que el número de
// respuestas correctas o cero, y menor que el número total de respuestas posibles
Comprobaciones.testTablaMaximos(maxChoices2, "Max2_", estado, max2, "del grupo 2",
      "columna", numCorrectas2, numGrup1);

// Número máximo de elecciones totales
String max = request.getParameter("NumMax");
// Comprueba el valor introducido para el total que sea mayor que el número de
```

```
      // respuestas correctas o cero, y menor que el número total de respuestas posibles
      EnteroPositivo  numMax = new EnteroPositivo(max);
      int totalCorrectas = 0;          // Total de respuestas marcadas como correctas
      for (i = 0; i < numGrup1; i++)
         totalCorrectas += numCorrectas1[i];
      Comprobaciones.testMaximo(numMax, estado, "NumMax", totalCorrectas,
            numGrup1 * numGrup2);

      //Lee los checkbox de respuesta en posición fija
      boolean[] respFija1 = new boolean[numGrup1]; // Respuestas fijas del grupo 1
      boolean[] respFija2 = new boolean[numGrup2]; // Respuestas fijas del grupo 2
      Comprobaciones.leeCheckBoxes(respFija1, "Fija1_",request);
      Comprobaciones.leeCheckBoxes(respFija2, "Fija2_", request);

      //Si todo está bien llama al método que finaliza el ítem y redirecciona al cliente
      if (estado.isEstado() == true){
         terminaMatch(request, response, respuestas1, respuestas2, identificadores1,
               identificadores2, numMax, maxChoices1, maxChoices2, respCorrecta,
               respFija1, respFija2);
      } else {
         // Si ha habido algún fallo pone los datos que ya ha introducido el usuario
         // junto a los mensajes de aviso y el foco en el primer fallo encontrado
         request.setAttribute("Grupo1", respuestas1);
         request.setAttribute("Grupo2", respuestas2);
         request.setAttribute("Identificador1", identificadores1);
         request.setAttribute("Identificador2", identificadores2);
         request.setAttribute("NumMax", max);
         request.setAttribute("Max1", max1);
         request.setAttribute("Max2", max2);
         request.setAttribute("Correcto", respCorrecta);
         request.setAttribute("Fija1", respFija1);
         request.setAttribute("Fija2", respFija2);
         request.setAttribute("Aviso", estado.getMensaje());
         request.setAttribute("Focus", estado.getFocusON());
      }
   } else {
      // Si es la primera vez que se visita la página pone el foco en el primer campo
      request.setAttribute("Focus", "NumMax");
      // Inicializa los identificadores de los dos grupos a unos valores por defecto
      for (int i = 0; i < identificadores1.length; i++)
         identificadores1[i] = new Identificador(identificadorBaseGrupo1 + (i+1));
      for (int i = 0; i < identificadores2.length; i++)
         identificadores2[i] = new Identificador(identificadorBaseGrupo2 + (i+1));
      request.setAttribute("Identificador1", identificadores1);
      request.setAttribute("Identificador2", identificadores2);
   }

   // El número de respuestas en cada grupo y si se mezclan las respuestas aleatoriamente
   // hacen falta siempre, sea la primera vez que visita la página o no y pone el foco en
   // el primer campo del formulario
   request.setAttribute("NumGrupo1",numGrupo1);
   request.setAttribute("NumGrupo2",numGrupo2);
   request.setAttribute("Shuffle",shuff);

   return true;
}
/**
 * Este método se ejecuta una vez que se comprueba que todos los parámetros son correctos.
 * Usa el objeto <code>MatchXML</code> creado anteriormente y almacenado en sesión para
 * guardar el resto de los datos asociados a la pregunta, redireccionando la respuesta a
 * la página "seleccionasignatura.jsp" para que el usuario seleccione una asignatura en la
 * que guardar el ítem XML. Si el objeto <code>MatchXML</code> no se ha podido recuperar
 * de sesión porque ésta ha expirado, redirecciona la respuesta a una página para informar
 * al usuario.
 *
 * @param   req            petición del cliente para acceder a la sesión
 * @param   res            respuesta para el cliente para redireccionarlo a otra página
 * @param   resp1       array de string con cada una de las opciones del grupo 1
 * @param   resp2       array de string con cada una de las opciones del grupo 2
 * @param   identif1    array de identificador con los identificadores del grupo 1
 * @param   identif2    array de identificador con los identificadores del grupo 2
 * @param   max            enteroPositivo con el número máximo de elecciones totales
 * @param   max1        array de enteroPositivo con el número máximo de elecciones de
 *                      cada elemento del grupo 1
 * @param   max2        array de enteroPositivo con el número máximo de elecciones de
 *                      cada elemento del grupo 2
```

```
  * @param   correct        array de boolean de dos dimensiones con las combinaciones
  *                      entre elementos del grupo 1 y del 2 que son correctas
  * @param   fija1       array de boolean indicando las opciones del grupo 1 fijas
  * @param   fija2       array de boolean indicando las opciones del grupo 2 fijas
  * @throws  IOException    si hay algún error al redireccionar la respuesta
  * @see     xml.items.MatchXML
  * @see     javax.servlet.http.HttpSession
  * @see     javax.servlet.http.HttpServletResponse#sendRedirect(java.lang.String)
  */
 private void terminaMatch(HttpServletRequest req, HttpServletResponse res,
       String[] resp1, String[] resp2, Identificador[] identif1, Identificador[] identif2,
       EnteroPositivo max, EnteroPositivo[] max1, EnteroPositivo[] max2,
       boolean[][] correct, boolean[] fija1, boolean[] fija2) throws IOException{
    HttpSession sesion = req.getSession();
    // Si el objeto sesión no es nuevo la sesión no ha expirado
    if (!sesion.isNew()){
       // Rellena los campos que faltan en el objeto ChoiceXML que se creó en la
       // primera parte de la toma de datos
       MatchXML maXML = (MatchXML)sesion.getAttribute("Match");
       maXML.setRespuestas1(resp1);
       maXML.setRespuestas2(resp2);
       maXML.setIdentificadores1(identif1);
       maXML.setIdentificadores2(identif2);
       maXML.setMaxChoices(max.getEnteroPositivo());
       maXML.setMax1(max1);
       maXML.setMax2(max2);
       maXML.setRespCorrecta(correct);
       maXML.setFija1(fija1);
       maXML.setFija2(fija2);
       // Almacena el item modificado en sesión para utilizarlo más adelante
       sesion.setAttribute("AItem", maXML);
       // Redirecciona la respuesta a "seleccionasignatura.jsp" indicándole la dirección
       // a la que debe ir después
       res.sendRedirect("seleccionasignatura.jsp?Destino=finitem.jsp");
    } else         // Si la sesión ha expirado
       res.sendRedirect("errorfinsesion.jsp");

 }
}
```

Respuestastextentry.java

```
package control;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.*;

import xml.items.TextentryXML;

import utilidades.*;

/**
 * Componente de control implícito asociado a la página JSP "respuestastextentry.jsp". Es la
 * lógica asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que
 * sea llamada la página. Su función es comprobar que todos los parámetros finales de la toma
 * de datos de una pregunta tipo "Text Entry" sean correctos, en cuyo caso redirecciona la
 * respuesta a la página "seleccionasignatura.jsp" para elegir donde guardar el ítem. La clase
 * implementa la interfaz <code>Control</code>, implementando su método <code>doLogic</code>,
 * que es el encargado de realizar la lógica asociada a la página "respuestastextentry.jsp".
 * Tiene otro método que simplemente se ejecuta si los parámetros son correctos para
 * redireccionar la respuesta y terminar de crear el objeto <code>TextentryXML</code>.
 *
 *
 * @author   David Domínguez
 * @see      Control
 *
 */
public class Respuestastextentry implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página
  * "respuestashottext.jsp". Será llamado por el filtro <code>ControlFilter</code> en el
```

```
  * caso de que detecte que la página requerida es "respuestastextentry.jsp". Ya que es
  * llamado desde un filtro, será ejecutado siempre que llegue una petición de un cliente y
  * antes de que la petición llegue al Servlet (página JSP compilada a Servlet) que sirve la
  * respuesta al cliente. Comprueba uno por uno todos los parámetros, y, en caso de que sean
  * correctos, llama al método <code>terminaTextentry</code> para terminar de crear el
  * objeto del ítem y redireccionar la respuesta.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     utilidades.MensajeEstado
  * @see     utilidades.EnteroPositivo
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Comprueba si es la primera vez que se entra en la página
    if(request.getParameter("Form") != null){

       // Estado de la petición web. Almacena si son correctos los parámetros, los
       // mensajes de aviso al usuario y dónde colocar el foco en la página
       EstadoWeb estado = new EstadoWeb();

       // Recoge el valor de los dos campos de texto que pueden estar vacíos
       String texto1 = request.getParameter("Texto1");
       String texto2 = request.getParameter("Texto2");

       // Comprueba que se haya introducido la respuesta correcta
       String respuesta = request.getParameter("Respuesta");    // Respuesta
       String longitud = request.getParameter("Longitud");        // Longitud esperada
       if (respuesta == null || respuesta.equals(""))
          estado.añadeError("La respuesta correcta no puede ser una cadena vacía<br />",
                "Respuesta");

       // Procesa la longitud esperada, comprobando que sea válida si se ha introducido
       // alguna. Si no se ha introducido ninguna no pasa nada, no es un campo obligatorio
       EnteroPositivo longEsperada = null;
       MensajeEstado msg = null;
       if (longitud != null && !longitud.equals("")){
          longEsperada = new EnteroPositivo(longitud);
          msg = longEsperada.esPositivo();
          if (!msg.isEstado())    // Si no es un entero positivo
             estado.añadeError("Error: la longitud esperada " + msg.getMensaje() +
                   "<br />", "Longitud");
          else if (longEsperada.getEnteroPositivo() < 1)
             // Si el número es un entero positivo correcto pero es menor que uno
             estado.añadeError("La longitud esperada debe ser mayor o igual a 1<br />",
                   "Longitud");
       }

       // Si todo ha ido bien vamos a generar el archivo xml
       if (estado.isEstado() == true){
          terminaTextentry(request, response, respuesta, longEsperada, texto1, texto2);
       } else {
          // Si ha habido algún fallo pone los datos que ya ha introducido el usuario
          // junto a los mensajes de aviso y el foco en el primer fallo encontrado
          request.setAttribute("Respuesta", respuesta);
          request.setAttribute("Longitud", longitud);
          request.setAttribute("Texto1", texto1);
          request.setAttribute("Texto2", texto2);
          request.setAttribute("Aviso",estado.getMensaje());
          request.setAttribute("Focus",estado.getFocusON());
       }
    } else {
       // Si es la primera vez que se visita la página pone el foco en el primer elemento
       request.setAttribute("Focus","Texto1");
```

```
    }

    return true;
 }
 /**
  * Este método se ejecuta una vez que se comprueba que todos los parámetros son correctos.
  * Usa el objeto <code>TextentryXML</code> creado anteriormente y almacenado en sesión
  * para guardar el resto de los datos asociados a la pregunta, redireccionando la
  * respuesta a la página "seleccionasignatura.jsp" para que el usuario seleccione una
  * asignatura en la que guardar el ítem XML. Si el objeto <code>ChoiceXML</code> no se ha
  * podido recuperar de sesión porque ésta ha expirado, redirecciona la respuesta a una
  * página para informar al usuario.
  *
  * @param   req            petición del cliente para acceder a la sesión
  * @param   res            respuesta para el cliente para redireccionarlo a otra página
  * @param   respuesta      string con la respuesta correcta
  * @param   longitud    enteroPositivo con la longitud esperada de la respuesta
  * @param   texto1         Texto antes del hueco para introducir la respuesta
  * @param   texto2         Texto después del hueco para introducir la respuesta
  * @throws  IOException    si hay algún error al redireccionar la respuesta
  * @see     xml.items.TextentryXML
  * @see     javax.servlet.http.HttpSession
  * @see     javax.servlet.http.HttpServletResponse#sendRedirect(java.lang.String)
  */
 private void terminaTextentry(HttpServletRequest req, HttpServletResponse res,
       String respuesta, EnteroPositivo longitud, String texto1, String texto2)
       throws IOException{
    HttpSession sesion = req.getSession();
    // Si el objeto sesión no es nuevo la sesión no ha expirado
    if (!sesion.isNew()){
       // Rellena los campos que faltan en el objeto ChoiceXML que se creó en la
       // primera parte de la toma de datos
       TextentryXML teXML = (TextentryXML)sesion.getAttribute("TextEntry");
       teXML.setRespuesta(respuesta);
       if (longitud!= null)
          teXML.setLongitud(longitud.getEnteroPositivo());
       teXML.setTextos(texto1, texto2);
       // Almacena el ítem modificado en sesión para utilizarlo más adelante
       sesion.setAttribute("AItem", teXML);
       // Redirecciona la respuesta a "seleccionasignatura.jsp" indicándole la dirección
       // a la que debe ir después
       res.sendRedirect("seleccionasignatura.jsp?Destino=finitem.jsp");
    } else         // Si la sesión ha expirado
       res.sendRedirect("errorfinsesion.jsp");

 }
}
```

Seleccionaritems.java

```
package control;

import java.io.File;
import java.io.IOException;
import java.util.Vector;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import utilidades.ListaArchivos;
import utilidades.ResumenItem;
import utilidades.MensajeEstado;

import xml.test.*;

/**
 * Componente de control implícito asociado a la página JSP "seleccionaritems.jsp". Es la
 * lógica asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que
 * sea llamada la página. Su función es mostrar al usuario los ítems a seleccionar de los
 * directorios de las asignaturas y permitirle elegir los que desea añadir al test, que se
 * mostrarán en la capa de la derecha de la misma página. La clase implementa la interfaz
```

```
 * <code>Control</code>, implementando su método <code>doLogic</code>, que es el encargado de
 * realizar la lógica asociada a la página "seleccionaritems.jsp". Contiene, además, otros
 * métodos para ayudar en las tareas de añadir, eliminar y marcar a los ítems ya seleccionados.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Seleccionaritems implements Control {

 /**
  * Muestra el directorio de los ítems de una asignatura permitiendo navegar por los
  * subdirectorios y elegir los ítems a incluir en el test, que se mostrarán en otra capa de
  * la misma página. Será llamado por el filtro <code>ControlFilter</code> en el caso de que
  * detecte que la página requerida es "seleccionaritems.jsp". Ya que es llamado desde un
  * filtro, será ejecutado siempre que llegue una petición de un cliente y antes de que la
  * petición llegue al Servlet (página JSP compilada a Servlet) que sirve la respuesta al
  * cliente. Comprueba si se ha pulsado el botón de eliminar ítems, el de añadir ítems  o el
  * de introducir los pesos a los ítems ya añadidos.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       si hay algún problema con el acceso a los ficheros de los
  *                         ítems
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see  control.ControlFilter
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see     utilidades.EstadoWeb
  * @see     utilidades.MensajeEstado
  * @see     utilidades.EnteroPositivo
  * @see     xml.items.ChoiceXML
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    HttpSession sesion = request.getSession();
    ServletContext sc =  sesion.getServletContext();
    // Directorio base de trabajo (guardado en "web.xml")
    String dirTrabajo = sc.getInitParameter("Directorio Trabajo");
    // Directorio de asignatura si venimos de la página de selección de asignatura
    String dirAsignatura = request.getParameter("DirectorioAsignatura");
    // Directorio seleccionado a mostrar en este momento
    String dirItemActual = request.getParameter("DirectorioActual");
    // Directorio inicial de los ítems (guardado en "web.xml")
    String dirInicial = sc.getInitParameter("Directorio Items");
    // Directorio superior del actual (null si estamos en el inicial)
    String dirSuperior = null;
    // Si vamos a otra página o seguimos en la misma
    boolean salida = false;
    // Estado de la web y mensajes para el usuario en caso de error
    MensajeEstado msg = new MensajeEstado();

    // Si el objeto sesión no es nuevo la sesión no ha expirado
    if (!sesion.isNew()){
       if (dirItemActual == null){
          // Primera vez que se entra en la página, establece el directorio de inicio
          dirItemActual = "/" + dirAsignatura + dirInicial;
          // Comprueba si existe el directorio base de los ítems, y si no existe intenta
          // crearlo, para mostrarlo aunque sea vacío de ítems
          File dirItems = new File(sc.getRealPath(dirTrabajo) + "/" + dirAsignatura +
                dirInicial);
          if (!dirItems.exists())
             if (!dirItems.mkdir())
                throw new IOException("Error: No se pudo crear el directorio base de " +
                   "los Ítems");
       }

       // Recupera de sesión el vector con los ítems que ya están a añadir
       Vector /* ResumenItem */ itemsAAñadir =
             (Vector)sesion.getAttribute("Items_A_Añadir");
```

```
    // Si aún no se ha creado el vector de los ítems a añadir se crea uno vacío y se
    // guarda en sesión para utilizarlo
    if (itemsAAñadir == null){
       itemsAAñadir = new Vector();
       sesion.setAttribute("Items_A_Añadir", itemsAAñadir);
    }

    // Si se ha pulsado el botón "Añadir" para agregar algún ítem al test
    if (request.getParameter("Añadir") != null){
       // Se llama al método para añadir los ítems elegidos
       añadeItem(request);
    }

    // Si se ha pulsado el botón "Eliminar" para borrar algún ítem del test
    if (request.getParameter("Eliminar") != null){
       // Se llama al método para eliminar los ítems elegidos
       eliminaItem(request);
    }

    // Si se ha pulsado el botón "Introducir Pesos"
    if (request.getParameter("Pesos") != null){
       // Comprueba que el número de ítems a añadir al test sea mayor que cero
       if (itemsAAñadir.size() > 0){
          // Comprueba que el número de ítems que se seleccionan, si se realiza alguna
          // selección, sea igual o mayor que el número de ítems a añadir
          AssessmentTest test = (AssessmentTest) sesion.getAttribute("Test1");
          Selection sel = test.getTestPart().getSeccion().getSeleccion();
          if (sel != null){
             int select = sel.getSeleccion();
             if (select > itemsAAñadir.size()){
                msg.setEstado(false);
                msg.setMensaje("Debe seleccionar un número de ítems mayor o igual" +
                      " que el número de ítems a seleccionar, " + select +
                      "<br />");
             }
          }
       } else {
          msg.setEstado(false);
          msg.setMensaje("El número de ítems del test no puede ser cero<br />");
       }
       if (msg.isEstado() == true) {
          // Se borra de sesión el vector de los ítems disponibles para añadir que ya
          // no es necesario
          sesion.removeAttribute("Items");
          // Convierte el Vector con los ítems a añadir en un array pues su tamaño ya
          // no se va a modificar más y lo almacena en sesión para utilizarlo luego
          ResumenItem[] riArray = new ResumenItem[itemsAAñadir.size()];
          riArray = (ResumenItem[])itemsAAñadir.toArray(riArray);
          sesion.removeAttribute("Items A Añadir");
          sesion.setAttribute("Items A Añadir", riArray);
          // Se redirecciona la respuesta a la página siguiente
          response.sendRedirect("introducirpesos.jsp");
          salida = true;
       }
    }
 } else   {     // Si la sesión ha expirado
    response.sendRedirect("errorfinsesion.jsp");
    salida = true;
 }

 // Si se sigue en esta página muestra los directorios para poder navegar por ellos
 if (salida != true){
    // Busca la segunda localización del carácter '/' para ver si estamos en el inicio
    int loc = dirItemActual.indexOf('/', 1);
    // Si longitud de la dirección actual es distinta de la que hay del inicio al
    // segundo  carácter '/' más la del directorio inicial de los ítems no estamos en
    // el base, y por lo tanto mostramos la opción de ir al directorio superior
    if (loc + dirInicial.length() != dirItemActual.length()) {
       // Si no estamos en el inicial guarda el nombre del directorio padre para
       // poder volver atrás
       dirSuperior = dirItemActual.substring(0, dirItemActual.lastIndexOf('/'));
    }

    // Guarda en request los nombres del directorio actual y el del directorio padre
    request.setAttribute("DirectorioActual", dirItemActual);
    request.setAttribute("DirectorioSuperior", dirSuperior);
```

```
      // Lista los nombres de los directorios del directorio actual y los guarda en
      // request
      File dirActual = new File(sc.getRealPath(dirTrabajo) + dirItemActual);
      Vector /* String */ dirNames = ListaArchivos.ListaDirectorios(dirActual);
      request.setAttribute("NombresDirectorios",dirNames);

      // Lista los nombres y los títulos de los ítems del directorio actual y los guarda
      // en sesión para añadir el correspondiente al seleccionado en la página
      Vector /* ResumenItem */ items = ListaArchivos.ListaItems(dirActual);
      sesion.setAttribute("Items", items);

      // Boolean con los ítems que están ya seleccionados de la lista de disponibles
      boolean[] itemsSeleccionados = marcaItemsSeleccionados(items, request);

      // Guarda en request el array marcando los ítems de los disponibles que ya se han
      // seleccionado
      request.setAttribute("Seleccionados", itemsSeleccionados);

      // Guarda en request los mensajes de aviso para el usuario
      request.setAttribute("Aviso", msg.getMensaje());
   }

   return true;
}

/**
 * Añade los ítems que se han seleccionado en la página marcando sus checkboxes, a la lista
 * de ítems a añadir que se encuentra en sesión. Recibe sólo el parámetro <code>request
 * </code> para acceder a la petición que guarda los ítems marcados, y a sesión, donde se
 * guarda la lista de ítems a añadir.
 *
 * @param   request     petición del usuario para acceder a los ítems marcados y a la sesión
 */
private void añadeItem(HttpServletRequest request){
   // Objeto sesión para acceder a los objetos ahí almacenados
   HttpSession sesion = request.getSession();
   // Recupera los ítems que están actualmente disponibles para añadir
   Vector /* ResumenItem*/ itemsDisponibles = (Vector)sesion.getAttribute("Items");
   // Se recupera de sesión el vector con los ítems ya elegidos que están a añadir
   Vector /* ResumenItem */ itemsAAñadir = (Vector)sesion.getAttribute("Items_A_Añadir");
   // Número de ítems añadidos
   int num = 0;
   // Comprueba todos los ítems a ver si se ha marcado alguno para añadir
   for (int i = 0; i < itemsDisponibles.size(); i++){
      String check = request.getParameter("Añade_" + i);
      // Si ese ítem se ha marcado para añadir
      if (check != null && check.equalsIgnoreCase("ON")){
         // Se recupera el ítem a añadir, que está en la misma posición en el vector
         // que la posición indicada por el checkbox marcado
         ResumenItem ri = (ResumenItem)itemsDisponibles.elementAt(i);
         // Se añade el ítem al vector de los ítems a añadir
         itemsAAñadir.add(ri);
         num++;
      }
   }
   // Mensajes de información al usuario
   if (num > 1)
      request.setAttribute("Informacion", "Ítems añadidos con éxito<br />");
   else if(num == 1)
      request.setAttribute("Informacion", "Ítem añadido con éxito<br />");
   else
      request.setAttribute("Informacion", "No ha seleccionado ningún ítem<br />");

   // Deja en sesión el Vector de ítems a añadir para terminar de añadirlos al Test en la
   // página siguiente
   sesion.setAttribute("Items_A_Añadir", itemsAAñadir);
}

/**
 * Elimina los ítems que se han seleccionado en la página marcando sus checkboxes, de la
 * lista de ítems a añadir que se encuentra en sesión. Recibe sólo el parámetro <code>
 * request</code> para acceder a la sesión, donde se guarda la lista de ítems a añadir.
 *
 * @param   request     petición del usuario para acceder a los ítems marcados y a la sesión
 */
```

```
private void eliminaItem(HttpServletRequest request){
   // Objeto sesión para acceder a los objetos ahí almacenados
   HttpSession sesion = request.getSession();
   // Recupera los ítems que están elegidos actualmente a añadir
   Vector /* ResumenItem */ itemsElegidos = (Vector)sesion.getAttribute("Items_A_Añadir");
   // Número de ítems eliminados
   int num = 0;
   // Número de ítems a comprobar
   int numItems = itemsElegidos.size();
   // Comprueba todos los ítems a ver si se ha marcado alguno para eliminar
   for (int i = 0; i < numItems; i++){
      String check = request.getParameter("Elimina_" + i);
      // Si ese ítem se ha marcado para eliminar
      if (check != null && check.equalsIgnoreCase("ON")){
         // Se elimina el ítem, que está en la misma posición en el vector
         // que la posición indicada por el checkbox marcado
         itemsElegidos.remove(i - num);
         num++;
      }
   }
   // Mensajes de información al usuario
   if (num > 1)
      request.setAttribute("Informacion", "Ítems eliminados con éxito<br />");
   else if(num == 1)
      request.setAttribute("Informacion", "Ítem eliminado con éxito<br />");
   else
      request.setAttribute("Informacion", "No ha seleccionado ningún ítem<br />");

   // Deja en sesión el Vector de ítems a añadir para terminar de añadirlos al Test en la
   // página siguiente
   sesion.setAttribute("Items_A_Añadir", itemsElegidos);
}

/**
 * Rellena un array de <code>boolean</code> indicando con true la posición de los elementos
 * del vector de ítems disponibles para seleccionar que ya se han seleccionado. Mediante la
 * petición se accede al objeto de sesión para recoger el objeto de ítems seleccionados a
 * añadir.
 *
 * @param   itemsDisponibles  vector de ítems disponibles para añadir
 * @param   request           petición del usuario para acceder a los ítems seleccionados
 *                        a añadir
 * @return                array de boolean indicando qué ítems se han seleccionado ya
 */
private boolean[] marcaItemsSeleccionados(Vector itemsDisponibles,
      HttpServletRequest request) {

   // Array de boolean para marcar los elementos del vector de ítems disponibles que ya se
   // han seleccionado
   boolean[] seleccionados = new boolean[itemsDisponibles.size()];
   // Objeto sesión para acceder a los objetos ahí almacenados
   HttpSession sesion = request.getSession();
   // Recupera los ítems que están elegidos actualmente a añadir
   Vector /* ResumenItem */ itemsElegidos = (Vector)sesion.getAttribute("Items_A_Añadir");

   // Comprueba qué ítems de los disponibles están ya seleccionados
   for (int i = 0; i < itemsDisponibles.size(); i++){
      ResumenItem itemDisp = (ResumenItem)itemsDisponibles.elementAt(i);
      // Nombre del ítem disponible
      String disponible = itemDisp.getArchivo().getPath();
      boolean encontrado = false;
      for (int j = 0; j < itemsElegidos.size() && encontrado == false; j++){
         ResumenItem itemEle = (ResumenItem)itemsElegidos.elementAt(j);
         // Nombre del ítem elegido a añadir
         String elegido = itemEle.getArchivo().getPath();
         if (disponible.equals(elegido)){
            seleccionados[i] = true;
            encontrado = true;
         }
         else
            seleccionados[i] = false;
      }
   }
   return seleccionados;
}
}
```

Seleccionasignatura.java

```
package control;

import java.io.File;
import java.io.IOException;
import java.util.Vector;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import utilidades.ListaArchivos;

/**
 * Componente de control implícito asociado a la página JSP "seleccionasignatura.jsp". Es la
 * lógica asociada a esa página JSP. Su método <code>doLogic</code> se ejecutará siempre que
 * sea llamada la página. Se encarga de pedir al usuario una asignatura de trabajo.
 *
 * @author   David Domínguez
 * @see      control.Control
 *
 */
public class Seleccionasignatura implements Control {

 /**
  * Será llamado por el filtro <code>ControlFilter</code>" en el caso de que detecte que la
  * página requerida es "seleccionasignatura.jsp". Ya que es llamado desde un filtro, será
  * ejecutado siempre que llegue una petición de un cliente y antes de que la petición
  * llegue al Servlet (página JSP compilada a Servlet) que sirve la respuesta al cliente.
  * Pide al usuario que seleccione una asignatura en la que realizar las acciones, y le
  * permite crear nuevos directorios de asignaturas. Si todo es correcto permite seleccionar
  * una asignatura e ir a la página que indique el parámetro recibido en request como <code>
  * Destino</code>.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se le envíe la respuesta correctamente
  * @throws  IOException       si hay algún problema con el acceso a los directorios de
  *                         asignatura
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     java.io.File
  * @see     java.util.Vector
  * @see     utilidades.ListaArchivos#ListaDirectorios(File)
  *
  */
 public boolean doLogic(HttpServletRequest request,
       HttpServletResponse response) throws IOException, ServletException {

    // Aviso recibido de otra página a mostrar en esta
    String aviso = request.getParameter("Aviso");
    // Informacion recibida de otra página a mostrar en esta
    String informacion = request.getParameter("Informacion");
    ServletContext sc =  request.getSession().getServletContext();
    // Directorio inicial de trabajo (guardado en "web.xml")
    String dirTrabajo = sc.getInitParameter("Directorio Trabajo");
    // Página a la que ir una vez seleccionada la asignatura
    String destino = request.getParameter("Destino");

    // El directorio de las asignaturas es el directorio base de trabajo
    File dirAsignaturas = new File(sc.getRealPath(dirTrabajo));

    // Si no existe el directorio de trabajo lo intenta crear
    if (!dirAsignaturas.exists())
       if (!dirAsignaturas.mkdir())
          throw new IOException("Error: No se puede crear el directorio base de " +
                "trabajo");

    // Obtiene una lista de los nombre de las asignaturas y lo guarda en request
```

```
    Vector /* String */ dirNames = ListaArchivos.ListaDirectorios(dirAsignaturas);
    request.setAttribute("NombresAsignaturas",dirNames);

    // Almacena en request el nombre del directorio de trabajo y de la página a la que ir
    // después
    request.setAttribute("DirectorioTrabajo", dirTrabajo);
    request.setAttribute("Destino", destino);

    // Avisos e información para el usuario de la página
    request.setAttribute("Aviso", aviso);
    request.setAttribute("Informacion", informacion);

    return true;
 }
}
```

Textentry.java

```
package control;

import utilidades.MensajeEstado;
import utilidades.EstadoWeb;
import utilidades.ParserNeutro;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import xml.items.TextentryXML;

/**
 * Componente de control implícito asociado a la página JSP "textentry.jsp". Es la lógica
 * asociada a esa página JSP. Se ejecutará su método <code>doLogic</code> siempre que sea
 * llamada la página. Su función es comprobar que todos los parámetros iniciales de la toma de
 * datos de una pregunta tipo "Text Entry" sean correctos, en cuyo caso redirecciona la
 * respuesta a la siguiente página de toma de datos de tipo "Text Entry". La clase implementa
 * la interfaz <code>Control</code>, implementando su método <code>doLogic</code>, que es el
 * único que contiene, y que es el encargado de realizar la lógica asociada a la página
 * "textentry.jsp".
 *
 * @author   David Domínguez
 * @see      Control
 *
 */
public class Textentry implements Control {

 /**
  * Realiza la comprobación de los parámetros que se recogen en la página "textentry.jsp".
  * Será llamado por el filtro <code>ControlFilter</code> en el caso de que detecte que la
  * página requerida es "textentry.jsp". Ya que es llamado desde un filtro, será ejecutado
  * siempre que llegue una petición de un cliente y antes de que la petición llegue al
  * Servlet (página JSP compilada a Servlet) que sirve la respuesta al cliente. Comprueba
  * uno por uno todos los parámetros, y, en caso de que sean correctos, crea un objeto <code>
  * TextentryXML</code> para guardar los datos asociados al tipo de pregunta y lo guarda en
  * el ámbito de sesión para utilizarlos más adelante para terminar de crear el ítem.
  *
  * @param   request           la petición http enviada por el cliente
  * @param   response       la respuesta http a enviar desde el servidor
  * @return                 true para que se sigan ejecutando los demás filtros del
  *                         servidor y se envíe la respuesta correctamente
  * @throws  IOException       no debe de producirse
  * @throws  ServletException  si hay algún problema accediendo a request o response
  * @see
 control.Control#doLogic(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletRe
sponse)
  * @see     javax.servlet.http.HttpServletRequest
  * @see     javax.servlet.http.HttpServletResponse
  * @see  control.ControlFilter
  * @see     utilidades.EstadoWeb
  * @see     xml.items.TextentryXML
  */
```

```
public boolean doLogic(HttpServletRequest request,
      HttpServletResponse response) throws IOException, ServletException {

   // Comprueba que no sea la primera vez que se visita la página
   if (request.getParameter("Form") != null){

      // Caracteres reservados no permitidos
      final String reservados = ";/?:@&=+$,\\*\"<>|";

      String identificador = request.getParameter("Identif"); // Identificador del ítem
      String titulo = request.getParameter("Tit");       // Título del ítem
      String instrucciones = request.getParameter("Instruc"); // Instrucciones del ítem

      // Estado de la petición web. Almacena si son correctos los parámetros, los
      // mensajes de aviso al usuario y dónde colocar el foco en la página
      EstadoWeb estado = new EstadoWeb();

      // Comprueba que estén introducidos todos los atributos obligatorios
      if (identificador != null && !identificador.equals("") &&
            titulo != null && !titulo.equals("")){

         // Comprueba que los caracteres del identificador estén permitidos
           for(int i = 0; i < identificador.length() && estado.isEstado() == true; i++) {
            char c = identificador.charAt(i);   // Lee el siguiente carácter
            int marca = reservados.indexOf(c);
            if (marca >= 0 ){
               estado.añadeError("Error: El carácter '" + reservados.charAt(marca) +
                     "' en la posición " + (i+1) + " del Identificador no está " +
                     "permitido<br />", "Identif");
            }
           }

         // Comprueba que las instrucciones sean un código XHTML válido
         if (instrucciones != null && !instrucciones.equals("")){
            MensajeEstado msg = ParserNeutro.compruebaInstruccionesXHTML(instrucciones);
            if (msg.isEstado() == false)
               estado.añadeError(msg.getMensaje(), "Instruc");
         }

         // Si el estado es todo correcto pasa a generar el objeto contenedor del ítem
         if ( estado.isEstado()){
            // Crea un objeto que contiene las características del ítem Text Entry a
            // crear
            TextentryXML teXML = new TextentryXML(identificador, titulo,
                  instrucciones);
            // Guarda el objeto Textentry en sesión para seguir utilizándolo luego
            HttpSession sesion = request.getSession();
            sesion.setAttribute("TextEntry", teXML);
            // Redirecciona a la página de introducir la respuesta
            response.sendRedirect("respuestastextentry.jsp");
         }
      }
      // si no están todos los atributos obligatorios comprueba los que fallan
      // para informar al usuario
      else{
         if (identificador.equals(""))
            estado.añadeError("Debe introducir un Identificador<br />", "Identif");
         if (titulo.equals(""))
            estado.añadeError("Debe introducir un Título<br />", "Tit");
      }
      // Si ha habido algún fallo pone los datos que ya ha introducido el usuario junto
      // a los mensajes de aviso y el foco en el primer fallo encontrado
      if (estado.isEstado() == false){
         request.setAttribute("Aviso",estado.getMensaje());
         request.setAttribute("Focus",estado.getFocusON());
         request.setAttribute("Identif",identificador);
         request.setAttribute("Tit",titulo);
         request.setAttribute("Instruc",instrucciones);
      }
   } else {
      // Si es la primera vez que se visita la página borra el objeto textentryXML del
      // ámbito de sesión para iniciar desde cero el ítem y pone el foco en el primer
      // campo del formulario
      HttpSession sesion = request.getSession();
      sesion.removeAttribute("TextEntry");
      request.setAttribute("Focus", "Identif");
```

```
    }

    return true;
 }
}
```

2.2 Paquete `utilidades`

Comprobaciones.java

```
package utilidades;

import javax.servlet.http.HttpServletRequest;

/**
 * Clase con diversos métodos estáticos para leer los datos introducidos en las páginas de
 * tomas de datos de los ítems y el test y comprobar, en aquellos que haga falta, que sean
 * correctos.
 *
 * @author David Domínguez
 *
 */
public class Comprobaciones{

 /**
  * Método para ayudar a leer un conjunto de checkboxes con un nombre base común de una
  * página Web informando del número que están marcadas. Lee los checkboxes recogidos en una
  * página web con nombre base común, y rellena un <code> array</code> de boolean con las
  * respuestas correctas, devolviendo el número de respuestas correctas totales.
  *
  * @param   respCorrecta   array de boolean a rellenar marcando las respuestas correctas
  * @param   origen         nombre base de los checkboxes a leer
  * @param   request        petición recibida del cliente donde están almacenados las
  *                      respuestas correctas marcadas por el usuario
  * @return              entero con el número total de respuestas marcadas correctas
  *
  * @see     javax.servlet.http.HttpServletRequest#getParameter
  * @see     utilidades.EstadoWeb#añadeError(String, String)
  */
 public static int sumaCorrectas (boolean[] respCorrecta, String origen,
       HttpServletRequest request){

    int numCorrectas = 0;      // Inicializa el número de respuestas correctas a cero
    String check;           // Variable intermedia para leer el valor de los checkboxes

    for (int i = 0; i < respCorrecta.length; i++){
       check = request.getParameter(origen + i);    // Lee el checkbox i
       if (check != null && check.equalsIgnoreCase("ON")){
          // Si el checkbox es igual a "ON" la respuesta es correcta y se incrementa el
          // número de respuestas correctas
          respCorrecta[i] = true;
          numCorrectas++;
       }
       else
          // Si no es igual a "ON", respuesta incorrecta
          respCorrecta[i] = false;
    }


    return numCorrectas;
 }

 /**
  * Método para ayudar a leer un conjunto de checkboxes con un nombre base común de una
  * página Web. Lee un conjunto de checkboxes inicializando la tabla de <code>boolean</code>
  * que recibe como parámetro en función de si está cada checkbox marcado o no.
  *
  * @param   respFija array de boolean con las respuestas fijas
  * @param   origen      nombre base de los checkbox de respuesta fija
  * @param   request     petición del cliente
  * @see     javax.servlet.http.HttpServletRequest#getParameter
  */
 public static void leeCheckBoxes(boolean[] respFija, String origen,
```

```
      HttpServletRequest request){
   String checkValor;
   for (int i = 0; i < respFija.length; i++){
      checkValor = request.getParameter(origen + i);
      if (checkValor != null && checkValor.equalsIgnoreCase("ON"))
         respFija[i] = true;
      else
         respFija[i] = false;
   }

}

/**
 * Comprueba que el número máximo de elecciones introducido sea correcto en preguntas con
 * un solo grupo de respuestas y entre las que se puede seleccionar más de una como
 * correcta. Comprueba que el valor introducido sea un entero positivo correcto mediante
 * el método <code>esPositivo</code> de <code>EnteroPositivo</code>, mayor que el número
 * de opciones marcadas como correctas o cero, y menor que el número total de opciones de
 * respuesta disponibles.
 *
 * @param   maxChoices     enteroPositivo inicializado con el string introducido por el
 *                         usuario con el número máximo de elecciones a comprobar
 * @param   estado         estadoWeb, representando el estado de la web. Se modifica con
 *                         el estado tras la comprobación del número máximo de elecciones.
 * @param   campo          campo de la web de donde se ha leído el número máximo de
 *                         elecciones indicando dónde poner el foco en caso de error
 * @param   numCorrectas   número de respuestas marcadas como correctas totales
 * @param   numMaxOpciones número máximo de respuestas posibles que tiene el candidato
 *                         para elegir
 * @see     utilidades.MensajeEstado
 * @see     utilidades.EstadoWeb#añadeError(String, String)
 * @see     utilidades.EnteroPositivo
 */
public static void testMaximo(EnteroPositivo maxChoices, EstadoWeb estado, String campo,
      int numCorrectas, int numMaxOpciones){

   MensajeEstado msg = maxChoices.esPositivo();
   if (!msg.isEstado())    // Si el número no es un entero positivo válido
      estado.añadeError("Error: el número máximo de elecciones " + msg.getMensaje() +
            "<br />", campo);
   else if (maxChoices.getEnteroPositivo() != 0 &&
         maxChoices.getEnteroPositivo() < numCorrectas)
      estado.añadeError("El número máximo de elecciones debe ser mayor o igual que el " +
            "número de respuestas correctas o cero", campo);
   else if (maxChoices.getEnteroPositivo() > numMaxOpciones)
      estado.añadeError("El número máximo de elecciones debe ser menor o igual que el " +
            "número de respuestas posibles<br />", campo);
}

/**
 * Comprueba los números máximos de elecciones introducidos para una pregunta en la que
 * hay varios grupos de opciones, y donde hay al menos un conjunto de elementos en el que
 * cada elemento tiene su propio número máximo de elecciones con elementos de otro grupo.
 * Comprueba todos los elementos de un grupo uno por uno, examinando que el valor
 * introducido sea un entero positivo correcto mediante el método <code>esPositivo</code>
 * de <code>EnteroPositivo</code>, mayor que el número de opciones marcadas como correctas
 * o cero, y menor que el número total de opciones de respuesta disponibles.
 *
 * @param   maxChoices     array de enteroPositivo con el número máximo de elecciones de
 *                         cada elemento del grupo a rellenar con los string leídos de la
 *                         página web contenidos en el parámetro <code>max</code>
 * @param   campo          nombre base del campo de la web de donde se ha leído el número
 *                         máximo de elecciones y dónde poner el foco en caso de error
 * @param   estado         estadoWeb, representando el estado de la web. Se modifica con
 *                         el estado tras las comprobaciones
 * @param   max            array de string con los número máximos de elecciones leídos de
 *                         la página web
 * @param   grupo          string con el grupo al que pertenecen los números máximos de
 *                         elecciones. Para rellenar correctamente los avisos al usuario.
 * @param   elemento       Si tratamos a las filas o las columnas. Para rellenar
 *                         correctamente los avisos al usuario en caso de error.
 * @param   numCorrectas   array con el número de respuestas marcadas como correctas de
 *                         cada elemento
 * @param   numOpciones    número máximo de opciones posibles a elegir como correctas
 * @see     utilidades.MensajeEstado
 * @see     utilidades.EstadoWeb#añadeError(String, String)
```

```
  * @see     utilidades.EnteroPositivo
  */
 public static void testTablaMaximos(EnteroPositivo[] maxChoices, String campo,
          EstadoWeb estado, String[] max, String grupo, String elemento,
          int[] numCorrectas, int numOpciones){

    // Cadena con el nombre de los elementos opuestos
    String elemOpuesto;
    if (elemento.equals("fila"))
          elemOpuesto = "columna";
    else
          elemOpuesto = "fila";

    for (int i = 0; i < maxChoices.length; i++){
       maxChoices[i] = new EnteroPositivo(max[i]);
       // Comprueba el número de elecciones, que sea un entero correcto, que no sea menor
       // que el número de elementos marcados si es distinto de cero y que no sea mayor
       // que el número de elementos disponibles
       MensajeEstado msg = maxChoices[i].esPositivo();
       if (!msg.isEstado())    // Si el número no es un entero positivo válido
          estado.añadeError("Error: el número máximo de elecciones de la " + elemento +
                " " + (i+1) + " " + msg.getMensaje() + "<br />", campo + i);
       else if (maxChoices[i].getEnteroPositivo() != 0 &&
             maxChoices[i].getEnteroPositivo() < numCorrectas[i])
          estado.añadeError("El número máximo de elecciones de la " + elemento + " " +
                (i+1) +  " " + grupo + " debe ser mayor o igual que el número de " +
                "respuestas " +   "correctas en esa " + elemento + " o cero<br />",
                campo+ i);
       else if (maxChoices[i].getEnteroPositivo() > numOpciones)
          estado.añadeError("El número máximo de elecciones de la " + elemento + " " +
                (i+1) +  " " + grupo + " debe ser menor que el número de " +
                elemOpuesto + "s<br />",
                campo + i);
    }

 }
}
```

EnteroPositivo.java

```
package utilidades;

/**
 * Clase que contiene la representación de un número entero positivo. Se encarga de recibirlo
 * como un <code>String</code> y de comprobar que sea efectivamente un número entero positivo.
 * Contiene métodos para almacenar y recoger el número, e informa de cualquier anomalía en él.
 *
 * @author David Domínguez
 *
 */
public class EnteroPositivo {
 /**
  * Variable que contiene el número en un String.
  */
 private String stringNumero = null;
 /**
  * Variable con el número ya en un entero.
  */
 private int enteroPositivo;

 /**
  * Constructor que recibe la representación del número en un <code>String</code>. Lo
  * almacena en la variable de clase dedicada a tal efecto.
  *
  * @param   stringNumero   número a almacenar en la clase como un String.
  */
 public EnteroPositivo(String stringNumero) {
    super();
    this.stringNumero = stringNumero;
 }

 /**
  * Método "get" para el número entero positivo como un entero.
  * ATENCIÓN: Antes de usarlo se debe haber comprobado que el número es un entero realmente
```

```
 * mediante el método <code>esPositivo</code>, y en caso de que devuelva un <code>
 * MensajeEstado</code> que indique validez, podrá usarse este método.
 *
 * @return     entero representando el valor que almacena la clase
 */
public int getEnteroPositivo() {
   return enteroPositivo;
}

/**
 * Método "get" para el número entero positivo almacenado como un <code>String</code>.
 * Se puede usar siempre, pero sin usar el método <code>esPositivo</code> no se tendrá
 * seguridad de que el <code>String</code> represente un número entero positivo válido.
 *
 * @return     String con el valor almacenado en la clase
 */
public String getStringNumero() {
   return stringNumero;
}

/**
 * Comprueba que el número <code>String</code> almacenado sea un número entero positivo.
 * Comprueba que no sea una cadena vacía, que sea un número entero, y que sea cero o mayor
 * que cero, es decir, un número entero positivo válido.
 *
 * @return     mensajeEstado indicando si ha ido bien la comprobación, en cuyo caso no
 *             incluye  mensajes, o, si ha ido mal, indicando en el mensaje el error
 *             ocurrido en la comprobación
 * @see        utilidades.MensajeEstado
 * @see        Integer#parseInt(java.lang.String)
 */
public MensajeEstado esPositivo(){
   MensajeEstado msg = new MensajeEstado();  // Mensaje de salida con el estado final

   if (stringNumero != null && !stringNumero.equals("")){
      // Si la cadena no está vacía
      try {
         enteroPositivo = Integer.parseInt(stringNumero);
         if (enteroPositivo < 0){
            // Si el número es menor que cero (negativo)
            msg.setMensaje("es un número menor que cero");
            msg.setEstado(false);
         }
      } catch (NumberFormatException e){
         // Si la cadena no contiene un número entero válido captura la excepción
         msg.setMensaje("es un entero positivo no válido");
         msg.setEstado(false);
      }
   } else {
      // Si la cadena está vacía
      msg.setMensaje("está vacío");
      msg.setEstado(false);
   }

   // Devuelve el mensaje con el estado de la comprobación
   return msg;
}
}
```

EstadoWeb.java

```
package utilidades;

/**
 * Clase que representa el estado de comprobación de una Web. La clase hereda de <code>
 * MensajeEstado</code>, añadiéndole la funcionalidad de almacenar además en qué componente del
 * formulario de la página web se debe localizar el foco en caso de error. Se encarga de
 * almacenar el estado de la comprobación de los datos en cualquier toma de datos de creación
de
 * un ítem o un test, y, en caso de error, contiene los mensajes de aviso a mostrar al usuario
y
 * dónde posicionar el foco en la página.
 *
 * @author   David Domínguez
```

```
 * @see      utilidades.MensajeEstado
*
 */
public class EstadoWeb extends MensajeEstado {

 /**
  * Nombre del campo erróneo sobre el que irá el foco en la página.
  */
 protected String focusON;

 /**
  * El constructor llama al constructor de la superclase e inicializa la posición del foco
  * a null.
  *
  * @see      utilidades.MensajeEstado
  */
 public EstadoWeb() {
    super();
    focusON = null;
 }

 /**
  * Método "get" para obtener la posición del foco. Obtiene el <code>String</code> que lo
  * almacena.
  *
  * @return  string con el nombre del campo del formulario en el que localizar el foco en
  *          caso de  error
  */
 public String getFocusON() {
    return focusON;
 }

 /**
  * Añade un error al estado de la comprobación. Al añadir el error se añade el nuevo aviso
  * al ya existente, se pone el estado en falso, ya que añade un error, y, si el foco aún
  * no se ha inicializado a ningún valor, lo inicializa al que recibe como parámetro. Con
  * esto logra que foco se posicione en el primer error encontrado y no varíe al añadir
  * más errores.
  *
  * @param   aviso mensaje de aviso a añadir a los existentes
  * @param   focus campo erróneo en el que posicionar el foco en la página
  * @see     utilidades.MensajeEstado
  */
 public void añadeError(String aviso, String focus){
    this.mensaje += aviso;        // Añade el nuevo aviso a los ya existentes
    this.estado = false;       // El estado es incorrecto ya que se añade un error
    if (this.focusON == null)     // Si el foco aún no se ha inicializado se inicializa
       this.focusON = focus;      // al valor recibido en el parámetro
 }
}
```

Identificador.java

```
package utilidades;

import javax.servlet.http.HttpServletRequest;

/**
 * Clase que almacena la representación de un tipo <i>identifier</i> válido según la
 * norma QTI del IMS. Contiene métodos para comprobar si es válido, para obtener el
 * identificador, un método estático para comprobar que en un grupo de identificadores sean
 * todos distintos entre sí, y otro método estático para leer un grupo de identificadores del
 * request y comprobar que sean válidos.
 *
 * @author David Domínguez
 *
 */
public class Identificador {

 /**
  * Variable de clase con el identificador String genérico
  */
 private String identificador;
```

```
/**
 * Constructor de la clase en el que simplemente se inicializa el valor de la variable de
 * clase con el valor recibido en el parámetro.
 *
 * @param   identificador  string con el valor inicial para la variable de la clase
 */
public Identificador(String identificador) {
   super();
   this.identificador = identificador;
}

/**
 * Método "get" para obtener el valor de la variable de clase. Si no se usa el método <code>
 * esIdentificador</code> antes no se tendrá certeza de si la variable es un identificador
 * válido o no.
 *
 * @return              valor del string del identificador
 */
public String getIdentificador() {
   return identificador;
}

/**
 * Comprueba si el identificador almacenado en la variable <code>identificador</code> de
 * tipo <code>String</code> de la clase es un identificador válido según la especificación
 * del QTI. Comprueba que tenga menos de 32 caracteres, que el primer carácter sea una
 * letra o '_' (guión bajo), y que el resto de los caracteres sean letras, dígitos,'-'
 * (guión), o '_' (guión bajo). En caso de error devuelve un mensaje con la posición del
 * primer carácter erróneo encontrado.
 *
 * @return              mensaje con el resultado de la comprobación y los posibles
 *                      avisos al usuario en caso de que haya error
 * @see  utilidades.MensajeEstado
 */
public MensajeEstado esIdentificador(){

   MensajeEstado msg = new MensajeEstado();     // resultado de la comprobación

   // Comprueba que el string a comprobar tenga algún valor
   if (identificador != null && !identificador.equals("")){
      // Comprueba que la longitud sea menor de 32 caracteres por compatibilidad con la
      // versión 1 del QTI
      if (identificador.length() <= 32){
         // Comprueba que el primer caracter del identificador sea una letra o '_'
         if (identificador.charAt(0) == '_' ||
               Character.isLetter(identificador.charAt(0))){
            for (int i = 1; i < identificador.length()&& msg.isEstado(); i++){
               // Si algún carácter no es del juego de caracteres permitidos
               if (identificador.charAt(i) != '_' && identificador.charAt(i) != '-' &&
                     !Character.isLetterOrDigit(identificador.charAt(i))) {
                  int loc = i + 1;
                  msg.setEstado(false);
                  msg.setMensaje("el carácter en la posición " + loc + " no está " +
                        "permitido");
               }
            }
         } else {
            msg.setEstado(false);
            msg.setMensaje("el primer carácter debe ser una letra o '_'");
         }
      } else {
         msg.setEstado(false);
         msg.setMensaje("debe reducir el número de caracteres a 32 o menos");
      }
   } else {
      msg.setEstado(false);
      msg.setMensaje("no puede quedar vacío");
   }

   return msg;
}

/**
 * Método estático para comprobar que los identificadores de una tabla de objetos <code>
 * Identificador</code> sean todos diferentes entre sí.
 *
```

```
     * @param   tabla    tabla de objetos identificador a comprobar
     * @return           un mensajeEstado con el estado de la comprobación y los mensajes
     *                   pertinentes
     */
    public static MensajeEstado compruebaIdentificadores(Identificador[] tabla){
       MensajeEstado msg = new MensajeEstado();
       for (int i = 0; i < tabla.length; i++)
          for (int j = i + 1; j < tabla.length; j++)
             if (tabla[i].getIdentificador().equals(tabla[j].getIdentificador())){
                // Si dos identificadores son iguales
                msg.setMensaje("el identificador " + (i+1) + " es igual al identificador "
                      + (j+1));
                msg.setEstado(false);
             }

       return msg;
    }

    /**
     * Método estático para leer un grupo de identificadores recibidos como parámetros en el
     * request. Lee todos los identificadores, inicializando  el array de <code>Identificador
     * </code>, comprueba que sean correctos y modifica si es necesario el objeto <code>
     * EstadoWeb</code> del estado de la página avisando en caso de error.
     *
     * @param   identificadores      array de objetos identificador a rellenar con los
     *                          identificadores   del request
     * @param   origen             string con el nombre base de los campos donde se leen en
     *                          la web los identificadores
     * @param   estado             estadoWeb, representando el estado de la web. Se modifica
     *                          con el estado tras la comprobación de los identificadores.
     * @param   request            petición recibida del cliente donde están almacenados los
     *                          identificadores introducidos por el usuario
     * @param   pre                Texto previo para los mensajes de error
     * @see     javax.servlet.http.HttpServletRequest#getParameter
     * @see     utilidades.EstadoWeb#añadeError(String, String)
     * @see     utilidades.MensajeEstado
     */
    public static void testIdentificadores(Identificador[] identificadores, String origen,
          EstadoWeb estado, HttpServletRequest request, String pre){

       // Lee los identificadores de request con el nombre base común
       MensajeEstado msg = new MensajeEstado();
       int i;
       for (i = 0; i < identificadores.length; i++)
          identificadores[i]= new Identificador(request.getParameter(origen + i));
       // Comprueba que sean identificadores válidos
       for (i = 0; i < identificadores.length && msg.isEstado(); i++)
          msg = identificadores[i].esIdentificador();
       if (msg.isEstado()){
          // Si todos son identificadores correctos comprueba que sean todos distintos
          msg = Identificador.compruebaIdentificadores(identificadores);
          if (!msg.isEstado()) // Si hay dos identificadores repetidos
             estado.añadeError("Error: " + pre + msg.getMensaje() + "<br />", origen + "0");
       } else                  // Si encuentra un identificador no válido
          estado.añadeError("Error: " + pre + "el identificador " + i + " " +
                msg.getMensaje() +   "<br />", origen + (i-1));
    }
}
```

ListaArchivos.java

```
package utilidades;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

import org.apache.commons.io.FileUtils;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
```

```
/**
 * Clase que contiene métodos estáticos que devuelven determinadas listas del contenido de los
 * directorios.
 *
 * @author David Domínguez
 *
 */
public class ListaArchivos {

 /**
  * Devuelve un vector listando los nombres de los directorios que se incluyen en <code>
  * directorio</code>. Antes comprueba que sea realmente un directorio.
  *
  * @param   directorio     file del directorio a comprobar
  * @return              vector con <code>String</code>s con los nombres de los
  *                      subdirectorios de <code>directorio</code>
  * @see     java.io.File
  * @see     java.util.Vector
  * @see     java.util.Collections
  */
 public static Vector ListaDirectorios(File directorio){

    // Vector con los nombres de los directorios
    Vector /* String */ dirNames = new Vector();

    // Comprueba que exista el directorio
    if (directorio.exists()){
       // Comprueba que sea un directorio de verdad
       if (directorio.isDirectory()){
          // Busca los directorios del directorio actual
          File[] files = directorio.listFiles();
          Vector directorios = new Vector();
          for (int i = 0; i < files.length; i++){
             if (files[i].isDirectory())
                directorios.add(files[i]);
          }
          // Ordena el vector de directorios
          Collections.sort(directorios);
          // Mete en el vector los nombres de los directorios y lo almacena en request
          for (int i = 0; i < directorios.size(); i++)
             dirNames.add(((File)directorios.elementAt(i)).getName());
       }
    }

    return dirNames;

 }

 /**
  * Lista las imágenes incluidas en <code>directorio</code> (no en subdirectorios de éste).
  * Sólo las imágenes permitidas, cuyas extensiones (y su versión en mayúsculas), debe estar
  * incluida en el array de EXTENSIONES. Devuelve los nombres en un Vector.
  *
  * @param   directorio     file del directorio cuyas imágenes hay que listar
  * @return              vector de <code>String</code>s con los nombres de las imágenes
  * @see     File
  * @see     java.util.Vector
  * @see     java.util.Collection
  * @see     java.util.Collections
  * @see     org.apache.commons.io.FileUtils#listFiles(java.io.File, java.lang.String[],
boolean)
  */
 public static Vector ListaImagenes(File directorio){

    // Extensiones de imágenes permitidas
    String[] EXTENSIONES = new String[10];
    EXTENSIONES[0]= "jpg";
    EXTENSIONES[1]= "jpeg";
    EXTENSIONES[2]= "gif";
    EXTENSIONES[3]= "bmp";
    EXTENSIONES[4]= "png";
    EXTENSIONES[5]= "JPG";
    EXTENSIONES[6]= "JPEG";
    EXTENSIONES[7]= "GIF";
    EXTENSIONES[8]= "BMP";
```

```
    EXTENSIONES[9]= "PNG";

    // Vector con los nombres de las imágenes
    Vector /* String */ imagNames = new Vector();

    // Comprueba que el directorio existe
    if (directorio.exists()){
       // Comprueba que sea un directorio de verdad
       if (directorio.isDirectory()){
          // Busca las imágenes sólo en el directorio actual (no subdirectorios)
          Collection c = FileUtils.listFiles(directorio, EXTENSIONES, false);
          Vector imagenes = new Vector(c);
          Collections.sort(imagenes);
          // Mete en un Vector los nombres de las imágenes y lo almacena en request
          for (int i = 0; i < imagenes.size(); i++)
             imagNames.add(((File)imagenes.elementAt(i)).getName());
       }
    }

    return imagNames;
}

/**
 * Lista los ítems (en realidad archivos con extensión "xml") incluidos en <code>directorio
 * </code> (no en subdirectorios de éste). Devuelve un <code>Vector</code> con los nombres
 * de los archivos de ítems XML, su título y un objeto <code>File</code> apuntando a cada
 * ítem.
 *
 * @param   directorio  file del directorio a comprobar
 * @return          vector de objetos <code>ResumenItem</code> con los nombres y los
 *                  títulos de los archivos de los ítems xml
 * @throws  IOException si hay algún problema leyendo el contenido de los archivos XML
 * @see     java.io.File
 * @see     java.util.Vector
 * @see     java.util.Collections
 * @see     utilidades.ResumenItem
 */
public static Vector ListaItems (File directorio) throws IOException{

    // Vector con los objetos ResumenItem
    Vector /* ResumenItem */ items = new Vector();

    // Comprueba que el directorio existe y que sea un directorio, no un archivo
    if (directorio.exists() && directorio.isDirectory()){
       // Busca los ítems sólo en el directorio actual (no subdirectorios)
       File[] files = directorio.listFiles();
       for (int i = 0; i < files.length; i++){
          // Comprueba que los archivos sean archivos XML
          if (files[i].isFile()){
             String nombre = files[i].getName();    // Nombre del archivo
             String extension = nombre.substring(nombre.lastIndexOf('.'),
                   nombre.length());          // Extensión del archivo
             if (extension.equalsIgnoreCase(".xml")){
                // Si es un archivo XML se procesa usando StAX
                FileReader fr = null;
                try {
                   fr = new FileReader(files[i]);
                   XMLInputFactory xmlif = XMLInputFactory.newInstance();
                   // Parser para el archivo XML
                   XMLStreamReader xmlsr = xmlif.createXMLStreamReader(fr);
                   // Boolean indicando si se ha encontrado el título del ítem
                   boolean encontrado = false;
                   // Busca el título en el archivo XML
                   while(xmlsr.hasNext() && encontrado == false){
                      // Si el cursor está en una etiqueta de apertura
                      if(xmlsr.getEventType() == XMLStreamReader.START_ELEMENT){
                         // Si la etiqueta es la de apertura de "assesmentItem", en
                         // la que se encuentra el atributo buscado, título
                         if(xmlsr.getLocalName().equalsIgnoreCase("assessmentItem")){
                            // Recorre todos los atributos buscando el título
                            for (int j = 0; j < xmlsr.getAttributeCount() &&
                                  encontrado == false; j++) {
                               // Lee el nombre del atributo
                               String localName = xmlsr.getAttributeLocalName(j);
                               // Si el atributo es el buscado
                               if (localName.equals("title")) {
```

```
                                // Resumen del ítem con el identificador, el
                                // título y el File representándolo
                                ResumenItem ri = new ResumenItem(nombre,
                                      xmlsr.getAttributeValue(j), files[i]);
                                // Se añade el resumen del ítem al vector
                                items.add(ri);
                                encontrado = true; // Título del ítem encontrado
                             }
                          }
                       }
                    }
                    xmlsr.next();     // Avanza al siguiente evento en el ítem XML
                 }
                 xmlsr.close(); // Cierra el parser una vez ha terminado con el ítem
              } catch (FileNotFoundException e) {
                 // Si no se ha podido crear el reader para este ítem pasa al
                 // siguiente archivo sin hacer nada
              } catch (XMLStreamException e){
                 // Si hay algún problema leyendo el archivo XML no se hace nada
                 // y se pasa al siguiente archivo
              }
              fr.close();  // Cierra el Reader del ítem actual para pasar al siguiente
           }
        }
     }
     Collections.sort(items);   // Ordena los resúmenes de los ítems del Vector
  }

  return items;
}
}
```

MensajeEstado.java

```
package utilidades;

/**
 * Clase que contiene un estado y un mensaje. Se utiliza como salida de muchas comprobaciones,
 * avisando de si es correcta y de qué ha fallado en su caso.
 *
 * @author David Domínguez
 *
 */
public class MensajeEstado {

 /**
  * Mensaje a almacenar en caso de que haya error.
  */
 protected String mensaje;
 /**
  * Estado de la comprobación.
  */
 protected boolean estado;

 /**
  * Constructor que simplemente pone el estado correcto, inicializando el estado a true.
  * Si no lo fuera, deberá ponerse a falso posteriormente. El mensaje es un <code>String
  * </code> que se inicializa a la cadena vacía para después ir añadiendo avisos sin
  * problemas de si es la primera vez o no que se añade una cadena, añadiendo las sucesivas
  * con el operador '+' para cadenas.
  */
 public MensajeEstado() {
    super();
    estado = true;          // Estado correcto
    mensaje = "";           // Cadena vacía
 }

 /**
  * Método para obtener el estado de la clase.
  *
  * @return           boolean con el estado de la clase
  */
 public boolean isEstado() {
    return estado;
```

```
 }

 /**
  * Método "set" para establecer el estado de la clase.
  *
  * @param   estado      el estado al que poner la clase
  */
 public void setEstado(boolean estado) {
    this.estado = estado;
 }

 /**
  * Método "get" para obtener el mensaje de la clase.
  *
  * @return           el mensaje almacenado en la clase
  * @see     #setMensaje
  */
 public String getMensaje() {
    return mensaje;
 }

 /**
  * Método "set" para establecer el mensaje.
  *
  * @param   mensaje     string con el mensaje a añadir al mensaje actual
  * @see     #getMensaje
  */
 public void setMensaje(String mensaje) {
    this.mensaje += mensaje;
 }
}
```

ParserNeutro.java

```
package utilidades;

import java.io.StringReader;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamWriter;
import javax.xml.stream.XMLStreamConstants;

/**
 * Clase que contiene métodos estáticos para escribir con formato XML cadenas que ya tienen
 * algún formato XHTML y comprobar que sean cadenas XML válidas.
 *
 * @author   David Domínguez
 */
public class ParserNeutro {

 /**
  * Método que escribe un <code>String</code> con formato XHTML en un <code>XMLStreamWriter
  * </code>. En primer lugar elimina las entidades "&amp;nbsp;" que se puedan encontrar en
  * las instrucciones, ya que no son soportadas por la norma, sustituyéndolas por caracteres
  * de espacio en blanco normales. Luego crea un parser "StAX", <code>
  * XMLStreamReader</code>, que va leyendo los eventos XML del <code>String</code> y
  * escribiéndolos según el evento que corresponda en el <code>XMLStreamWriter</code>.
  *
  * @param   xhtml          string con el texto XHTML a enviar al Writer XML
  * @param   xsw               xmlStreamWriter con el que escribir el código XML
  * @throws  XMLStreamException   si hay algún error en la estructura del código XML leído o
  *                         al escribirlo
  * @see     javax.xml.stream.XMLStreamReader
  * @see     javax.xml.stream.XMLStreamWriter
  */
 public static void escribeInstruccionesXML (String xhtml, XMLStreamWriter xsw)
       throws XMLStreamException{

    // Limpia la cadena eliminando las entidades de espacio en blanco no separable que
    // pueden haberse introducido en el editor WYSIWYG
    final String entidad = " ";
    final String sustitutoEntidad = " ";
```

```
    // Buffer provisional para hacer los cambios sobre la cadena
    StringBuffer buffer = new StringBuffer(xhtml);
    int indice = buffer.indexOf(entidad);
    // Se buscan las entidades " " para sustituirlas por espacios en blanco habituales
    while (indice != -1){
       buffer.replace(indice, indice + entidad.length(), sustitutoEntidad);
       xhtml = buffer.toString();
       indice = xhtml.indexOf(entidad, indice);
    }

    // La cadena XHTML hay que colocarla dentro de unas etiquetas de apertura y cierre
    // iniciales ficticias que no son XHTML para poder pasarle el parser, ya que un
    // documento XML leído debe tener una etiqueta de principio y de fin que lo englobe por
    // completo. Estas etiquetas se cuidará más adelante de no se escribirlas en el
    // XMLStreamWriter.
    final String Tag = "instrucciones";
    xhtml= "<" + Tag + ">" + xhtml + "</" + Tag + ">";
    // StringReader para que el parser lea la cadena XHTML recibida
    StringReader sr = new StringReader(xhtml);
    XMLInputFactory xif = XMLInputFactory.newInstance();
    // Configura el comportamiento del parser
    xif.setProperty("javax.xml.stream.isReplacingEntityReferences", new Boolean(false));
    xif.setProperty("javax.xml.stream.isSupportingExternalEntities", new Boolean(false));
    // Parser para la cadena XHTML
    XMLStreamReader xsr = xif.createXMLStreamReader(sr);
    // Se recorren todos los eventos de la cadena XHTML utilizando el método de escritura
    // que corresponda al tipo de método
    for(int event = xsr.next(); xsr.hasNext(); event = xsr.next()){
       switch (event){
          case XMLStreamConstants.START_ELEMENT:
             // Se escriben todas etiquetas de comienzo excepto la ficticia añadida arriba
             if (!xsr.getLocalName().equals(Tag)){
                xsw.writeStartElement(xsr.getLocalName());
                  int numAtributos = xsr.getAttributeCount();
                  if (numAtributos > 0) {
                      for (int i = 0; i < numAtributos; i++){
                         xsw.writeAttribute(xsr.getAttributeLocalName(i),
                               xsr.getAttributeValue(i));
                      }
                  }
             }
             break;
          case XMLStreamConstants.CHARACTERS:
             // Se escriben los caracteres
             xsw.writeCharacters(xsr.getText());
             break;
          case XMLStreamConstants.END_ELEMENT:
             // Se escriben todas las etiquetas de cierre excepto la ficticia añadida
             // arriba
             if (!xsr.getLocalName().equals(Tag))
                xsw.writeEndElement();
             break;
          case XMLStreamConstants.SPACE:
             // Se escriben los espacios en blanco
             String space = xsr.getText();
             xsw.writeCharacters(space);
             break;
          case XMLStreamConstants.COMMENT:
             // Se escriben los comentarios
             xsw.writeComment(xsr.getText());
             break;
          case XMLStreamConstants.ENTITY_REFERENCE:
             // Se escribe la misma referencia a la entidad
             xsw.writeEntityRef(xsr.getLocalName());
       }
    }

    xsr.close();
    sr.close();
}

/**
 * Escribe todos los caracteres con el método <code>writeCharacters(String)</code>, excepto
 * los nueva línea, que los sustituye por el equivalente XHTML &lt;br /&gt; y los escribe
 * mediante el método <code>writeEmptyElement</code> en el <code>XMLStreamWriter</code>.
```

```
 * @param   texto          cadena de texto plano a escribir sustituyendo los nueva línea
 *                         por su carácter equivalente &lt;br /&gt;
 * @param   xsw               xmlStreamWriter con el que escribir el código XML
 * @throws  XMLStreamException   si hay algún error en la estructura del código XML leído o
 *                         al escribirlo
 * @see     javax.xml.stream.XMLStreamReader
 * @see     javax.xml.stream.XMLStreamWriter
 */
public static void escribeTextoXML(String texto, XMLStreamWriter xsw)
      throws XMLStreamException {

   // Comienzo del fragmento a escribir en cada iterción
   int inicio = 0;
   // Fin del fragmento a escribir en cada iteración
   int fin = texto.indexOf('\n');

   // Mientras haya caracteres nueva línea
   while (fin != -1){
      // El carácter nueva línea puede ir precedido del retorno de carro o no. Se escribe
      // el fragmento de texto sin los nueva línea ni los retorno de carro
      if (texto.charAt(fin - 1) == '\r'){
         if (inicio < fin - 1)
            xsw.writeCharacters(texto.substring(inicio, fin - 1));
      } else {
         if (inicio < fin)
            xsw.writeCharacters(texto.substring(inicio, fin));
      }

      // Después de cada fragmento se escribe el código nueva línea de XHTML
      xsw.writeEmptyElement("br");

      // Si el carácter '\n' no es el último de la cadena de texto se inicializan a un
      // nuevo valor las variables inicio y fin
      if (fin + 1 < texto.length()){
         inicio = fin + 1;
         fin = texto.indexOf('\n', inicio);
      } else{
         inicio = texto.length();
         fin = -1;
      }
   }

   // Comprueba que se hayan escrito la cadena completa por si queda un último fragmento
   // por escribir
   if (inicio < texto.length())
      xsw.writeCharacters(texto.substring(inicio, texto.length()));
}

/**
 * Comprueba un campo de instrucciones de un test o un ítem. Ya que es código XHTML, se
 * comprueba mediante un parser XML, a la espera de algún error. Ya que es sólo un fragmento
 * XHTML, antes de poder comprobar nada, se le pone una etiqueta de inicio ficticia y la de
 * fin, sólo para la comprobación.
 * @param   fragmento   string con la cadena XHTML a comprobar
 * @return           mensajeEstado con el resultado final de la comprobación
 */
public static MensajeEstado compruebaInstruccionesXHTML (String fragmento){
   MensajeEstado msg = new MensajeEstado();
   // La cadena XHTML hay que colocarla dentro de unas etiquetas de apertura y cierre
   // iniciales ficticias que no son XHTML para poder pasarle el parser, ya que un
   // documento XML leído debe tener una etiqueta de principio y de fin que lo englobe por
   // completo. Estas etiquetas se cuidará más adelante de no se escribirlas en el
   // XMLStreamWriter.
   final String Tag = "instrucciones";
   String fragmentoXML = "<" + Tag + ">" + fragmento + "</" + Tag + ">";
   // StringReader para que el parser lea la cadena XHTML recibida
   StringReader sr = new StringReader(fragmentoXML);
   XMLInputFactory xif = XMLInputFactory.newInstance();
   // Configura el comportamiento del parser
   xif.setProperty("javax.xml.stream.isReplacingEntityReferences", new Boolean(false));
   xif.setProperty("javax.xml.stream.isSupportingExternalEntities", new Boolean(false));
   // Parser para la cadena XHTML
   XMLStreamReader xsr;
   try {
      xsr = xif.createXMLStreamReader(sr);
      // Se recorren todos los eventos de la cadena XHTML
```

```
        for(xsr.next(); xsr.hasNext(); xsr.next()){
           // Vacío. No se hace nada con los datos leídos. Simplemente se recorre la cadena
           // XHTML para comprobar si es correcta.
        }
    } catch (XMLStreamException e) {
        msg.setEstado(false);
        msg.setMensaje("Las instrucciones tienen un código incorrecto. Debe revisarlas." +
              "<br />");
    }

    return msg;
 }
}
```

ResumenItem.java

```
package utilidades;

import java.io.File;

/**
 * Clase que representa un resumen de los ítems (archivos XML independientes que representan al
 * ítem). El resumen del ítem contiene el nombre del ítem (nombre del archivo XML) en un <code>
 * String</code>, el título del ítem en otro <code>String</code>, y el objeto <code>File</code>
 * que representa al archivo XML del ítem. Contiene un constructor que inicializa todos sus
 * valores, métodos "getters" para obtenerlos, e implementa la interfaz <code>Comparable</code>
 * para permitir que se ordene una colección de objetos <code>ResumenItem</code>.
 *
 * @author   David Domínguez
 * @see      Comparable
 */
public class ResumenItem implements Comparable {

 /**
  * Nombre del ítem (nombre del archivo XML).
  */
 private String nombre;
 /**
  * Título del ítem.
  */
 private String titulo;
 /**
  * File al archivo XML del ítem.
  */
 private File archivo;

 /**
  * Constructor de la clase que simplemente da a las variables de clase los valores
  * recibidos como parámetros.
  *
  * @param   nombre      string con el nombre del ítem
  * @param   titulo      string con el título del ítem
  * @param   archivo     file representando al archivo XML del ítem
  */
 public ResumenItem(String nombre, String titulo, File archivo) {
    super();
    this.nombre = nombre;
    this.titulo = titulo;
    this.archivo = archivo;
 }

 /**
  * Método "get" que devuelve el <code>String</code> con el nombre del ítem.
  *
  * @return        string con el nombre del ítem
  */
 public String getNombre() {
    return nombre;
 }

 /**
  * Método "get" que devuelve el <code>String</code> con el título del ítem.
  *
  * @return        string con el título del ítem
```

```
 */
public String getTitulo() {
   return titulo;
}

/**
 * Método "get" que devuelve el objeto <code>File</code> que representa al archivo XML del
 * ítem.
 *
 * @return        file con el archivo XML del ítem
 */
public File getArchivo() {
   return archivo;
}

/**
 * Método definido por la interfaz <code>Comparable</code> para permitir la comparación
 * entre los objetos que implementen la interfaz. Este método utiliza el método del mismo
 * nombre implementado por la clase <code>String</code> para comparar los objetos <code>
 * ResumenItem</code> según sus nombres. De este modo se permite ordenar de forma correcta
 * colecciones de objetos <code>ResumenItem</code> según el nombre del ítem. Para su
 * correcto funcionamiento, el método comprueba primero que el objeto con el que se compara
 * sea también de tipo <code>ResumenItem</code>.
 *
 * @param   o              objeto con el que comparar este ResumenItem
 * @return                 entero resultado de la comparación de los dos nombres de
 *                         los resúmenes de los ítems
 * @throws  ClassCastException   si el objeto con el que comparar no es de este mismo tipo
 */
public int compareTo(Object o) {

   // Comprueba que el objeto "o" sea una instancia de esta misma clase
   if (!(o instanceof ResumenItem))
      throw new ClassCastException ("Se deben comparar dos objetos \"ResumenItem\"");
   else {
      ResumenItem ri = (ResumenItem) o;
      return nombre.compareTo(ri.nombre);
   }
}
}
```

UrlUtils.java

```
package utilidades;

/**
 * Clase con métodos para codificar URLs. <br />
 * ------------------------------------------------------------------------------
 * URL Utils - UrlUtils.java
 * Author: C. Enrique Ortiz
 * Copyright (c) 2004-2005 C. Enrique Ortiz
 *
 * This is free software; you can redistribute it and/or modify it under
 * the terms of the GNU Lesser General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option) any
 * later version.
 *
 * Usage & redistributions of source code must retain the above copyright notice.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should get a copy of the GNU Lesser General Public License from
 * the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
 * Boston, MA  02111-1307  USA
 * ------------------------------------------------------------------------------<br />
 *
 * Modificado por David Domínguez. <br />
 *
 * La clase contiene un método estático para codificar todos los caracteres, excepto los de
 * tipo <code>mark</code> y los permitidos, de un <code>String</code> que representa una URL.
Se
```

```
 * ha añadido un método similar pero que no codifica tampoco los caracteres reservados.
 *
 * @author   C. Enrique Ortiz
 * @author   David Domínguez
 *
 */
public class UrlUtils {

    /**
     * Caracteres marcas no reservadas
     */
    private static String mark = "-_.!~*'()\"";
    /**
     * Caracteres reservados
     */
    private static String reservados = ";/?:@&=+$,";

    /**
     * Converts Hex digit to a UTF-8 "Hex" character.
     *
     * @param digitValue digit to convert to Hex
     * @return the converted Hex digit
     */
    static private char toHexChar(int digitValue) {
        if (digitValue < 10)
            // Convert value 0-9 to char 0-9 hex char
            return (char)('0' + digitValue);
        else
            // Convert value 10-15 to A-F hex char
            return (char)('A' + (digitValue - 10));
    }

    /**
     * Encodes a URL - This method assumes UTF-8
     * Codifica todos los caracteres especiales, incluidos los caracteres reservados. Se
     * utiliza para codificar nombres individuales de archivos o directorios de una URL.
     * Método original de la clase.
     *
     * @param url URL to encode
     * @return the encoded URL
     */
    static public String encodeURLReserved(String url) {
        StringBuffer encodedUrl = new StringBuffer(); // Encoded URL
        int len = url.length();
        // Encode each URL character
        for(int i = 0; i < len; i++) {
            char c = url.charAt(i); // Get next character
            if ((c >= '0' && c <= '9') ||
                (c >= 'a' && c <= 'z') ||
                (c >= 'A' && c <= 'Z'))
                // Alphanumeric characters require no encoding, append as is
                encodedUrl.append(c);
            else {
                int imark = mark.indexOf(c);
                if (imark >=0) {
                    // Las marcas de puntuación no reservadas, los símbolos y los
                  // caracteres reservados no quieren codificación, se añaden
                  // tal y como son
                    encodedUrl.append(c);
                } else {
                    // Encode all other characters to Hex, using the format "%XX",
                    //  where XX are the hex digits
                    encodedUrl.append('%'); // Add % character
                    // Encode the character's high-order nibble to Hex
                    encodedUrl.append(toHexChar((c & 0xF0) >> 4));
                    // Encode the character's low-order nibble to Hex
                    encodedUrl.append(toHexChar (c & 0x0F));
                }
            }
        }
        return encodedUrl.toString(); // Return encoded URL
    }

    /**
     * Encodes a URL - This method assumes UTF-8
     * Codifica todos los caracteres excepto los permitidos, los reservados y los <code>mark
```

```
     * </code>. Se utiliza para codificar URLs totales o parciales para que no codifique los
     * caracteres reservados, como los separadores, que se supone que son separadores.
     * Método añadido a la clase.
     *
     * @param   url   string con la URL a codificar
     * @return        string con la URL codificada
     */
    static public String encodeURL(String url) {
        StringBuffer encodedUrl = new StringBuffer(); // Encoded URL
        int len = url.length();
        // Encode each URL character
        for(int i = 0; i < len; i++) {
            char c = url.charAt(i); // Get next character
            if ((c >= '0' && c <= '9') ||
                (c >= 'a' && c <= 'z') ||
                (c >= 'A' && c <= 'Z'))
                // Alphanumeric characters require no encoding, append as is
                encodedUrl.append(c);
            else {
             String caracteres = mark + reservados;
                int imark = caracteres.indexOf(c);
                if (imark >=0) {
                    // Las marcas de puntuación no reservadas, los símbolos y los
                   // caracteres reservados no quieren codificación, se añaden
                   // tal y como son
                     encodedUrl.append(c);
                } else {
                    // Encode all other characters to Hex, using the format "%XX",
                    //  where XX are the hex digits
                    encodedUrl.append('%'); // Add % character
                    // Encode the character's high-order nibble to Hex
                    encodedUrl.append(toHexChar((c & 0xF0) >> 4));
                    // Encode the character's low-order nibble to Hex
                    encodedUrl.append(toHexChar (c & 0x0F));
                }
            }
        }
        return encodedUrl.toString(); // Return encoded URL
    }
}
```

2.3 Paquete `utilidades.etiquetas.util`

AdminMailTag.java

```
package utilidades.etiquetas.util;

import javax.servlet.jsp.tagext.SimpleTagSupport;
import javax.servlet.jsp.*;

import java.io.IOException;

/**
 * Etiqueta de página JSP para imprimir en pantalla la dirección de correo electrónico del
 * administrador de la Aplicación. La clase extiende a la clase <code>SimpleTagSupport</code>
 * que representa a una Etiqueta Personalizada de JSP Simple. La etiqueta se encarga de
 * devolver por la salida de la página JSP la dirección de correo electrónico de contacto del
 * administrador, parámetro definido en el descriptor de despliegue de la Aplicación,
 * "web.xml". El proceso se realiza en el método sobreescrito <code>doTag</code>, que se
 * ejecuta al llamar a la etiqueta en una página JSP.
 *
 * @author David Domínguez
 *
 */
public class AdminMailTag extends SimpleTagSupport {

 /**
  * Imprime por pantalla la dirección de correo electrónico del administrador de la
  * Aplicación. Recoge el parámetro definido en el descriptor de despliegue de la Aplicación
  * leído del contexto, y lo imprime por la salida estándar de la página JSP desde la que se
  * llame a la etiqueta.
  *
  * @throws  IOException    si hay algún problema al intentar escribir en la JSP
```

```
   * @see  javax.servlet.jsp.tagext.SimpleTag#doTag()
   */
 public void doTag() throws IOException {
    // Contexto JSP para obtener el "Writer" para escribir en la página JSP
    JspContext jspC = getJspContext();
    JspWriter out = jspC.getOut();
    // Contexto de la página para acceder al contexto del Servlet y obtener el parámetro
    // requerido
    PageContext pc = (PageContext)jspC;
    String mail = pc.getServletContext().getInitParameter("admin email");
    // Se imprime en pantalla el parámetro recogido
    out.print(mail);
 }
}
```

RequestedURITag.java

```
package utilidades.etiquetas.util;

import java.io.IOException;
import java.net.URLDecoder;

import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

/**
 * Etiqueta de página JSP para imprimir en pantalla la dirección de la página Web que causó el
 * error. La clase extiende a la clase <code>SimpleTagSupport</code> que representa a una
 * Etiqueta Personalizada de JSP Simple. La etiqueta se encarga de devolver por la salida de la
 * página JSP, que debe ser una página de error, la URI que provocó el error. El proceso se
 * realiza en el método sobreescrito <code>doTag</code>, que se ejecuta al llamar a la etiqueta
 * en una página JSP.
 *
 * @author David Domínguez
 *
 */
public class RequestedURITag extends SimpleTagSupport {

 /**
  * Imprime por pantalla la dirección de la Web que provocó el error. Recoge el parámetro
  * contenido en el ámbito de request de la página JSP desde la que se llama  a la etiqueta,
  * que debe ser una página de error, indicando la uri que provocó ese error, y lo imprime
  * por la salida estándar de la página JSP desde la que se llame a la etiqueta.
  *
  * @throws  IOException    si hay algún problema al intentar escribir en la JSP
  * @see  javax.servlet.jsp.tagext.SimpleTag#doTag()
  */
 public void doTag() throws IOException{

    // Contexto de la página JSP para obtener el "Writer" para escribir en la página
    JspContext jspC = getJspContext();
    JspWriter out = jspC.getOut();
    // Se obtiene del ámbito de request el valor de la uri que causó el error y se imprime
    // en la página JSP
    String uri = URLDecoder.decode((String)jspC.
          getAttribute("javax.servlet.error.request_uri", PageContext.REQUEST_SCOPE),
          "UTF-8");
    out.print(uri);
 }
}
```

2.4 Paquete `utilidades.etiquetas.util.logging`

AplicacionLogger.java

```
package utilidades.etiquetas.util.logging;

import javax.servlet.*;
```

```
import java.util.logging.*;

/**
 * Arranca y detiene el <code>Logger</code> de la Aplicación Web cuando ésta se arranca y se
 * detiene. La clase implementa la interfaz <code>ServletContextListener</code> y ejecutará
 * sus métodos siempre que el contexto de la Aplicación Web se cree o se destruya (normalmente
 * al arrancar o parar la Aplicación). Se encarga de preparar el objeto que se ocupa de las
 * acciones relacionadas con el registro de los errores que se produzcan en la Aplicación, es
 * decir, el "logging".
 * Contiene una variable de clase <code>String</code> con el nombre del logger de la Aplicación
 * para acceder a él desde otros puntos de la Aplicación por su nombre, y otra de tipo <code>
 * Level</code> donde se define el nivel mínimo de importancia de registro de los errores de la
 * Aplicación.
 *
 * @author   David Domínguez
 * @see      ServletContextListener
 *
 */
public class AplicacionLogger implements ServletContextListener {

 /**
  * Nombre del logger de la Aplicación para acceder a él por su nombre en cualquier punto de
  * la Aplicación que se requiera
  */
 static String nombreLogger = "Aplicacion.QTI";
 /**
  * Nivel mínimo de importancia de los errores que serán guardados por el logger
  */
 static Level nivelLogger = Level.WARNING;

 /**
  * Método que arranca el <code>Logger</code> de la Aplicación Web. El método se ejecutará
  * siempre que se cree el contexto de la Aplicación (normalmente al arrancar ésta). Se
  * encarga de crear un objeto <code>Logger</code> para crear un registro de los errores que
  * ocurran en la Aplicación, estableciendo su nivel mínimo de registro. El manejador del
  * Logger se deberá añadir siempre que se vaya a realizar algún registro con el Logger.
  *
  * @param   sce      sevletContextEvent con los datos del contexto que se ha inicializado
  * @see
javax.servlet.ServletContextListener#contextInitialized(javax.servlet.ServletContextEvent)
  * @see  java.util.logging.Logger
  */
 public void contextInitialized(ServletContextEvent sce) {

    // Crea una instancia de un Logger
    Logger logger = Logger.getLogger(nombreLogger);
    // Establece el nivel mínimo para almacenar los informes
    logger.setLevel(nivelLogger);

    // El manejador del loggger se añade en la ejecución de la etiqueta en caso de que
    // ocurra algún error y se vaya a registrar
 }

 /**
  * Método que detiene el <code>Logger</code> de la Aplicación Web. El método se ejecutará
  * siempre que se destruya el contexto de la Aplicación (normalmente al detenerla). Se
  * encarga de cerrar todos los <code>Handler</code>s y eliminarlos del <code>Logger</code>.
  *
  * @param   sce      sevletContextEvent con los datos del contexto que se ha inicializado
  * @see
javax.servlet.ServletContextListener#contextDestroyed(javax.servlet.ServletContextEvent)
  * @see  java.util.Handler
  */
 public void contextDestroyed(ServletContextEvent sce) {
    // Se obtiene el Logger ya creado por el nombre
    Logger logger = Logger.getLogger(nombreLogger);
    // Array con los Handlers añadidos al logger
    Handler[] handlers = logger.getHandlers();
    // Cada handler se vacía, se cierra y se elimina del logger
    for (int i = 0; i < handlers.length; i++){
       handlers[i].flush();
       handlers[i].close();
       logger.removeHandler(handlers[i]);
    }
 }
}
```

ErrorManagerPersonalizado.java

```
package utilidades.etiquetas.util.logging;

import java.util.logging.ErrorManager;

/**
 * Manejador de los errores que se produzcan en el <code>Logger</code> de la Aplicación Web. La
 * clase se encarga de informar de que se ha producido algún error en el proceso del registro
 * de errores. Sobreescribe el método <code>error</code> que será el llamado en caso de error
 * al intentar registrar algo con el logger de la Aplicación.
 *
 * @author   David Domínguez
 * @see      java.util.logging.ErrorManager
 */
public class ErrorManagerPersonalizado extends ErrorManager {

 /**
  * Constructor de la clase que simplemente llama al constructor de la superclase.
  */
 public ErrorManagerPersonalizado() {
    super();
 }


 /**
  * Informa de que se ha producido un error al registrar algún evento de la Aplicación Web.
  * El método será llamado cuando se produzca algún error al realizar algún registro. Este
  * método mira qué tipo de error se ha producido y genera en consecuencia un mensaje a
  * medida, añadiendo también el mensaje que informa del error y el mensaje de la excepción,
  * y sacándolo todo finalmente por la salida estándar de errores.
  *
  * @param   msg      string con el mensaje informando del error ocurrido en el logger
  * @param   ex    excepion con la excepción que ha provocado el error del logger
  * @param   code  entero con el código de error que indica el tipo de error ocurrido
  * @see java.util.logging.ErrorManager#error(java.lang.String, java.lang.Exception, int)
  */
 public void error(String msg, Exception ex, int code) {
    // Mensaje de salida para informar del error producido
    String cadena = "";
    // Se crea un mensaje en función del tipo de error que se haya producido
    switch (code){
       case ErrorManager.CLOSE_FAILURE:
          cadena = "Error al cerrar el flujo de salida del manejador del logger de la " +
                "aplicación QTI\n";
          break;
       case ErrorManager.FLUSH_FAILURE:
          cadena = "Error al realizar el \"flush\" sobre el flujo de salida del " +
                "manejador del logger de la aplicación QTI\n";
          break;
       case ErrorManager.FORMAT_FAILURE:
          cadena = "Error al dar formato en el manejador del logger de la aplicación " +
                "QTI\n";
          break;
       case ErrorManager.GENERIC_FAILURE:
          cadena = "Error genérico en el manejador del logger de la aplicación QTI\n";
          break;
       case ErrorManager.OPEN_FAILURE:
          cadena = "Error al abrir el flujo de salidad del manejador del logger de la " +
                "aplicación QTI\n";
          break;
       case ErrorManager.WRITE_FAILURE:
          cadena = "Error al escribir en el flujo de salida del manejador del logger " +
                "de la aplicación QTI\n";
          break;
    }

    // Se añade al mensaje de salida el mensaje recibido y el mensaje de la excepción
    cadena += msg + ex.getMessage();
    // Se saca por la salida estándar de errores el mensaje creado
    System.err.println(cadena);
 }
}
```

FormatterPersonalizado.java

```
package utilidades.etiquetas.util.logging;

import java.io.StringWriter;
import java.io.PrintWriter;
import java.text.DateFormatSymbols;
import java.text.SimpleDateFormat;
import java.util.logging.Formatter;
import java.util.logging.LogRecord;
import java.util.Date;

/**
 * <code>Formater</code> personalizado para realizar los registros de la Aplicación Web con el
 * formato personalizado. La clase extiende a la clase abstracta <code>Formatter</code>,
 * sobreescribiendo sólo su método <code>format</code>, aplicando un formato personalizado al
 * registro de los errores de la Aplicación.
 *
 * @author   David Domínguez
 * @see      java.util.logging.Formatter
 */
public class FormatterPersonalizado extends Formatter {

 /**
  * Aplica un formato personalizado a la información sobre el error a registrar recibida en
  * el <code>LogRecord</code>. Pone inicialmente el nivel de importancia del registro con la
  * fecha y la hora entre corchetes aplicándole un formato específico (con los nombres
  * cortos de los días de la semana y los meses en castellano); luego el mensaje del
  * registro recibido; en la línea siguiente la uri que ha disparado el registro en el
  * servidor; y finalmente la traza del error si ha sido una excepción.
  *
  * @param   logRecord   logRecord con la información a formatear para registrar
  * @return           string con el formato final de la información a registrar
  * @see java.util.logging.Formatter#format(java.util.logging.LogRecord)
  */
 public String format(LogRecord logRecord) {
    // Objeto con los símbolos a utilizar para crear una fecha
    DateFormatSymbols dfs = new DateFormatSymbols();
    // Se modifica el objeto de símbolos de fecha poniendo los nombres cortos a utilizar
    // en castellano
    LoggerTag.setNombresCortosCastellano(dfs);
    // Objeto para formatear una fecha según el formato indicado y con los símbolos que
    // indica el objeto de símbolos de fecha
    SimpleDateFormat sdf = new SimpleDateFormat("EEE d MMM HH:mm:ss zzz yyyy", dfs);
    // Objeto con la fecha indicada por el logRecord
    Date date = new Date(logRecord.getMillis());
    // Nombre del nivel de importancia del registro a escribir
    String level = logRecord.getLevel().getName();
    // Primera línea de la cadena con la información formateada con el nivel y la fecha
    // según el formato indicado
    String cadena = "[" + level + " " + sdf.format(date) + "]\n";
    // Se añade la segunda línea con el mensaje informando del registro recibido
    cadena += logRecord.getMessage() + "\n";
    // Se obtiene la dirección de la uri que disparó el registro del parámetro en la
    // posición #0 del array de parámetros
    String origen = logRecord.getParameters()[0].toString();
    if (origen != null)
       cadena += "URI que provocó el error: " + origen;
    // Se añade si el registro implica una excepción la traza de ésta
    Throwable thrown = logRecord.getThrown();
    if (thrown != null) {
       StringWriter sw = new StringWriter();
       thrown.printStackTrace(new PrintWriter(sw));
       cadena += "\n" + sw.toString();
    }
    cadena +=  "\n\n";

    return cadena;
 }
}
```

LoggerTag.java

```
package utilidades.etiquetas.util.logging;
```

```
import java.io.File;
import java.io.IOException;
import java.text.DateFormatSymbols;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.logging.*;

import javax.servlet.jsp.tagext.SimpleTagSupport;
import javax.servlet.jsp.*;

/**
 * Etiqueta de página JSP para realizar el registro de algún error o evento que se requiera
 * registrar de la Aplicación Web. La clase extiende a la clase <code>SimpleTagSupport</code>
 * que representa a una Etiqueta Personalizada de JSP Simple. La etiqueta se encarga de
 * registrar algún error o información en el <code>Logger</code> que se creó al iniciar la
 * Aplicación Web. Tiene dos variables de clase que son los dos atributos que se le pasan a la
 * etiqueta, con sus dos métodos "setters" correspondientes: el primero de ellos es un <code>
 * String</code> con el nivel de importancia del registro a realizar, y el segundo el mensaje
 * del registro. Todo el proceso de registro se realiza en el método sobreescrito <code>
 * doTag</code>, que se ejecuta al llamar a la etiqueta en una página JSP.
 *
 * Contiene también un método estático para poner en castellano los nombres cortos de los días
 * de la semana y de los meses del año de un objeto de formato de los símbolos de fechas,
 * <code>DateFormatSymbols</code>.
 *
 * @author   David Domínguez
 * @see      javax.servlet.jsp.tagext.SimpleTagSupport
 */
public class LoggerTag extends SimpleTagSupport {

 /**
  * String con el nombre del nivel de importancia del registro.
  */
 private String nivel;
 /**
  * String con el mensaje a registrar.
  */
 private String mensaje;


 /**
  * Método "set" que establece el mensaje a registrar con el <code>Logger</code>.
  *
  * @param   mensaje     string con el mensaje a registrar
  */
 public void setMensaje(String mensaje) {
    this.mensaje = mensaje;
 }


 /**
  * Método "set" que establece el nombre del nivel de importancia del registro a realizar.
  *
  * @param   nivel    string con el nombre del nivel de importancia del registro
  */
 public void setNivel(String nivel) {
    this.nivel = nivel;
 }


 /**
  * Realiza el registro de la información requerida en el <code>Logger</code> de la
  * Aplicación. El método se ejecuta siempre que se llama a la etiqueta personalizada
  * asociada a esta clase. Se encarga de asociar y configurar un manejador con el <code>
  * Logger</code> de la Aplicación para registrar la información recibida en la página desde
  * la que se llama a esta etiqueta, y la indicada en los atributos, mediante ese manejador,
  * o <code>Handler</code>.
  *
  * @see  javax.servlet.jsp.tagext.SimpleTag#doTag()
  * @see  java.util.logging.Logger
  * @see  java.util.logging.Handler
  * @see     java.text.DateFormatSymbols
  * @see     java.text.SimpleDateFormat
  */
 public void doTag(){
```

```
// Directorio en el que almacenar los registros
final String dirRegistro = "/WEB-INF/log/";
// Manejador de mensajes; los recibe del logger y los escribe en un archivo
FileHandler fh = null;
// Contexto de la página para acceder a la dirección real del directorio de registro
PageContext pc = (PageContext)getJspContext();
// Nombre del directorio donde almacenar los registros
String root = pc.getServletContext().getRealPath(dirRegistro);
File dir = new File(root);
// Si el directorio no existe intenta crearlo
boolean dirExiste = dir.exists();
if (!dirExiste)
   dirExiste = dir.mkdirs();

//Si el directorio existe
if (dirExiste){
   // Objeto con los símbolos a utilizar para crear una fecha
   DateFormatSymbols dfs = new DateFormatSymbols();
   // Se modifica el objeto de símbolos de fecha poniendo los nombres cortos a
   // utilizar en castellano
   setNombresCortosCastellano(dfs);
   // Objeto para formatear una fecha según el formato indicado y con los símbolos que
   // indica el objeto de símbolos de fecha
   SimpleDateFormat sdf = new SimpleDateFormat("EEE, d-MMM-yyyy", dfs);
   // Objeto con la fecha actual
   Date date = new Date();
   // Nombre del archivo de registro con la fecha actual
   String nombre = sdf.format(date) + ".log";
   try {
      // Se recupera la instancia del logger creada por el nombre
      Logger logger = Logger.getLogger(AplicacionLogger.nombreLogger);
      // File del archivo de registro
      File logTxt = new File(root + "\\" + nombre);
      // Se recupera el array de Handlers asociados al logger
      Handler[] handlers = logger.getHandlers();
      // Si el archivo no existe o si el logger no tiene handlers asociados se crea
      // el manejador apropiado para el logger
      if (!logTxt.exists() || handlers.length == 0 ){
         // Nuevo manejador de registro en fichero en el fichero que corresponde, e
         // indicando que los nuevos registros se añaden al final
         fh = new FileHandler(root + "\\" + nombre, true);
         // Se establece el administrador de errores del handler que capturará sus
         // excepciones y actúa en consecuencia
         fh.setErrorManager(new ErrorManagerPersonalizado());
         // Establece el objeto Formmatter que aplica el formato a la información a
         // registrar en el archivo
         fh.setFormatter(new FormatterPersonalizado());
         // Vacía y cierra los manejadores que están asociados al logger y los
         // disocia de él
         for (int i = 0; i < handlers.length; i++){
            handlers[i].flush();
            handlers[i].close();
            logger.removeHandler(handlers[i]);
         }
         // Añade el nuevo handler de archivo creado al logger
         logger.addHandler(fh);
      }
      // Se crea un objeto registro con la información a almacenar de los atributos
      LogRecord record = new LogRecord(Level.parse(nivel), mensaje);
      // Se almacena en el objeto registro el objeto Throwable contenido en la página
      // que llama a la etiqueta si lo hay, almacenado en "request" en caso de error
      Throwable thrown = (Throwable)pc.getAttribute("javax.servlet.error.exception",
            PageContext.REQUEST_SCOPE);
      if (thrown != null)
         record.setThrown(thrown);
      // Más parámetros a añadir al objeto registro en un array
      Object[] params = new Object[1];
      // Se añade sólo la uri que ha causado el error que está almacenada en "request"
      params[0] = (String)pc.getAttribute("javax.servlet.error.request_uri",
            PageContext.REQUEST_SCOPE);
      // Se añade el array de parámetros al objeto registro
      record.setParameters(params);
      // Se envía la información a registrar al logger
      logger.log(record);
   } catch (IOException e){
      // Si no se ha podido cargar el logger se avisa por la salida estándar de
```

```
            // errores
            System.err.println("No se pudo cargar el logger: " +e.getMessage());
        }
    }
    else {
        // si no se ha podido crear el directorio de los registros se avisa por la salida
        // estándar de errores
        System.err.println("No se ha podido crear el directorio de registro: " +
                dirRegistro);
    }
}


/**
 * Personaliza un objeto <code>DateFormatSymbols</code> poniendo algunos nombres cortos en
 * castellano. El método estático se encarga de modificar los nombres cortos de los días de
 * la semana y los de los meses del año de un objeto de formato de los símbolos de las
 * fechas, <code>DateFormatSymbols</code>, poniéndolos en castellano.
 *
 * @param   dfs      dateFormatSymbols a modificar
 */
public static void setNombresCortosCastellano(DateFormatSymbols dfs){
    // Array con los nombres cortos de los meses del año indexados según las constantes de
    // la clase Calendar
    String[] mesesCortos = new String[12];
    mesesCortos[Calendar.JANUARY] = "Ene";
    mesesCortos[Calendar.FEBRUARY] = "Feb";
    mesesCortos[Calendar.MARCH] = "Mar";
    mesesCortos[Calendar.APRIL] = "Abr";
    mesesCortos[Calendar.MAY] = "May";
    mesesCortos[Calendar.JUNE] = "Jun";
    mesesCortos[Calendar.JULY] = "Jul";
    mesesCortos[Calendar.AUGUST] = "Ago";
    mesesCortos[Calendar.SEPTEMBER] = "Sep";
    mesesCortos[Calendar.OCTOBER] = "Oct";
    mesesCortos[Calendar.NOVEMBER] = "Nov";
    mesesCortos[Calendar.DECEMBER] = "dic";

    // Array con los nombres cortos de los días de la semana indexados según las constantes
    // de la clase Calendar (de 8 elementos pues las constantes van del 1 al 7, ninguna
    // vale 0)
    String[] diasCortos = new String[8];
    diasCortos[Calendar.MONDAY] = "Lun";
    diasCortos[Calendar.TUESDAY] = "Mar";
    diasCortos[Calendar.WEDNESDAY] = "Mier";
    diasCortos[Calendar.THURSDAY] = "Jue";
    diasCortos[Calendar.FRIDAY] = "Vie";
    diasCortos[Calendar.SATURDAY] = "Sab";
    diasCortos[Calendar.SUNDAY] = "Dom";

    // Establece los nuevos días cortos de los días de la semana y de los meses del año
    dfs.setShortWeekdays(diasCortos);
    dfs.setShortMonths(mesesCortos);
}
}
```

2.5 Paquete `xml.items`

AssessmentItem.java

```
package xml.items;

import utilidades.MensajeEstado;

import java.io.IOException;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import javax.servlet.ServletContext;

/**
 * Representación de un ítem XML de la norma QTI del IMS general. Es una clase abstracta,
 * superclase de todas las clases contenedoras de los ítems XML. Esta clase representa un ítem
```

```
 * XML general, y como tal, contiene los tres campos que tienen todos los ítems: <code>
 * identificador</code>, <code>título</code>, e <code>instrucciones</code> del ítem. Contiene
 * un constructor que inicializa esos tres campos. También un método abstracto para escribir el
 * ítem XML en disco y un método para escribir la cabecera común de todos los ítems y otro para
 * el pie.
 *
 * @author David Domínguez
 *
 */
public abstract class AssessmentItem {


 /**
  * El identificador del ítem, que será también el nombre del archivo.
  */
 protected String identificador;
 /**
  * Título del ítem.
  */
 protected String titulo;
 /**
  * Instrucciones para responder el ítem.
  */
 protected String instrucciones;

 /**
  * Constructor que le da valor a las tres variables de clase. Simplemente las inicializa a
  * esos valores.
  *
  * @param   identificador     identificador del ítem (y el nombre del archivo)
  * @param   titulo            título del ítem
  * @param   instrucciones     instrucciones para el ítem
  */
 public AssessmentItem (String identificador, String titulo, String instrucciones) {

    this.identificador = identificador;
    this.titulo = titulo;
    this.instrucciones = instrucciones;
 }

 /**
  * Método "get" para obtener el identificador actual de ítem. Para comprobar si el nombre
  * del fichero ya existe antes de intentar crearlo.
  *
  * @return              string con el identificador actual del ítem
  * @see     #setIdentificador
  */
 public String getIdentificador() {
    return identificador;
 }

 /**
  * Método "set" para establecer el identificador del ítem. Si el que tiene no es un nombre
  * de archivo válido con este método se le pone un nombre nuevo.
  *
  * @param   identificador  string con el nuevo identificador
  * @see     #getIdentificador
  */
 public void setIdentificador(String identificador) {
    this.identificador = identificador;
 }

 /**
  * Escribe en disco el ítem XML al que representa esta clase. Debe ser sobreescrito por
  * todos los ítems distintos.El método se encarga de construir un ítem XML de acuerdo con
  * la norma QTI con los campos que tiene almacenados y en la dirección que recibe como
  * parámetro. Devuelve un <code>MensajeEstado</code> informando del resultado del proceso.
  *
  * @param   path     string con la dirección absoluta en la que crear el archivo XML
  * @param   sc       servletContext para acceder a los parámetros iniciales de "web.xml"
  * @param   req         string con la url base de la Aplicación Web requerida
  * @return           un mensajeEstado informando del estado final del proceso
  * @throws  IOException si hay algún problema al intentar escribir el archivo en disco
  */
 public abstract MensajeEstado creaItemXML(String path, ServletContext sc, String req)
       throws IOException;
```

```
/**
 * Escribe en un <code>writer</code> específico para escribir archivos XML la cabecera
 * común a todos los tipos de ítem XML.
 *
 * @param   xsw              XMLStreamWriter, un writer específico para escribir
 *                           contenido XML
 * @param   sc               servletContext para leer los parámetros del nombre de la
 *                           herramienta y la versión de "web.xml"
 * @throws  XMLStreamException   si hay algún problema con los datos XML a escribir
 * @see     XMLStreamWriter
 */
protected void writeCabecera(XMLStreamWriter xsw, ServletContext sc)
      throws XMLStreamException{
   xsw.writeStartDocument("ISO-8859-1","1.0");
   // Elemento raíz de todos los ítems XML
   xsw.writeStartElement("assessmentItem");
   xsw.writeDefaultNamespace("http://www.imsglobal.org/xsd/imsqti_v2p1");
   xsw.writeNamespace("xsi", "http://www.w3.org/2001/XMLSchema-instance");
   xsw.setPrefix("xsi", "http://www.w3.org/2001/XMLSchema-instance");
   xsw.writeAttribute("http://www.w3.org/2001/XMLSchema-instance", "schemaLocation",
         "http://www.imsglobal.org/xsd/imsqti_v2p1 imsqti_v2p1.xsd");
   // La implementación no admite ítems adaptativos
   xsw.writeAttribute("adaptive", "false");
   // La implementación no admite ítems dependientes del tiempo
   xsw.writeAttribute("timeDependent", "false");
   // Nombre y versión de esta herramienta de creación de ítems XML
   xsw.writeAttribute("toolName", sc.getInitParameter("nombre herramienta"));
   xsw.writeAttribute("toolVersion", sc.getInitParameter("versión herramienta"));
}

/**
 * Escribe en un <code>writer</code> específico para escribir archivos XML el pie común a
 * todos los tipos de ítem XML. Al finalizar llama a <code>flush</code> del <code>writer
 * </code> y finalmente lo cierra llamando a <code>close</code>.
 *
 * @param   xsw              XMLStreamWriter, un writer específico para escribir
 *                           contenido XML
 * @param   sc               servletContext para leer la uri de la plantilla para el
 *                           procesado de la respuesta
 * @param   req                 strng con la URL base del proyecto
 * @throws  XMLStreamException   si hay algún problema con los datos XML a escribir
 * @see     XMLStreamWriter
 */
protected void writePie(XMLStreamWriter xsw, ServletContext sc, String req)
      throws XMLStreamException {
   xsw.writeEndElement();                    // Cierra "itemBody"
   xsw.writeStartElement("responseProcessing"); // ResponseProcessing de la respuesta
   String template;

   // uri de la plantilla de respuesta a partir de la URL base del proyecto más su uri
   // relativa
   template = req + sc.getInitParameter("Uri match_response");
   xsw.writeAttribute("template", template);

   xsw.writeEndElement();                    // Cierra responseProcessing
   xsw.writeEndElement();                    // Cierra "assessmentItem"
   xsw.writeEndDocument();                   // Fin del documento
   xsw.flush();                           // Vacía el Writer
   xsw.close();                           // Cierra el Writer
}
}
```

ChoiceXML.java

```
package xml.items;

import java.io.*;

import javax.xml.stream.*;
import javax.servlet.ServletContext;

import utilidades.*;
```

```
import org.apache.commons.io.FileUtils;

/**
 * Clase contenedora del ítem tipo "Choice". Contiene un campo para cada parámetro del ítem
 * necesario para construir el archivo del ítem XML. Con los métodos que incluye permite crear
 * un ítem tipo "Choice" completo y crear el archivo XML en disco.
 *
 * @author   David Domínguez
 * @see      Identificador
 */
public class ChoiceXML extends AssessmentItem{

 /**
  * Pregunta del ítem.
  */
 protected String pregunta;
 /**
  * Mezclar las respuestas del ítem sí o no.
  */
 protected boolean shuffle;
 /**
  * Texto de cada una de las respuestas.
  */
 protected String[] respuestas;
 /**
  * Identificador de cada una de las respuestas.
  */
 protected Identificador[] identificadores;
 /**
  * Número máximo de respuestas seleccionables.
  */
 protected int maxChoices;
 /**
  * Array de boolean de respuesta correcta.
  */
 protected boolean[] correcto;
 /**
  * Array de boolean de respuesta fija.
  */
 protected boolean[] fija;
 /**
  * Cardinalidad de la variable respuesta.
  */
 protected String cardinalidad;

 /**
  * Constructor que permite inicializar el objeto con un primer grupo de parámetros. Son
  * los que se recogen en la primera página de la toma de datos. Para los parámetros
  * comunes a todos los ítems se llama al constructor de la superclase.
  *
  * @param   identificador  identificador del ítem
  * @param   titulo         título del ítem
  * @param   instrucciones  instrucciones del ítem
  * @param   pregunta    pregunta del ítem
  * @param   shuffle        mezclar las respuestas o no
  */
 public ChoiceXML (String identificador, String titulo, String instrucciones,
       String pregunta, boolean shuffle) {
    super(identificador, titulo, instrucciones);
    this.pregunta = pregunta;
    this.shuffle = shuffle;
 }

 /**
  * Método que inicializa el <code>array</code> de <code>boolean</code> que indica si la
  * respuesta en esa misma posición es correcta.
  *
  * @param   correcto    array con las respuestas que son correctas
  */
 public void setCorrecto(boolean[] correcto) {
    this.correcto = correcto;
 }

 /**
  * Método que inicializa el <code>array</code> de <code>boolean</code> que indica si la
  * respuesta en esa misma posición debe quedar fija en el caso de que se tengan que
```

```
 * mezclar aleatoriamente.
 *
 * @param   fija        array con las respuestas en posición fija
 */
public void setFija(boolean[] fija) {
   this.fija = fija;
}

/**
 * Método que inicializa el <code>array</code> de objetos <code>Identificador</code> que
 * contiene en cada posición el identificador de la respuesta en esa posición.
 *
 * @param   identificadores   array con los identificadores de las respuestas
 */
public void setIdentificadores(Identificador[] identificadores) {
   this.identificadores = identificadores;
}

/**
 * Método que inicializa el número máximo de respuestas que se pueden elegir. En función
 * del número, inicializa el valor de la variable <code>cardinalidad</code>.
 *
 * @param   maxChoices     número máximo de respuestas que se pueden elegir
 */
public void setMaxChoices(int maxChoices) {
   this.maxChoices = maxChoices;
   switch (maxChoices){
      case 1:     cardinalidad = "single"; break;
      default:    cardinalidad = "multiple"; break;
   }
}

/**
 * Método que inicializa el <code>array</code> con las respuestas posibles a elegir.
 *
 * @param   respuestas     array con las respuestas
 */
public void setRespuestas(String[] respuestas) {
   this.respuestas = respuestas;
}

/**
 * Método que construye el ítem XML tipo "Choice"  en la dirección que recibe como
 * parámetro con los campos que tiene almacenados. Devuelve información de estado.
 *
 * @param   path     dirección absoluta en la que crear el archivo XML
 * @param   sc       servletContext para acceder a los parámetros iniciales de "web.xml"
 * @param   req         string con la url base de la Aplicación Web requerida
 * @return           información del estado final del proceso
 * @throws  IOException si hay algún problema al intentar escribir el archivo en disco
 * @see     XMLOutputFactory#createXMLStreamWriter(java.io.Writer)
 * @see     XMLStreamWriter
 * @see     StringWriter
 */
public MensajeEstado creaItemXML(String path, ServletContext sc, String req)
      throws IOException{

   MensajeEstado msg = new MensajeEstado();     // Mensaje de salida del proceso

   try {
      XMLOutputFactory xof =  XMLOutputFactory.newInstance();
      XMLStreamWriter xsw = null;            // Writer para escribir datos XML
      StringWriter sw = new StringWriter(); // Buffer en el que escribir
      xsw = xof.createXMLStreamWriter(sw);  // El writer XML escribe en el buffer
      // Escribe cabecera común a todos los tipos de pregunta
      writeCabecera(xsw, sc);
      xsw.writeAttribute("identifier", identificador);
      xsw.writeAttribute("title", titulo);
      xsw.writeStartElement("responseDeclaration");  // Declara la variable de respuesta
      xsw.writeAttribute("identifier", "RESPONSE");
      xsw.writeAttribute("cardinality", cardinalidad);
      xsw.writeAttribute("baseType", "identifier");
      xsw.writeStartElement("correctResponse"); // Valores de las respuestas correctas
      for (int i = 0; i < respuestas.length; i++){
         if (correcto[i]){
            xsw.writeStartElement("value");        // Valor correcto
```

```
            xsw.writeCharacters(identificadores[i].getIdentificador());
            xsw.writeEndElement();              // Cierra "value"
         }
      }
      xsw.writeEndElement();                    // Cierra "correctResponse"
      xsw.writeEndElement();                    // Cierra "responseDeclaration"
      xsw.writeStartElement("outcomeDeclaration"); // Variable de puntuación
      xsw.writeAttribute("identifier", "SCORE");
      xsw.writeAttribute("cardinality", "single");
      xsw.writeAttribute("baseType", "integer");
      xsw.writeStartElement("defaultValue");       // Valor por defecto de la salida
      xsw.writeStartElement("value");              // Valor
      xsw.writeCharacters("0");
      xsw.writeEndElement();                    // Cierra "value"
      xsw.writeEndElement();                    // Cierra "defaultValue"
      xsw.writeEndElement();                    // Cierra "outcomeDeclaration"
      xsw.writeStartElement("itemBody");           // Cuerpo del ítem
      // Las instrucciones se escriben mediante el parser neutro puesto que es XHTML, sin
      // que el writer XML modifique los datos
      if (instrucciones != null)
         ParserNeutro.escribeInstruccionesXML(instrucciones, xsw);
      xsw.writeStartElement("choiceInteraction");     // Interacción tipo choice
      xsw.writeAttribute("responseIdentifier","RESPONSE");
      xsw.writeAttribute("shuffle",Boolean.toString(shuffle));// Mezclar respuestas
      xsw.writeAttribute("maxChoices",Integer.toString(maxChoices));// Máximas elecciones
      if (pregunta != null && !pregunta.equals("")){
         xsw.writeStartElement("prompt");       // Escribe la pregunta si la hay
         xsw.writeCharacters(pregunta);
         xsw.writeEndElement();                 // Cierra "prompt"
      }
      for (int i = 0; i < respuestas.length; i++){ // Escribe las respuestas posibles
         xsw.writeStartElement("simpleChoice");
         xsw.writeAttribute("identifier", identificadores[i].getIdentificador());
         if (shuffle)                           // Si mezcla marca las fijas
            if (fija[i])                        // Respuesta fija
               xsw.writeAttribute("fixed", Boolean.toString(fija[i]));
         xsw.writeCharacters(respuestas[i]);       // Respuestas
         xsw.writeEndElement();                 // Cierra "simpleChoice"
      }
      xsw.writeEndElement();                    // Cierra "choiceInteraction"
      // Escribe el final del ítem XML y cierra el writer
      writePie(xsw, sc, req);

      // Vacía y cierra el Writer
      sw.flush();
      sw.close();
      // Pasa a un string el contenido del Writer
      String fich = sw.toString();
      // Fichero de salida
      File file = new File(path + identificador + ".xml");
      // Escribe en el fichero de salida el contenido del ítem
      FileUtils.writeStringToFile(file, fich, null);
   } catch (XMLStreamException e) {
      // Error en el procesamiento de los datos XML
      msg.setMensaje("Error procesando el archivo XML");
      msg.setEstado(false);
   }
   if (msg.isEstado()){
      // Si todo ha ido bien se iforma del resultado
      msg.setMensaje("Ítem creado con éxito");
   }

   return msg;
 }
}
```

GapmatchXML.java

```
package xml.items;

import java.io.File;
import java.io.IOException;
import java.io.StringWriter;
```

```
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import javax.servlet.ServletContext;

import org.apache.commons.io.FileUtils;

import utilidades.*;

/**
 * Clase contenedora del ítem tipo "Gap Match". Contiene un campo para cada parámetro del ítem
 * necesario para construir el archivo del ítem XML. Con los métodos que incluye permite crear
 * un ítem tipo "Gap Match" completo y crear el archivo XML en disco.
 *
 * @author   David Domínguez
 * @see      Identificador
 * @see      EnteroPositivo
 */
public class GapmatchXML extends AssessmentItem {

 /**
  * Pregunta del ítem.
  */
 protected String pregunta;
 /**
  * Mezclar las opciones aleatoriamente o no.
  */
 protected boolean shuffle;
 /**
  * Textos en los que están los huecos imbuidos.
  */
 protected String[] textos;
 /**
  * Valores posibles que pueden tomar los huecos.
  */
 protected String[] opciones;
 /**
  * Identificadores de las opciones.
  */
 protected Identificador[] identOpciones;
 /**
  * Identificadores de los huecos.
  */
 protected Identificador[] identHuecos;
 /**
  * Máximo de asociaciones de cada opción.
  */
 protected EnteroPositivo[] maxOpciones;
 /**
  * Tabla con el número de la opción correcta de cada hueco. Es el índice de las tablas de
  * opciones.
  */
 protected int[] respCorrecta;
 /**
  * Array con las opciones que deben ser fijas.
  */
 protected boolean[] fija;
 /**
  * Cardinalidad de la variable respuesta.
  */
 protected String cardinalidad;

 /**
  * Constructor que permite inicializar el objeto con un primer grupo de parámetros. Son
  * los que se recogen en la primera página de la toma de datos. Para los parámetros
  * comunes a todos los ítems se llama al constructor de la superclase.
  *
  * @param   identificador  identificador del ítem
  * @param   titulo         título del ítem
  * @param   instrucciones  instrucciones del ítem
  * @param   pregunta    pregunta del ítem
  * @param   shuffle        mezclar las respuestas o no
  */
 public GapmatchXML(String identificador, String titulo, String instrucciones,
       String pregunta, boolean shuffle) {
    super(identificador, titulo, instrucciones);
```

```
        this.pregunta = pregunta;
        this.shuffle = shuffle;
    }

    /**
     * Método que inicializa el <code>array</code> de objetos <code>Identificador</code> que
     * contiene en cada posición el identificador del hueco en esa misma posición.
     *
     * @param   identHuecos    array con los identificadores de los huecos
     */
    public void setIdentHuecos(Identificador[] identHuecos) {
        this.identHuecos = identHuecos;
        // Si el número de huecos es 1 la cardinalidad será simple
        switch(identHuecos.length){
            case 1: cardinalidad = "single";break;
            default: cardinalidad = "multiple";break;
        }
    }

    /**
     * Método que inicializa el <code>array</code> de objetos <code>Identificador</code> que
     * contiene en cada posición el identificador de la opción en esa misma posición.
     *
     * @param   identOpciones     array con los identificadores de las opciones
     */
    public void setIdentOpciones(Identificador[] identOpciones) {
        this.identOpciones = identOpciones;
    }

    /**
     * Método que inicializa el <code>array</code> de <code>EnteroPositivo</code> conteniendo
     * en cada elemento el número máximo de huecos con los que esa opción se puede asociar.
     *
     * @param   maxOpciones    array con los números máximos de asociaciones de cada opción
     */
    public void setMaxOpciones(EnteroPositivo[] maxOpciones) {
        this.maxOpciones = maxOpciones;
    }

    /**
     * Método que inicializa el <code>array</code> de <code>String</code> conteniendo en cada
     * elemento una de las opciones con las que se pueden asociar los huecos.
     *
     * @param   opciones    array con las opciones
     */
    public void setOpciones(String[] opciones) {
        this.opciones = opciones;
    }

    /**
     * Método que inicializa el <code>array</code> de <code>String</code> con los textos que
     * rodean a los huecos. Hay un texto más que el número de huecos.
     *
     * @param   textos         array con los textos que rodean a los huecos
     */
    public void setTextos(String[] textos) {
        this.textos = textos;
    }

    /**
     * Método que inicializa el <code>array</code> de enteros que indica la posición de la
     * opción que es correcta para cada hueco.
     *
     * @param   respCorrecta   array de enteros con la posición de la opción correcta de cada
     *                         hueco
     */
    public void setRespCorrecta(String[] respCorrecta) {
        this.respCorrecta = new int[respCorrecta.length];
        for (int i = 0; i < respCorrecta.length; i++)
            this.respCorrecta[i] = Integer.parseInt(respCorrecta[i]);
    }

    /**
     * Método que inicializa el <code>array</code> de <code>boolean</code> indicando si esa
     * opción debe quedarse en esa posición o mezclarse si se deben mezclar aleatoriamente.
     *
```

```
 * @param   fija        array de boolean indicando posiciones fijas
 */
public void setFija(boolean[] fija) {
   this.fija = fija;
}

/**
 * Método que construye el ítem XML tipo "Gap Match" con los campos que tiene almacenados
 * en la dirección que recibe como parámetro. Devuelve información de estado.
 *
 * @param   path     dirección absoluta en la que crear el archivo XML
 * @param   sc       servletContext para acceder a los parámetros iniciales de "web.xml"
 * @param   req         string con la url base de la Aplicación Web requerida
 * @return           información del estado final del proceso
 * @throws  IOException si hay algún problema al intentar escribir el archivo en disco
 * @see     XMLOutputFactory#createXMLStreamWriter(java.io.Writer)
 * @see     XMLStreamWriter
 * @see     StringWriter
 */
public MensajeEstado creaItemXML(String path, ServletContext sc, String req)
      throws IOException {

   MensajeEstado msg = new MensajeEstado();     // Mensaje de salida del método

   try {
      XMLOutputFactory xof =  XMLOutputFactory.newInstance();
      XMLStreamWriter xsw = null;            // Writer para escribir datos XML
      StringWriter sw = new StringWriter();  // Buffer en el que escribir
      xsw = xof.createXMLStreamWriter(sw);   // El writer XML escribe en el buffer
      // Escribe cabecera común a todos los tipos de pregunta
      writeCabecera(xsw, sc);
      xsw.writeAttribute("identifier", identificador);
      xsw.writeAttribute("title", titulo);
      xsw.writeStartElement("responseDeclaration");   // Declara la variable de respuesta
      xsw.writeAttribute("identifier", "RESPONSE");
      xsw.writeAttribute("cardinality", cardinalidad);
      xsw.writeAttribute("baseType", "identifier");
      xsw.writeStartElement("correctResponse"); // Valores de las respuestas correctas
      for (int i = 0; i < identHuecos.length; i++){
         xsw.writeStartElement("value");        // Valor correcto
         xsw.writeCharacters(identHuecos[i].getIdentificador() + " ");
         xsw.writeCharacters(identOpciones[respCorrecta[i]].getIdentificador());
         xsw.writeEndElement();              // Cierra "value"
      }
      xsw.writeEndElement();                    // Cierra "correctResponse"
      xsw.writeEndElement();                    // Cierra "responseDeclaration"
      xsw.writeStartElement("outcomeDeclaration"); // Variable de puntuación
      xsw.writeAttribute("identifier", "SCORE");
      xsw.writeAttribute("cardinality", "single");
      xsw.writeAttribute("baseType", "integer");
      xsw.writeStartElement("defaultValue");       // Valor por defecto de la salida
      xsw.writeStartElement("value");              // Valor
      xsw.writeCharacters("0");
      xsw.writeEndElement();                    // Cierra "value"
      xsw.writeEndElement();                    // Cierra "defaultValue"
      xsw.writeEndElement();                    // Cierra "outcomeDeclaration"
      xsw.writeStartElement("itemBody");           // Cuerpo del ítem
      // Las instrucciones se escriben mediante el parser neutro puesto que es xhtml, sin
      // que el writer XML modifique los datos
      if (instrucciones != null)
         ParserNeutro.escribeInstruccionesXML(instrucciones, xsw);
      xsw.writeStartElement("gapMatchInteraction");  // Interacción tipo GapMatch
      xsw.writeAttribute("responseIdentifier","RESPONSE");
      xsw.writeAttribute("shuffle",Boolean.toString(shuffle));   // Mezclar opciones
      if (pregunta != null && !pregunta.equals("")){
         xsw.writeStartElement("prompt");       // Escribe la pregunta si la hay
         xsw.writeCharacters(pregunta);
         xsw.writeEndElement();                 // Cierra "prompt"
      }
      for (int i = 0; i < identOpciones.length; i++){// Escribe las opciones posibles
         xsw.writeStartElement("gapText");
         xsw.writeAttribute("identifier", identOpciones[i].getIdentificador());
         xsw.writeAttribute("matchMax",Integer.toString(
               maxOpciones[i].getEnteroPositivo()));
         if (shuffle)                        // Si mezcla marca las fijas
            if (fija[i])                     // Respuesta fija
```

```
                xsw.writeAttribute("fixed", Boolean.toString(fija[i]));
            xsw.writeCharacters(opciones[i]);
            xsw.writeEndElement();                  // Cierra "gapText"
         }
         xsw.writeStartElement("blockquote");
         xsw.writeStartElement("p");                  // Mete el texto en un párrafo
         for (int i = 0; i < identHuecos.length; i++){
            if (textos[i] != null && !textos[i].equals(""))      // Escribe texto 'i'
               // Los Textos se escriben mediante el método de escribir textos XML para
               // sustituir los caracteres de nueva línea por los equivalentes <br /> de
               // XHTML
               ParserNeutro.escribeTextoXML(textos[i], xsw);
            xsw.writeEmptyElement("gap");          // Hueco 'i'
            // Identificador del hueco 'i'
            xsw.writeAttribute("identifier", identHuecos[i].getIdentificador());
         }
         // Escribe el último texto
         if (textos[textos.length - 1] != null && !textos[textos.length - 1].equals(""))
            ParserNeutro.escribeTextoXML(textos[textos.length - 1], xsw);
         xsw.writeEndElement();                    // Cierra "p"
         xsw.writeEndElement();                    // Cierra "blockquote"
         xsw.writeEndElement();                    // Cierra "gapMatchInteraction"
         // Escribe el final del ítem XML y cierra el writer
         writePie(xsw, sc, req);

         // Vacía y cierra el Writer
         sw.flush();
         sw.close();
         // Escribe en un String el contenido del Writer
         String fich = sw.toString();
         // Fichero de salida
         File file = new File(path + identificador + ".xml");
         // Escribe en el fichero de salida
         FileUtils.writeStringToFile(file, fich, null);
      } catch (XMLStreamException e) {
         // Error en el procesamiento de los datos XML
         msg.setMensaje("Error procesando el archivo XML");
         msg.setEstado(false);
      }
      if (msg.isEstado()){
         // Si todo ha ido bien se iforma del resultado
         msg.setMensaje("Ítem creado con éxito");
      }

      return msg;
   }
}
```

HottextXML.java

```
package xml.items;

import java.io.File;
import java.io.IOException;
import java.io.StringWriter;

import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import javax.servlet.ServletContext;

import org.apache.commons.io.FileUtils;

import utilidades.*;

/**
 * Clase contenedora del ítem tipo "Hot Text". Contiene un campo para cada parámetro del ítem
 * necesario para construir el archivo del ítem XML. Con los métodos que incluye permite crear
 * un ítem tipo "Hot Text" completo y crear el archivo XML en disco.
 *
 * @author   David Domínguez
 * @see      Identificador
 */
public class HottextXML extends AssessmentItem {
```

```
/**
 * Pregunta del ítem.
 */
protected String pregunta;
/**
 * Textos que rodean a los hot texts.
 */
protected String[] textos;
/**
 * Hot texts a incluir en el texto.
 */
protected String[] hottexts;
/**
 * Identificadores de los hottexts.
 */
protected Identificador[] identHottexts;
/**
 * Máximo número de hot texts seleccionables.
 */
protected int maxChoices;
/**
 * Array de boolean con los hottexts correctos.
 */
protected boolean[] correcto;
/**
 * Cardinalidad de la variable de respuesta.
 */
protected String cardinalidad;

/**
 * Constructor que permite inicializar el objeto con un primer grupo de parámetros. Son
 * los que se recogen en la primera página de la toma de datos. Para los parámetros
 * comunes a todos los ítems se llama al constructor de la superclase.
 *
 * @param   identificador  identificador del ítem
 * @param   titulo         título del ítem
 * @param   instrucciones  instrucciones del ítem
 * @param   pregunta    pregunta del ítem
 */
public HottextXML(String identificador, String titulo, String instrucciones,
      String pregunta) {
   super(identificador, titulo, instrucciones);
   this.pregunta = pregunta;
}

/**
 * Método que inicializa el <code>array</code> de <code>boolean</code> que indica si el
 * hot text en esa posición es correcto o no.
 *
 * @param   correcto    array de boolean indicando hot text correcto
 */
public void setCorrecto(boolean[] correcto) {
   this.correcto = correcto;
}

/**
 * Método que inicializa el <code>array</code> de <code>String</code> que contiene cada
 * uno de los hot texts a incluir en el texto.
 *
 * @param   hottexts       array con los hot texts
 */
public void setHottexts(String[] hottexts) {
   this.hottexts = hottexts;
}

/**
 * Método que inicializa el <code>array</code> de objetos <code>Identificador</code> que
 * contiene en cada posición el identificador del hot text en esa misma posición.
 *
 * @param   identHottexts  array con los identificadores de los hot texts
 */
public void setIdentHottexts(Identificador[] identHottexts) {
   this.identHottexts = identHottexts;
}
```

```
/**
 * Método que inicializa el número máximo de hot texts que se pueden seleccionar como
 * correctos. En función de éste, inicializa <code>cardinalidad</code> a un valor adecuado.
 *
 * @param   maxChoices     entero indicando el número máximo de elecciones
 */
public void setMaxChoices(int maxChoices) {
   this.maxChoices = maxChoices;
   switch(maxChoices){
      case 1: cardinalidad = "single"; break;
      default: cardinalidad = "multiple"; break;
   }
}

/**
 * Método que inicializa el <code>array</code> de <code>String</code> con los textos que
 * rodean a los huecos. El número de textos es igual al número de huecos más uno.
 *
 * @param   textos      array con los textos que rodean a los huecos
 */
public void setTextos(String[] textos) {
   this.textos = textos;
}

/**
 * Método que construye el ítem XML tipo "Hot Text" con los campos que tiene almacenados
 * en la dirección que recibe como parámetro. Devuelve información de estado.
 *
 * @param   path     dirección absoluta en la que crear el archivo XML
 * @param   sc       servletContext para acceder a los parámetros iniciales de "web.xml"
 * @param   req         string con la url base de la Aplicación Web requerida
 * @return           información del estado final del proceso
 * @throws  IOException si hay algún problema al intentar escribir el archivo en disco
 * @see     XMLOutputFactory#createXMLStreamWriter(java.io.Writer)
 * @see     XMLStreamWriter
 * @see     StringWriter
 */
public MensajeEstado creaItemXML(String path, ServletContext sc, String req)
      throws IOException {

   MensajeEstado msg = new MensajeEstado();     // Mensaje de salida del método

   try {
      XMLOutputFactory xof =  XMLOutputFactory.newInstance();
      XMLStreamWriter xsw = null;            // Writer para escribir datos XML
      StringWriter sw = new StringWriter();  // Buffer en el que escribir
      xsw = xof.createXMLStreamWriter(sw);   // El writer XML escribe en el buffer
      // Escribe cabecera común a todos los tipos de pregunta
      writeCabecera(xsw, sc);
      xsw.writeAttribute("identifier", identificador);
      xsw.writeAttribute("title", titulo);
      xsw.writeStartElement("responseDeclaration");  // Declara la variable de respuesta
      xsw.writeAttribute("identifier", "RESPONSE");
      xsw.writeAttribute("cardinality", cardinalidad);
      xsw.writeAttribute("baseType", "identifier");
      xsw.writeStartElement("correctResponse");    // Respuestas correctas
      for (int i = 0; i < correcto.length; i++){
         if (correcto[i]){
            xsw.writeStartElement("value");        // Valor correcto
            xsw.writeCharacters(identHottexts[i].getIdentificador());
            xsw.writeEndElement();                 // Cierra "value"
         }
      }
      xsw.writeEndElement();                       // Cierra "correctResponse"
      xsw.writeEndElement();                       // Cierra "responseDeclaration"
      xsw.writeStartElement("outcomeDeclaration"); // Variable de puntuación
      xsw.writeAttribute("identifier", "SCORE");
      xsw.writeAttribute("cardinality", "single");
      xsw.writeAttribute("baseType", "integer");
      xsw.writeStartElement("defaultValue");       // Valor por defecto de la salida
      xsw.writeStartElement("value");              // Valor
      xsw.writeCharacters("0");
      xsw.writeEndElement();                       // Cierra "value"
      xsw.writeEndElement();                       // Cierra "defaultValue"
      xsw.writeEndElement();                       // Cierra "outcomeDeclaration"
      xsw.writeStartElement("itemBody");           // Cuerpo del ítem
```

```
        // Las instrucciones se escriben directamente puesto que es xhtml, sin que el
        // writer XML modifique los datos
        if (instrucciones != null)
           ParserNeutro.escribeInstruccionesXML(instrucciones, xsw);
        xsw.writeStartElement("hottextInteraction"); // Interacción tipo hottext
        xsw.writeAttribute("responseIdentifier","RESPONSE");
        xsw.writeAttribute("maxChoices",Integer.toString(maxChoices));
        if (pregunta != null && !pregunta.equals("")){
           xsw.writeStartElement("prompt");       // Escribe la pregunta si la hay
           xsw.writeCharacters(pregunta);
           xsw.writeEndElement();                 // Cierra "prompt"
        }
        xsw.writeStartElement("p");                  // Mete el texto en un párrafo
        for (int i = 0; i < identHottexts.length; i++){
           if (textos[i] != null && !textos[i].equals(""))      // Escribe texto 'i'
              // Los Textos se escriben mediante el método de escribir textos XML para
              // sustituir los caracteres de nueva línea por los equivalentes <br /> de
              // XHTML
              ParserNeutro.escribeTextoXML(textos[i], xsw);
           xsw.writeStartElement("hottext");         // Hottext 'i'
           xsw.writeAttribute("identifier", identHottexts[i].getIdentificador());
           xsw.writeCharacters(hottexts[i]);
           xsw.writeEndElement();                 // Cierra "hottext" 'i'
        }
        // Escribe el último texto
        if (textos[textos.length - 1] != null && !textos[textos.length - 1].equals(""))
           ParserNeutro.escribeTextoXML(textos[textos.length - 1], xsw);
        xsw.writeEndElement();                    // Cierra "p"
        xsw.writeEndElement();                    // Cierra "hottextInteraction"
        // Escribe el final del ítem XML y cierra el writer
        writePie(xsw, sc, req);

        // Vacía y cierra el Writer
        sw.flush();
        sw.close();
        // Escribe en un String el contenido del Writer
        String fich = sw.toString();
        // Fichero de salida
        File file = new File(path + identificador + ".xml");
        // Escribe en el fichero de salida
        FileUtils.writeStringToFile(file, fich, null);
     } catch (XMLStreamException e) {
        // Error en el procesamiento de los datos XML
        msg.setMensaje("Error procesando el archivo XML");
        msg.setEstado(false);
     }
     if (msg.isEstado()){
        // Si todo ha ido bien se iforma del resultado
        msg.setMensaje("Ítem creado con éxito");
     }

     return msg;
  }
}
```

InlinechoiceXML.java

```
package xml.items;

import java.io.File;
import java.io.IOException;
import java.io.StringWriter;

import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import javax.servlet.ServletContext;

import org.apache.commons.io.FileUtils;

import utilidades.*;

/**
 * Clase contenedora del ítem tipo "Inline Choice". Contiene un campo para cada parámetro del
```

```
 * ítem necesario para construir el archivo del ítem XML. Con los métodos que incluye permite
 * crear un ítem tipo "Inline Choice" completo y crear el archivo XML en disco.
 *
 * @author   David Domínguez
 * @see      Identificador
 */
public class InlinechoiceXML extends AssessmentItem {
 /**
  * Mezclar las respuestas aleatoriamente sí o no.
  */
 protected boolean shuffle;
 /**
  * Textos de cada una de las respuestas.
  */
 protected String[] respuestas;
 /**
  * Texto antes de las respuestas.
  */
 protected String texto1;
 /**
  * Texto tras las respuestas.
  */
 protected String texto2;
 /**
  * Identificador de cada una de las respuestas.
  */
 protected Identificador[] identificadores;
 /**
  * Posición de la respuesta correcta.
  */
 protected int correcto;
 /**
  * Array con las respuestas que deben ser fijas.
  */
 protected boolean[] fija;
 /**
  * Cardinalidad de la variable respuesta que en este caso siempre es "single".
  */
 String cardinalidad = "single";


 /**
  * Constructor que permite inicializar el objeto con un primer grupo de parámetros. Son
  * los que se recogen en la primera página de la toma de datos. Para los parámetros
  * comunes a todos los ítems se llama al constructor de la superclase.
  *
  * @param   identificador  identificador del ítem
  * @param   titulo         título del ítem
  * @param   instrucciones  instrucciones del ítem
  * @param   shuffle        mezclar las respuestas o no
  */
 public InlinechoiceXML(String identificador, String titulo, String instrucciones,
       boolean shuffle) {
    super(identificador, titulo, instrucciones);
    this.shuffle = shuffle;
 }

 /**
  * Método que inicializa el entero que indica la posición de la respuesta que es correcta.
  *
  * @param   correcto    posición de la opción que es correcta
  */
 public void setCorrecto(int correcto) {
    this.correcto = correcto;
 }

 /**
  * Método que inicializa el <code>array</code> de <code>boolean</code> que indica si la
  * respuesta en esa misma posición debe quedar fija en el caso de que se tengan que
  * mezclar aleatoriamente.
  *
  * @param   fija        array con las respuestas en posición fija
  */
 public void setFija(boolean[] fija) {
    this.fija = fija;
 }
```

```
/**
 * Método que inicializa el <code>array</code> de objetos <code>Identificador</code> que
 * contiene en cada posición el identificador de la respuesta en esa posición.
 *
 * @param   identificadores   array con los identificadores de las respuestas
 */
public void setIdentificadores(Identificador[] identificadores) {
   this.identificadores = identificadores;
}

/**
 * Método que inicializa el <code>array</code> con las respuestas posibles a elegir.
 *
 * @param   respuestas     array con las respuestas
 */
public void setRespuestas(String[] respuestas) {
   this.respuestas = respuestas;
}

/**
 * Método que inicializa el <code>array</code> con los textos antes y después de las
 * opciones.
 *
 * @param   texto1         string con el texto antes de las respuestas
 * @param   texto2         string con el texto después de las respuestas
 */
public void setTextos(String texto1, String texto2) {
   this.texto1 = texto1;
   this.texto2 = texto2;
}

/**
 * Método que construye el ítem XML tipo "Inline Choice"  en la dirección que recibe como
 * parámetro con los campos que tiene almacenados. Devuelve información de estado.
 *
 * @param   path     dirección absoluta en la que crear el archivo XML
 * @param   sc       servletContext para acceder a los parámetros iniciales de "web.xml"
 * @param   req         string con la url base de la Aplicación Web requerida
 * @return           información del estado final del proceso
 * @throws  IOException si hay algún problema al intentar escribir el archivo en disco
 * @see     XMLOutputFactory#createXMLStreamWriter(java.io.Writer)
 * @see     XMLStreamWriter
 * @see     StringWriter
 */
public MensajeEstado creaItemXML(String path, ServletContext sc, String req)
      throws IOException {

   MensajeEstado msg = new MensajeEstado();     // Mensaje de salida del método

   try {
      XMLOutputFactory xof =  XMLOutputFactory.newInstance();
      XMLStreamWriter xsw = null;            // Writer para escribir datos XML
      StringWriter sw = new StringWriter();  // Buffer en el que escribir
      xsw = xof.createXMLStreamWriter(sw);   // El writer XML escribe en el buffer
      // Escribe cabecera común a todos los tipos de pregunta
      writeCabecera(xsw, sc);
      xsw.writeAttribute("identifier", identificador);
      xsw.writeAttribute("title", titulo);
      xsw.writeStartElement("responseDeclaration");   // Variable de respuesta
      xsw.writeAttribute("identifier", "RESPONSE");
      xsw.writeAttribute("cardinality", cardinalidad);
      xsw.writeAttribute("baseType", "identifier");
      xsw.writeStartElement("correctResponse");    // Valor de la respuesta correcta
      xsw.writeStartElement("value");              // Valor correcto
      xsw.writeCharacters(identificadores[correcto].getIdentificador());
      xsw.writeEndElement();                    // Cierra "value"
      xsw.writeEndElement();                    // Cierra "correctResponse"
      xsw.writeEndElement();                    // Cierra "responseDeclaration"
      xsw.writeStartElement("outcomeDeclaration"); // Variable de puntuación
      xsw.writeAttribute("identifier", "SCORE");
      xsw.writeAttribute("cardinality", "single");
      xsw.writeAttribute("baseType", "integer");
      xsw.writeStartElement("defaultValue");       // Valor por defecto de la salida
      xsw.writeStartElement("value");              // Valor
      xsw.writeCharacters("0");
```

```
            xsw.writeEndElement();                    // Cierra "value"
            xsw.writeEndElement();                    // Cierra "defaultValue"
            xsw.writeEndElement();                    // Cierra "outcomeDeclaration"
            xsw.writeStartElement("itemBody");           // Cuerpo del ítem
            // Las instrucciones se escriben directamente puesto que es xhtml, sin que el
            // writer XML modifique los datos
            if (instrucciones != null)
               ParserNeutro.escribeInstruccionesXML(instrucciones, xsw);
            xsw.writeStartElement("blockquote");
            xsw.writeStartElement("p");                  // Mete el texto en un párrafo
            // Los Textos se escriben mediante el método de escribir textos XML para sustituir
            // los caracteres de nueva línea por los equivalentes <br /> de XHTML
            if (texto1 != null && !texto1.equals(""))
               ParserNeutro.escribeTextoXML(texto1, xsw);      // Texto antes de las opciones
            xsw.writeStartElement("inlineChoiceInteraction"); // Interacción tipo inlinechoice
            xsw.writeAttribute("responseIdentifier","RESPONSE");
            xsw.writeAttribute("shuffle",Boolean.toString(shuffle));// Mezclar preguntas
            for (int i = 0; i < respuestas.length; i++){ // Escribe las respuestas posibles
               xsw.writeStartElement("inlineChoice");
               xsw.writeAttribute("identifier", identificadores[i].getIdentificador());
               if (shuffle)                        // Si mezcla marca las fijas
                  if (fija[i])                     // Respuesta fija
                     xsw.writeAttribute("fixed", Boolean.toString(fija[i]));
               xsw.writeCharacters(respuestas[i]);
               xsw.writeEndElement();                 // Cierra "inlineChoice"
            }
            xsw.writeEndElement();                    // Cierra "inlineChoiceInteraction"
            // Los Textos se escriben mediante el método de escribir textos XML para sustituir
            // los caracteres de nueva línea por los equivalentes <br /> de XHTML
            if (texto2 != null && !texto2.equals(""))
               ParserNeutro.escribeTextoXML(texto2, xsw);      // Texto tras las opciones
            xsw.writeEndElement();                       // Cierra "p"
            xsw.writeEndElement();                       // Cierra "blockquote"
            // Escribe el final del ítem XML y cierra el writer
            writePie(xsw, sc, req);

            // Vacía y cierra el Writer
            sw.flush();
            sw.close();
            // Pasa a un String en contenido del Writer
            String fich = sw.toString();
            // Fichero de salida
            File file = new File(path + identificador + ".xml");
            // Escribe en el fichero de salida
            FileUtils.writeStringToFile(file, fich, null);
         } catch (XMLStreamException e) {
            // Error en el procesamiento de los datos XML
            msg.setMensaje("Error procesando el archivo XML");
            msg.setEstado(false);
         }
         if (msg.isEstado()){
            // Si todo ha ido bien se iforma del resultado
            msg.setMensaje("Ítem creado con éxito");
         }

         return msg;
      }
   }
```

MatchXML.java

```
package xml.items;

import java.io.File;
import java.io.IOException;
import java.io.StringWriter;

import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import javax.servlet.ServletContext;

import org.apache.commons.io.FileUtils;
```

```
import utilidades.*;

/**
 * Clase contenedora del ítem tipo "Match". Contiene un campo para cada parámetro del ítem
 * necesario para construir el archivo del ítem XML. Con los métodos que incluye permite crear
 * un ítem tipo "Match" completo y crear el archivo XML en disco.
 *
 * @author   David Domínguez
 * @see      Identificador
 * @see      EnteroPositivo
 */
public class MatchXML extends AssessmentItem {

 /**
  * Pregunta del ítem.
  */
 protected String pregunta;
 /**
  * Mezclar las respuestas aleatoriamente sí o no.
  */
 protected boolean shuffle;
 /**
  * Respuestas del grupo 1.
  */
 protected String[] respuestas1;
 /**
  * Respuestas del grupo 2.
  */
 protected String[] respuestas2;
 /**
  * Identificadores de las respuestas del grupo 1.
  */
 protected Identificador[] identificadores1;
 /**
  * Identificadores de las respuestas del grupo 2.
  */
 protected Identificador[] identificadores2;
 /**
  * Posiciones fijas de las respuestas del grupo 1.
  */
 protected boolean[] fija1;
 /**
  * Posiciones fijas de las respuestas del grupo 2.
  */
 protected boolean[] fija2;
 /**
  * Número total máximo de elecciones.
  */
 protected int maxChoices;
 /**
  * Número máximo de elecciones  de cada respuesta del grupo 1.
  */
 protected EnteroPositivo[] max1;
 /**
  * Número máximo de elecciones  de cada respuesta del grupo 2.
  */
 protected EnteroPositivo[] max2;
 /**
  * Combinaciones correctas entre los dos grupos.
  */
 protected boolean[][] respCorrecta;
 /**
  * Cardinalidad de la variable de respuesta.
  */
 protected String cardinalidad;


 /**
  * Constructor que permite inicializar el objeto con un primer grupo de parámetros. Son
  * los que se recogen en la primera página de la toma de datos. Para los parámetros
  * comunes a todos los ítems se llama al constructor de la superclase.
  *
  * @param   identificador     identificador del ítem
  * @param   titulo            título del ítem
  * @param   instrucciones     instrucciones del ítem
  * @param   pregunta       pregunta del ítem
```

```
 * @param   shuffle           mezclar las respuestas o no
 */
public MatchXML(String identificador, String titulo, String instrucciones,
      String pregunta, boolean shuffle) {
   super(identificador, titulo, instrucciones);
   this.pregunta = pregunta;
   this.shuffle = shuffle;
}

/**
 * Método que inicializa el <code>array</code> de <code>boolean</code> que indica si la
 * respuesta del grupo 1 en esa misma posición debe quedar fija en el caso de que se
 * tengan que mezclar aleatoriamente.
 *
 * @param   fija1          array con las respuestas en posición fija del grupo 1
 */
public void setFija1(boolean[] fija1) {
   this.fija1 = fija1;
}

/**
 * Método que inicializa el <code>array</code> de <code>boolean</code> que indica si la
 * respuesta del grupo 2 en esa misma posición debe quedar fija en el caso de que se
 * tengan que mezclar aleatoriamente.
 *
 * @param   fija2          array con las respuestas en posición fija del grupo 2
 */
public void setFija2(boolean[] fija2) {
   this.fija2 = fija2;
}

/**
 * Método que inicializa el <code>array</code> de objetos <code>Identificador</code> que
 * contiene en cada posición el identificador de la opción del grupo 1 en esa posición.
 *
 * @param   identificadores1  array con los identificadores de las opciones del grupo 1
 */
public void setIdentificadores1(Identificador[] identificadores1) {
   this.identificadores1 = identificadores1;
}

/**
 * Método que inicializa el <code>array</code> de objetos <code>Identificador</code> que
 * contiene en cada posición el identificador de la opción del grupo 2 en esa posición.
 *
 * @param   identificadores2  array con los identificadores de las opciones del grupo 2
 */
public void setIdentificadores2(Identificador[] identificadores2) {
   this.identificadores2 = identificadores2;
}

/**
 * Método que inicializa el <code>array</code> de <code>EnteroPositivo</code> conteniendo
 * en cada elemento el número máximo de opciones del grupo 2 con las que esa opción del
 * grupo 1 se puede asociar.
 *
 * @param   max1           array con los números máximos de asociaciones de cada
 *                         opción
 */
public void setMax1(EnteroPositivo[] max1) {
   this.max1 = max1;
}

/**
 * Método que inicializa el <code>array</code> de <code>EnteroPositivo</code> conteniendo
 * en cada elemento el número máximo de opciones del grupo 1 con las que esa opción del
 * grupo 2 se puede asociar.
 *
 * @param   max2           array con los números máximos de asociaciones de cada
 *                         opción
 */
public void setMax2(EnteroPositivo[] max2) {
   this.max2 = max2;
}

/**
```

```
 * Método que inicializa el número máximo de asociaciones en total que se pueden hacer
 * entre elementos del grupo 1 y del grupo 2.
 *
 * @param   maxChoices     número máximo de asociaciones que se pueden hacer
 */
public void setMaxChoices(int maxChoices) {
   this.maxChoices = maxChoices;
   switch(maxChoices){
      case 1:  cardinalidad = "single";
      default: cardinalidad = "multiple";
   }
}

/**
 * Método que inicializa el <code>array</code> de dos dimensiones de <code>boolean</code>
 * que indica si la combinación de la opción "i" del grupo 1 con la "j" del grupo 2 es
 * correcta o no.
 *
 * @param   respCorrecta      array con las respuestas que son correctas
 */
public void setRespCorrecta(boolean[][] respCorrecta) {
   this.respCorrecta = respCorrecta;
}

/**
 * Método que inicializa el <code>array</code> con las opciones posibles del grupo 1.
 *
 * @param   respuestas1    array con las opciones del grupo 1
 */
public void setRespuestas1(String[] respuestas1) {
   this.respuestas1 = respuestas1;
}

/**
 * Método que inicializa el <code>array</code> con las opciones posibles del grupo 2.
 *
 * @param   respuestas2    array con las opciones del grupo 2
 */
public void setRespuestas2(String[] respuestas2) {
   this.respuestas2 = respuestas2;
}

/**
 * Método que construye el ítem XML tipo "Match"  en la dirección que recibe como
 * parámetro con los campos que tiene almacenados. Devuelve información de estado.
 *
 * @param   path     dirección absoluta en la que crear el archivo XML
 * @param   sc       servletContext para acceder a los parámetros iniciales de "web.xml"
 * @param   req         string con la url base de la Aplicación Web requerida
 * @return           información del estado final del proceso
 * @throws  IOException si hay algún problema al intentar escribir el archivo en disco
 * @see     XMLOutputFactory#createXMLStreamWriter(java.io.Writer)
 * @see     XMLStreamWriter
 * @see     StringWriter
 */
public MensajeEstado creaItemXML(String path, ServletContext sc, String req)
      throws IOException {

   MensajeEstado msg = new MensajeEstado();     // Mensaje de salida del método

   try {
      XMLOutputFactory xof =  XMLOutputFactory.newInstance();
      XMLStreamWriter xsw = null;            // Writer para escribir datos XML
      StringWriter sw = new StringWriter();  // Buffer en el que escribir
      xsw = xof.createXMLStreamWriter(sw);   // El writer XML escribe en el buffer
      // Escribe cabecera común a todos los tipos de pregunta
      writeCabecera(xsw, sc);
      xsw.writeAttribute("identifier", identificador);
      xsw.writeAttribute("title", titulo);
      xsw.writeStartElement("responseDeclaration");   // Declara la variable de respuesta
      xsw.writeAttribute("identifier", "RESPONSE");
      xsw.writeAttribute("cardinality", cardinalidad);
      xsw.writeAttribute("baseType", "directedPair");
      xsw.writeStartElement("correctResponse"); // Valores de las respuestas correctas
      for (int i = 0; i < respuestas1.length; i++){
         for (int j = 0; j < respuestas2.length; j++)
```

```
            if (respCorrecta[i][j]){
                xsw.writeStartElement("value");        // Valor correcto
                xsw.writeCharacters(identificadores1[i].getIdentificador() + " ");
                xsw.writeCharacters(identificadores2[j].getIdentificador());
                xsw.writeEndElement();                 // Cierra "value"
            }
        }
        xsw.writeEndElement();                         // Cierra "correctResponse"
        xsw.writeEndElement();                         // Cierra "responseDeclaration"
        xsw.writeStartElement("outcomeDeclaration"); // Variable de puntuación
        xsw.writeAttribute("identifier", "SCORE");
        xsw.writeAttribute("cardinality", "single");
        xsw.writeAttribute("baseType", "integer");
        xsw.writeStartElement("defaultValue");         // Valor por defecto de la salida
        xsw.writeStartElement("value");                // Valor
        xsw.writeCharacters("0");
        xsw.writeEndElement();                         // Cierra "value"
        xsw.writeEndElement();                         // Cierra "defaultValue"
        xsw.writeEndElement();                         // Cierra "outcomeDeclaration"
        xsw.writeStartElement("itemBody");             // Cuerpo del ítem
        // Las instrucciones se escriben directamente puesto que es xhtml, sin que el
        // writer XML modifique los datos
        if (instrucciones != null)
            ParserNeutro.escribeInstruccionesXML(instrucciones, xsw);
        xsw.writeStartElement("matchInteraction");      // Interacción tipo match
        xsw.writeAttribute("responseIdentifier","RESPONSE");
        xsw.writeAttribute("shuffle",Boolean.toString(shuffle));// Mezclar opciones
        xsw.writeAttribute("maxAssociations",Integer.toString(maxChoices));
        if (pregunta != null && !pregunta.equals("")){
            xsw.writeStartElement("prompt");           // Escribe la pregunta si la hay
            xsw.writeCharacters(pregunta);
            xsw.writeEndElement();                     // Cierra "prompt"
        }
        xsw.writeStartElement("simpleMatchSet");      // Primer grupo de opciones
        for (int i = 0; i < respuestas1.length; i++){   // Escribe las respuestas posibles
            xsw.writeStartElement("simpleAssociableChoice");
            xsw.writeAttribute("identifier", identificadores1[i].getIdentificador());
            if (shuffle)                           // Si mezcla marca las fijas
                if (fija1[i])                      // Respuesta fija
                    xsw.writeAttribute("fixed", Boolean.toString(fija1[i]));
            xsw.writeAttribute("matchMax", Integer.toString(max1[i].getEnteroPositivo()));
            xsw.writeCharacters(respuestas1[i]);       // Opciones
            xsw.writeEndElement();                     // Cierra "simpleAssociableChoice"
        }
        xsw.writeEndElement();                     // Cierra el primer "simpleMatchSet"
        xsw.writeStartElement("simpleMatchSet");      // Segundo grupo de opciones
        for (int i = 0; i < respuestas2.length; i++){   // Escribe las respuestas posibles
            xsw.writeStartElement("simpleAssociableChoice");
            xsw.writeAttribute("identifier", identificadores2[i].getIdentificador());
            if (shuffle)                           // Si mezcla marca las fijas
                if (fija2[i])                      // Respuesta fija
                    xsw.writeAttribute("fixed", Boolean.toString(fija2[i]));
            xsw.writeAttribute("matchMax", Integer.toString(max2[i].getEnteroPositivo()));
            xsw.writeCharacters(respuestas2[i]);
            xsw.writeEndElement();                     // Cierra "simpleAssociableChoice"
        }
        xsw.writeEndElement();                     // Cierra el segundo "simpleMatchSet"
        xsw.writeEndElement();                     // Cierra "choiceInteraction"
        // Escribe el final del ítem XML y cierra el writer
        writePie(xsw, sc, req);

        // Vacía y cierra el Writer
        sw.flush();
        sw.close();
        // Pasa a String en contenido del Writer
        String fich = sw.toString();
        // Fichero de salida
        File file = new File(path + identificador + ".xml");
        // Escribe en el fichero de salida
        FileUtils.writeStringToFile(file, fich, null);
    } catch (XMLStreamException e) {
        // Error en el procesamiento de los datos XML
        msg.setMensaje("Error procesando el archivo XML");
        msg.setEstado(false);
    }
    if (msg.isEstado()){
```

```
        // Si todo ha ido bien se iforma del resultado
        msg.setMensaje("Ítem creado con éxito");
    }

    return msg;
 }
}
```

TextentryXML.java

```
package xml.items;

import java.io.File;
import java.io.IOException;
import java.io.StringWriter;

import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import javax.servlet.ServletContext;

import org.apache.commons.io.FileUtils;

import utilidades.*;

/**
 * Clase contenedora del ítem tipo "Text Entry". Contiene un campo para cada parámetro del
 * ítem necesario para construir el archivo del ítem XML. Con los métodos que incluye permite
 * crear un ítem tipo "Text Entry" completo y crear el archivo XML en disco.
 *
 * @author   David Domínguez
 */
public class TextentryXML extends AssessmentItem {

 /**
  * Respuesta correcta del ítem.
  */
 protected String respuesta;
 /**
  * Longitud esperada de la respuesta; inicializada a cero si no se debe incluir el atributo.
  */
 protected int longitud = 0;
 /**
  * Texto antes de la respuesta.
  */
 protected String texto1;
 /**
  * Texto después de la respuesta.
  */
 protected String texto2;
 /**
  * Cardinalidad de la variable de respuesta. En este caso siempre es "single".
  */
 protected String cardinalidad = "single";

 /**
  * Constructor que permite inicializar el objeto con un primer grupo de parámetros. Son
  * los que se recogen en la primera página de la toma de datos. En este caso sólo se
  * recogen los que son comunes, así que simplemente llama al constructor de la superclase.
  *
  * @param   identificador  identificador del ítem
  * @param   titulo         título del ítem
  * @param   instrucciones  instrucciones del ítem
  */
 public TextentryXML(String identificador, String titulo, String instrucciones) {
    super(identificador, titulo, instrucciones);
 }

 /**
  * Método que inicializa el valor de <code>longitud</code>, la longitud esperada de la
  * respuesta.
  *
  * @param   longitud    entero con la longitud esperada de la respuesta
  */
```

```
public void setLongitud(int longitud) {
   this.longitud = longitud;
}

/**
 * Método que inicializa el <code>String</code> con la respuesta correcta.
 *
 * @param   respuesta      string con la respuesta correcta
 */
public void setRespuesta(String respuesta) {
   this.respuesta = respuesta;
}

/**
 * Método que inicializa el <code>array</code> con los textos antes y después del hueco a
 * rellenar por el usuario.
 *
 * @param   texto1         string con el texto antes del hueco a rellenar
 * @param   texto2         string con el texto después del hueco a rellenar
 */
public void setTextos(String texto1, String texto2) {
   this.texto1 = texto1;
   this.texto2 = texto2;
}

/**
 * Método que construye el ítem XML tipo "Text Entry"  en la dirección que recibe como
 * parámetro con los campos que tiene almacenados. Devuelve información de estado.
 *
 * @param   path     dirección absoluta en la que crear el archivo XML
 * @param   sc       servletContext para acceder a los parámetros iniciales de "web.xml"
 * @param   req         string con la url base de la Aplicación Web requerida
 * @return           información del estado final del proceso
 * @throws  IOException si hay algún problema al intentar escribir el archivo en disco
 * @see     XMLOutputFactory#createXMLStreamWriter(java.io.Writer)
 * @see     XMLStreamWriter
 * @see     StringWriter
 */
public MensajeEstado creaItemXML(String path, ServletContext sc, String req)
      throws IOException {

   MensajeEstado msg = new MensajeEstado();     // Mensaje de salida del método

   try {
      XMLOutputFactory xof =  XMLOutputFactory.newInstance();
      XMLStreamWriter xsw = null;            // Writer para escribir datos XML
      StringWriter sw = new StringWriter();  // Buffer en el que escribir
      xsw = xof.createXMLStreamWriter(sw);   // El writer XML escribe en el buffer
      // Escribe cabecera común a todos los tipos de pregunta
      writeCabecera(xsw, sc);
      xsw.writeAttribute("identifier", identificador);
      xsw.writeAttribute("title", titulo);
      xsw.writeStartElement("responseDeclaration");  // Declara la variable de respuesta
      xsw.writeAttribute("identifier", "RESPONSE");
      xsw.writeAttribute("cardinality", cardinalidad);
      xsw.writeAttribute("baseType", "string");    // La respuesta es de tipo "string"
      xsw.writeStartElement("correctResponse");    // Valor de la respuesta correcta
      xsw.writeStartElement("value");              // Valor correcto
      xsw.writeCharacters(respuesta);
      xsw.writeEndElement();                    // Cierra "value"
      xsw.writeEndElement();                    // Cierra "correctResponse"
      xsw.writeEndElement();                    // Cierra "responseDeclaration"
      xsw.writeStartElement("outcomeDeclaration"); // Variable de puntuación
      xsw.writeAttribute("identifier", "SCORE");
      xsw.writeAttribute("cardinality", "single");
      xsw.writeAttribute("baseType", "integer");
      xsw.writeStartElement("defaultValue");       // Valor por defecto de la salida
      xsw.writeStartElement("value");              // Valor
      xsw.writeCharacters("0");
      xsw.writeEndElement();                    // Cierra "value"
      xsw.writeEndElement();                    // Cierra "defaultValue"
      xsw.writeEndElement();                    // Cierra "outcomeDeclaration"
      xsw.writeStartElement("itemBody");           // Cuerpo del ítem
      // Las instrucciones se escriben directamente puesto que es xhtml, sin que el
      // writer XML modifique los datos
      if (instrucciones != null)
```

```
            ParserNeutro.escribeInstruccionesXML(instrucciones, xsw);
        xsw.writeStartElement("blockquote");
        xsw.writeStartElement("p");                  // Mete el texto en un párrafo
        // Los Textos se escriben mediante el método de escribir textos XML para sustituir
        // los caracteres de nueva línea por los equivalentes <br /> de XHTML
        if (texto1 != null && !texto1.equals(""))
            ParserNeutro.escribeTextoXML(texto1, xsw);      // Texto antes del textentry
        xsw.writeEmptyElement("textEntryInteraction");     // Interacción tipo textentry
        xsw.writeAttribute("responseIdentifier","RESPONSE");
        if (longitud != 0)         // El atributo longitud se pone si es distinto de cero
            xsw.writeAttribute("expectedLength", Integer.toString(longitud));
        if (texto2 != null && !texto2.equals(""))
        ParserNeutro.escribeTextoXML(texto2, xsw);      // Texto tras el textentry
        xsw.writeEndElement();                     // Cierra "p"
        xsw.writeEndElement();                     // Cierra "blockquote"
        // Escribe el final del ítem XML y cierra el writer
        writePie(xsw, sc, req);

        // Vacía y cierra el Writer
        sw.flush();
        sw.close();
        // Pasa a String el contenido del Writer
        String fich = sw.toString();
        // Fichero de salida
        File file = new File(path + identificador + ".xml");
        // Escribe en el fichero de salida
        FileUtils.writeStringToFile(file, fich, null);
    } catch (XMLStreamException e) {
        msg.setMensaje("Error procesando el archivo XML");
        msg.setEstado(false);
    }
    if (msg.isEstado()){
        // Si todo ha ido bien se iforma del resultado
        msg.setMensaje("Ítem creado con éxito");
    }

    return msg;
 }
}
```

2.6 Paquete `xml.test`

AssessmentItemRef.java

```
package xml.test;

import java.io.File;
import java.net.URI;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

import utilidades.Identificador;

/**
 * Clase que representa las referencias a los ítems individuales dentro de un Test según la
 * norma QTI del IMS. Se utiliza para incorporar cada una de las cuestiones individuales dentro
 * de un Test. Contiene varios atributos que se corresponden con los atributos definidos en la
 * norma, más una variable tipo <code>File</code> del archivo del ítem a incluir en el test
 * para ayudar a calcular su URI relativa. Tal y como define la norma, esta clase hereda de la
 * clase "SectionPart", por lo tanto también contendrá sus variables de clase.
 *
 * @author David Domínguez
 *
 */
public class AssessmentItemRef extends SectionPart {


 /**
  * Objeto <code>File</code> apuntando al ítem XML al que hace referencia esta clase.
  */
 private File itemFile;
```

```
/**
 * URI relativa desde el test con la referencia al ítem XML que se debe incluir en el Test.
 */
private URI href;
/**
 * Objeto de tipo <code>Weight</code> con el peso que aplicar a la pregunta en caso de que
 * sea correcta
 */
private Weight ok;
/**
 * Objeto de tipo <code>Weight</code> con el peso que aplicar a la pregunta en caso de que
 * sea incorrecta
 */
private Weight noOk;

/**
 * Constructor de la clase referencia a un ítem. Le da los valores iniciales a los pesos
 * correcto e incorrecto del ítem, al objeto <code>File</code> que apunta al ítem, y llama
 * al constructor de la superclase para inicializar sus variables de clase con el resto de
 * los parámetros.
 *
 * @param   identificador  identificador de la referencia del ítem
 * @param   requerido      boolean indicando si el ítem es requerido
 * @param   fijo        boolean indicando si el ítem está en posición fija
 * @param   itemFile    file del ítem al que hace referencia la clase
 * @param   ok          weight con el peso del ítem en caso de respuesta correcta
 * @param   noOk        weight con el peso del ítem en caso de respuesta incorrecta
 */
public AssessmentItemRef(Identificador identificador, boolean requerido, boolean fijo,
      File itemFile, Weight ok, Weight noOk) {
   super(identificador, requerido, fijo);
   this.itemFile = itemFile;
   this.ok = ok;
   this.noOk = noOk;
}

/**
 * Calcula la URI de referencia del ítem a partir de la localización del propio ítem y de
 * la del test que se está creando, hallando la dirección relativa del ítem respecto al test.
 *
 * @param   test  file del test que se está creando
 * @see     java.net.URI
 */
public void creaItemRef (File test){
   // URI absoluta del ítem
   String itemAbsoluta = itemFile.toURI().toString();
   // URI absoluta del test
   String  testPath = test.toURI().toString();
   // String para calcular la parte de la dirección que comparten el test y el ítem
   String comun;
   // Índice para ir avanzando por cada uno de los separadores de la dirección del ítem
   int indexAbs = 0;
   // Boolean indicando que ya se ha encontrado la dirección común entre el test y el ítem
   boolean fin;
   // Va avanzando por los separadores hasta que sean diferentes las direcciones del test
   // y del ítem
   do {
      // localización del carácter separador desde la posición anterior
      indexAbs = itemAbsoluta.indexOf('/', indexAbs + 1);
      // Dirección base común a probar si es común de ambas
      comun = itemAbsoluta.substring(0, indexAbs);
      // Se comprueba si esta dirección es común a ambas direcciones
      fin = testPath.startsWith(comun);
   } while (fin == true);
   // La parte común de la dirección es desde el comienzo hasta el último carácter
   // separador
   comun = comun.substring(0, comun.lastIndexOf('/'));
   // Dirección absoluta del ítem menos la parte común con la dirección absoluta del test
   String rel = itemAbsoluta.substring(comun.length() + 1, itemAbsoluta.length());
   // Dirección absoluta del test menos la parte común con la dirección absoluta del ítem
   testPath = testPath.substring(comun.length() + 1, testPath.length());
   // Cálculo del número de separadores que hay desde la localización del test hasta
   // llegar a la dirección común con el ítem. Es el número de saltos de directorios
   // "hacia arriba" que hay que hacer para llegar al ítem desde el test
   int numDir = 0;
   for (int i = 0; i < testPath.length(); i++){
```

```
        if (testPath.charAt(i) == '/')
            numDir++;
    }
    // Dirección relativa del ítem respecto al test
    String relativa = "";
    // Primero se incluyen los caracteres de subir nivel en la estructura de directorios
    // para llegar a la parte común la dirección absoluta del ítem
    for (int i = 0; i < numDir; i++)
        relativa += "../";
    // Luego se añade la dirección del ítem menos la parte que tiene en común con el test
    relativa += rel;
    // Se crea una URI relativa a partir de la dirección relativa contenida en el String
    href = URI.create(relativa);
 }

 /**
  * Método que escribe este objeto <code>AssessmentItemRef</code> en el <code>XMLStreamWriter
  * </code> recibido como parámetro, según la norma QTI definida.
  *
  * @param   xsw                 xmlStreamWriter con el que escribir el elemento
  *                             correspondiente a esta clase en el archivo XML
  * @throws  XMLStreamException      si hay algún problema al escribir con el <code>
  *                             XMLStremaWriter</code> los datos XML en el archivo
  * @see     javax.xml.stream.XMLStreamWriter
  */
 public void creaAssessmentItemRef (XMLStreamWriter xsw) throws XMLStreamException {

    xsw.writeStartElement("assessmentItemRef");     // Comienzo del elemento
    // Identificador de la referencia al ítem
    xsw.writeAttribute("identifier", identificador.getIdentificador());
    xsw.writeAttribute("href", href.toString()); // URI de referencia al ítem
    if (requerido == true)  // El elemento requerido se escribe si el ítem es requerido
        xsw.writeAttribute("required", Boolean.toString(requerido));
    if (fijo == true)    // El elemento fijo se escribe si el ítem se queda fijo
        xsw.writeAttribute("fixed", Boolean.toString(fijo));
    ok.creaWeight(xsw);     // Se llama al método que escriba el peso de respuesta correcta
    noOk.creaWeight(xsw);  // Se llama al método que escriba el peso de respuesta incorrecta
    xsw.writeEndElement(); // Cierra el elemento "assessmentItemRef"
 }
}
```

AssessmentSection.java

```
package xml.test;

import java.io.File;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

import utilidades.Identificador;
import utilidades.ParserNeutro;

/**
 * Clase que representa a una sección del Test según la norma QTI del IMS. Se utiliza para
 * agrupar varios ítems dentro de un Test. Contiene varios atributos que se corresponden con
 * los atributos definidos en la norma, como son el título de la sección, si es visible, si se
 * sigue algún criterio de selección y ordenación, y el array de objetos <code>
 * AssessmentItemRef</code> con las referencias de los ítems incluidos en la sección. Tal y
 * como define la norma esta clase hereda de la clase "SectionPart", por lo tanto también
 * contendrá sus variables de clase.

 * @author David Domínguez
 *
 */
public class AssessmentSection extends SectionPart {

 /**
  * Título de la sección.
  */
 private String titulo;
 /**
  * Si la sección es visible para el candidato que realice el test.
  */
```

```
private boolean visible;
/**
 * Criterio de selección de los ítems posibles a seguir.
 */
private Selection seleccion;
/**
 * Criterio de ordenación de los ítems del test.
 */
private Ordering orden;
/**
 * Instrucciones del ítem. Es código XHTML.
 */
private String instrucciones;
/**
 * Array de referencias a los ítems que se incluyen en la sección.
 */
private AssessmentItemRef[] itemRefArray;

/**
 * Constructor que inicializa variables de clase. Inicializa las siguientes variables de
 * clase: <code>titulo</code>, <code>visible</code>, <code>seleccion</code>, <code>orden
 * </code>, e <code>instrucciones</code>, y llamando al constructor de la superclase con el
 * identificador e iniciando ambos atributos de sección requerida y fija a falso, ya que
 * esta implementación de la norma QTI del IMS sólo permite una sección dentro del test,
 * por lo que esos parámetros no tendrán en cuenta.
 *
 * @param   identificador     identificador de la sección
 * @param   titulo            string con el título de la sección
 * @param   visible           boolean indicando la visibilidad de la sección
 * @param   seleccion         selection con el criterio de selección de los ítems
 * @param   orden          ordering con el criterio de ordenación de los ítems
 * @param   instrucciones     string con las instrucciones XHTML
 */
public AssessmentSection(Identificador identificador, String titulo, boolean visible,
      Selection seleccion, Ordering orden, String instrucciones) {
   // Los parámetros requerido y fijo para la sección, como sólo hay una, no son
   // necesarios y no se tienen en cuenta
   super(identificador, false, false);
   this.titulo = titulo;
   this.visible = visible;
   this.seleccion = seleccion;
   this.orden = orden;
   this.instrucciones = instrucciones;
}

/**
 * Método "get" para obtener el array de objetos de referencia a los ítems, <code>
 * AssessmentItemRef</code>, de la sección.
 *
 * @return     array de objetos AssessmentItemRef de la sección
 * @see     #setItemRefArray
 */
public AssessmentItemRef[] getItemRefArray() {
   return itemRefArray;
}

/**
 * Método "set" para establecer el array de objetos de referencia a los ítems,<code>
 * AssessmentItemRef</code>, de la sección.
 *
 * @param   itemRefArray   array de objetos AssessmentItemRef de la sección
 * @see #getItemRefArray
 */
public void setItemRefArray(AssessmentItemRef[] itemRefArray) {
   this.itemRefArray = itemRefArray;
}

/**
 * Método "get" para obtener el criterio de ordenación de la sección.
 *
 * @return     ordering con el criterio de ordenación
 */
public Ordering getOrden() {
   return orden;
}
```

```
/**
 * Método "get" para obtener el criterio de selección de los ítems de la sección.
 *
 * @return     selection con el criterio de selección
 */
public Selection getSeleccion() {
   return seleccion;
}

/**
 * Calcula la URI relativa de todos los ítems de la sección. El método llama al método para
 * calcular la URI relativa de un ítem para cada uno de los objetos referencia de ítem de
 * la sección, <code>AssessmentItemRef</code>, pasándole el objeto <code>File</code>
 * representando la dirección en disco del Test.
 *
 * @param   referencia     file con la dirección final en disco del Test
 */
public void creaRefItems (File referencia){
   for (int i = 0; i < itemRefArray.length; i++)
      itemRefArray[i].creaItemRef(referencia);
}

/**
 * Método que escribe este objeto <code>AssessmentSection</code> en el <code>XMLStreamWriter
 * </code> recibido como parámetro, según la norma QTI definida.
 *
 * @param   xsw                 xmlStreamWriter con el que escribir el elemento
 *                             correspondiente a esta clase en el archivo XML
 * @throws  XMLStreamException      si hay algún problema al escribir con el <code>
 *                              XMLStremaWriter</code> los datos XML en el archivo
 * @see     javax.xml.stream.XMLStreamWriter
 */
public void creaAssessmentSection (XMLStreamWriter xsw) throws XMLStreamException{

   xsw.writeStartElement("assessmentSection");     // Comienzo del elemento
   xsw.writeAttribute("identifier", identificador.getIdentificador()); // Identificador
   xsw.writeAttribute("title", titulo);         // Título de la sección
   xsw.writeAttribute("visible", Boolean.toString(visible));  // Atributo visible
   if (seleccion != null)                       // Escribe la selección si la hay
      seleccion.creaSelection(xsw);
   if (orden != null)                           // Escribe el ordenación si lo hay
      orden.creaOrdering(xsw);
   if (instrucciones != null) {                 // Las instrucciones se escriben si hay
      xsw.writeStartElement("rubricBlock");     // Elemento para las instrucciones
      xsw.writeAttribute("view", "candidate"); // Las ve el candidato
      // Se escriben mediante el parser neutro ya que es código XHTML
      ParserNeutro.escribeInstruccionesXML(instrucciones, xsw);
      xsw.writeEndElement();                    // Cierra "rubricBlock"
   }
   for (int i = 0; i < itemRefArray.length; i++)
      // Escribe cada una de las referencias a los ítems
      itemRefArray[i].creaAssessmentItemRef(xsw);
   xsw.writeEndElement();                       // Cierra "assessmentSection"
}
}
```

AssessmentTest.java

```
package xml.test;

import java.io.File;
import java.io.IOException;
import java.io.StringWriter;

import utilidades.MensajeEstado;

import javax.servlet.ServletContext;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

import org.apache.commons.io.FileUtils;

/**
 * Clase que representa a un Test según la norma QTI del IMS. Contiene varios atributos que se
 * corresponden con los atributos definidos en la norma, como son el título, el identificador
```

```
 * del test, y las Partes del Test, <code>TestPart</code>, que incluye (en esta implementación
 * sólo puede incluir una Parte de Test en el Test).

 * @author David Domínguez
 */
public class AssessmentTest {

 /**
  * Identificador del Test, que será también el nombre del archivo XML que lo representa.
  */
 private String identificador;
 /**
  * Título del Test.
  */
 private String titulo;
 /**
  * Objeto TestPart que contiene el Test, donde está la sección con las referencias a los
  * ítems que incluye el Test.
  */
 private TestPart testPart;

 /**
  * Constructor de la clase que le da los valores iniciales al identificador, al título y a
  * la Parte de Test que incluye el Test.
  *
  * @param   identificador  string con el identificador del Test, y el nombre del archivo
  * @param   titulo         string con el título del Test
  * @param   testPart    testPart que incluye el Test
  */
 public AssessmentTest (String identificador, String titulo, TestPart testPart) {
    super ();
    this.identificador = identificador;
    this.titulo = titulo;
    this.testPart = testPart;
 }

 /**
  * Método "get" para obtener el identificador del Test.
  *
  * @return     string con el identificador del Test
  * @see #setIdentificador
  */
 public String getIdentificador() {
    return identificador;
 }

 /**
  * Método "get" para obtener la Parte del Test que incluye el Test.
  *
  * @return     testPart con la Parte del Test que incluye el Test
  */
 public TestPart getTestPart() {
    return testPart;
 }


 /**
  * Método "get" para obtener el título del Test.
  *
  * @return     string con el título del Test
  */
 public String getTitulo() {
    return titulo;
 }

 /**
  * Método "set" para establecer un nuevo identificador para el test. Utilizado para
  * establecer un nuevo identificador en el caso de que ya exista un test con el mismo
  * identificador en ese mismo directorio.
  *
  * @param   identificador  string con el nuevo identificador para el Test
  * @see     #getIdentificador
  */
 public void setIdentificador(String identificador) {
    this.identificador = identificador;
 }
```

```
/**
 * Método que escribe este objeto <code>AssessmentTest</code> en el directorio indicado por
 * el <code>String path</code>. Mediante el objeto <code>ServletContext</code> se accede al
 * valor de algunos parámetros inciales definidos en el archivo descriptor de despliegue de
 * la Aplicación, "web.xml". El objeto se escribe utilizando un <code>XMLStreamWriter
 * </code>.
 *
 * @param   path        string indicando la dirección del directorio en el que
 *                      almacenar el Test
 * @param   sc          servletContext de la Aplicación para acceder a parámetros
 *                      definidos en "web.xml"
 * @return              mensajeEstado indicando cómo ha ido el proceso
 * @throws IOException     si hay algún problema al escribir en disco
 * @see     javax.xml.stream.XMLStreamWriter
 */
public MensajeEstado creaAssessmentTest(String path, ServletContext sc) throws IOException{

   MensajeEstado msg = new MensajeEstado();     // Mensaje de salida del proceso

   try {
      XMLOutputFactory xof =  XMLOutputFactory.newInstance();
      XMLStreamWriter xsw = null;            // Writer para escribir datos XML
      StringWriter sw = new StringWriter();  // Buffer en el que escribir
      xsw = xof.createXMLStreamWriter(sw);   // El writer XML escribe en el buffer
      xsw.writeStartDocument("ISO-8859-1","1.0");     // Inicio del documento XML
      xsw.writeStartElement("assessmentTest");     // Comienzo del Test
      xsw.writeDefaultNamespace("http://www.imsglobal.org/xsd/imsqti_v2p1");
      xsw.writeNamespace("xsi", "http://www.w3.org/2001/XMLSchema-instance");
      xsw.setPrefix("xsi", "http://www.w3.org/2001/XMLSchema-instance");
      xsw.writeAttribute("http://www.w3.org/2001/XMLSchema-instance", "schemaLocation",
            "http://www.imsglobal.org/xsd/imsqti_v2p1 imsqti_v2p1.xsd");
      xsw.writeAttribute("identifier", identificador);   // Identificador del Test
      xsw.writeAttribute("title", titulo);           // Título del Test
      // Nombre y versión de la herramienta
      xsw.writeAttribute("toolName", sc.getInitParameter("nombre herramienta"));
      xsw.writeAttribute("toolVersion", sc.getInitParameter("versión herramienta"));

      xsw.writeStartElement("outcomeDeclaration"); // Variable de puntuación
      xsw.writeAttribute("identifier", "SCORE");      // Identificador de la puntuación
      xsw.writeAttribute("cardinality", "single"); // Cardinalidad
      xsw.writeAttribute("baseType", "float");     // Puntuación de tipo float
      xsw.writeStartElement("defaultValue");       // Valor por defecto 0.0
      xsw.writeStartElement("value");
      xsw.writeCharacters("0.0");
      xsw.writeEndElement();                    // Cierra "value"
      xsw.writeEndElement();                    // Cierra "defaultValue"
      xsw.writeEndElement();                    // Cierra "outcomeDeclaration"
      xsw.writeStartElement("outcomeDeclaration"); // Constante valor del ítem correcto
      xsw.writeAttribute("identifier", "CORRECTO");   // Identificador de la constante
      xsw.writeAttribute("cardinality", "single"); // Cardinalidad de la constante
      xsw.writeAttribute("baseType", "float");     // Tipo base float
      xsw.writeStartElement("defaultValue");       // Valor por defecto de la constante
      xsw.writeStartElement("value");              // Valor
      xsw.writeCharacters("1.0");                  // Constante de valor 1.0
      xsw.writeEndElement();                    // Cierra "value"
      xsw.writeEndElement();                    // Cierra "defaultValue"
      xsw.writeEndElement();                    // Cierra "outcomeDeclaration"
      xsw.writeStartElement("outcomeDeclaration");  // Constante valor del ítem incorrecto
      xsw.writeAttribute("identifier", "INCORRECTO"); // Identificador de la constante
      xsw.writeAttribute("cardinality", "single"); // Cardinalidad de la constante
      xsw.writeAttribute("baseType", "float");     // Tipo base float
      xsw.writeStartElement("defaultValue");       // Valor por defecto de la constante
      xsw.writeStartElement("value");              // Valor
      xsw.writeCharacters("-1.0");              // Constante de valor -1.0
      xsw.writeEndElement();                    // Cierra "value"
      xsw.writeEndElement();                    // Cierra "defaultValue"
      xsw.writeEndElement();                    // Cierra "outcomeDeclaration"

      // Se llama al método para escribir la Parte del Test, que contiene las secciones,
      // las referencias a los ítems, y demás parámetros
      testPart.creaTestPart(xsw);

      // Escribe el proceso de respuesta para la puntuación del Test
      xsw.writeStartElement("outcomeProcessing");
      // Array de referencias a los ítems con todos los que contiene el Test
```

```
    AssessmentItemRef[] items = testPart.getSeccion().getItemRefArray();
    // Para todos los ítems del Test
    for (int i = 0; i < items.length; i++){
       xsw.writeStartElement("outcomeCondition");      // Condición
       xsw.writeStartElement("outcomeIf");          // Si ()
       xsw.writeStartElement("equal");              // Son iguales
       xsw.writeAttribute("toleranceMode", "exact");  // Comparación exacta
       xsw.writeEmptyElement("variable");           // La variable
       // Identificador de la variable resultado del ítem
       xsw.writeAttribute("identifier",
             items[i].getIdentificador().getIdentificador() + ".SCORE");
       xsw.writeEmptyElement("variable");           // Y La variable definida arriba
       xsw.writeAttribute("identifier", "CORRECTO");  // Con el identificador CORRECTO
       xsw.writeStartElement("setOutcomeValue");    // Entonces establece el valor
       xsw.writeAttribute("identifier", "SCORE");      // De la variable SCORE igual a
       xsw.writeStartElement("sum");             // La suma de
       xsw.writeEmptyElement("variable");           // El valor de la variable
       xsw.writeAttribute("identifier", "SCORE");      // SCORE
       xsw.writeEmptyElement("variable");           // Y el de la variable
       // Identificador de la variable del ítem
       xsw.writeAttribute("identifier",
             items[i].getIdentificador().getIdentificador() + ".SCORE");
       xsw.writeAttribute("weightIdentifier", "WeightOK");     // Por su peso correcto
       xsw.writeEndElement();                    // Cierra "sum"
       xsw.writeEndElement();                    // Cierra "setOutcomeValue"
       xsw.writeEndElement();                    // Cierra "equal"
       xsw.writeEndElement();                    // Cierra "outcomeIf"

       // Si no se cumple la anterior condición: Si ()
       xsw.writeStartElement("outcomeElseIf");
       xsw.writeStartElement("equal");              // Son iguales
       xsw.writeAttribute("toleranceMode", "exact");  // Comparación exacta
       xsw.writeEmptyElement("variable");           // La variable
       // Identificador de la variable resultado del ítem
       xsw.writeAttribute("identifier",
             items[i].getIdentificador().getIdentificador() + ".SCORE");
       xsw.writeEmptyElement("variable");           // Y la variable definida arriba
       xsw.writeAttribute("identifier", "INCORRECTO");//Con el identificador INCORRECTO
       xsw.writeStartElement("setOutcomeValue");    // Entonces establece el valor
       xsw.writeAttribute("identifier", "SCORE");      // De la variable SCORE igual a
       xsw.writeStartElement("sum");             // La suma de
       xsw.writeEmptyElement("variable");           // El valor de la variable
       xsw.writeAttribute("identifier", "SCORE");      // SCORE
       xsw.writeEmptyElement("variable");           // Y el de la variable
       // Identificador de la variable del ítem
       xsw.writeAttribute("identifier",
             items[i].getIdentificador().getIdentificador() + ".SCORE");
       xsw.writeAttribute("weightIdentifier", "WeightNOK");   // Por su peso incorrecto
       xsw.writeEndElement();                    // Cierra "sum"
       xsw.writeEndElement();                    // Cierra "setOutcomeValue"
       xsw.writeEndElement();                    // Cierra "equal"
       xsw.writeEndElement();                    // Cierra "outcomeElseIf"
       xsw.writeEndElement();                    // Cierra "outcomeCondition"
    }


    xsw.writeEndElement();     // Cierra "outcomeProcessing"
    xsw.writeEndElement();     // Cierra "assessmentTest"
    xsw.writeEndDocument();    // Fin del documento
    xsw.flush();            // Vacía el Writer XML
    xsw.close();            // Cierra el Writer XML

    // Vacía y cierra el Writer
    sw.flush();
    sw.close();
    // Pasa a String el contenido del Writer
    String fich = sw.toString();
    // Fichero de salida
    File file = new File(path + identificador + ".xml");
    // Escribe en el fichero de salida
    FileUtils.writeStringToFile(file, fich, null);
 } catch (XMLStreamException e) {
    // Error en el procesamiento de los datos XML
    msg.setMensaje("Error procesando el archivo XML");
    msg.setEstado(false);
 }
```

```
    if (msg.isEstado()){
       // Si todo ha ido bien se informa del resultado
       msg.setMensaje("Test creado con éxito");
    }

    return msg;
 }
}
```

ItemSessionControl.java

```
package xml.test;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

/**
 * Clase que representa un objeto de control de la sesión al realizar el Test según la norma
 * QTI del IMS. Contiene sólo una variable, que se corresponde con el atributo "allowReview"
 * definido en la norma.
 *
 * @author David Domínguez
 *
 */
public class ItemSessionControl {

 /**
  * Permitir al candidato revisar la sección tras responderla con las respuestas
  * que ha dado.
  */
 boolean allowReview;

 /**
  * Constructor que establece el valor del único campo de la clase, <code>allowReview</code>.
  *
  * @param   allowReview    permitir o no revisar el examen tras responderlo
  */
 public ItemSessionControl (boolean allowReview) {
    this.allowReview = allowReview;
 }

 /**
  * Método que escribe este objeto <code>ItemSessionControl</code> en el <code>
  * XMLStreamWriter</code> recibido como parámetro, según la norma QTI definida.
  *
  * @param   xsw                 xmlStreamWriter con el que escribir el elemento
  *                           correspondiente a esta clase en el archivo XML
  * @throws  XMLStreamException      si hay algún problema al escribir con el <code>
  *                           XMLStremaWriter</code> los datos XML en el archivo
  * @see     javax.xml.stream.XMLStreamWriter
  */
 public void creaItemSessionControl(XMLStreamWriter xsw) throws XMLStreamException{

    xsw.writeEmptyElement("itemSessionControl"); // Escribe el elemento vacío
    xsw.writeAttribute("allowReview", Boolean.toString(allowReview)); // Escribe el Atributo
 }
}
```

Ordering.java

```
package xml.test;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

/**
 * Clase que representa el criterio de ordenación de los ítems de una sección de un Test según
 * la norma QTI del IMS. El único criterio definido por la norma es permitir mezclar los ítems
 * aleatoriamente o dejarlos en la posición definida en el Test. Contiene un atributo que se
 * corresponde con el definido en la norma.
 *
 * @author David Domínguez
```

```
 *
 */
public class Ordering {

 /**
  * Variable indicando si se deben mezclar aleatoriamente los ítems o no.
  */
 private boolean shuffle;

 /**
  * Constructor que simplemente inicializa la variable de la clase boolean.
  *
  * @param   shuffle     boolean indicando si se mezclan aleatoriamente los ítems
  */
 public Ordering (boolean shuffle) {
    this.shuffle = shuffle;
 }

 /**
  * Método que escribe este objeto <code>Ordering</code> en el <code>XMLStreamWriter</code>
  * recibido como parámetro, según la norma QTI definida.
  *
  * @param   xsw                 xmlStreamWriter con el que escribir el elemento
  *                            correspondiente a esta clase en el archivo XML
  * @throws  XMLStreamException      si hay algún problema al escribir con el <code>
  *                            XMLStremaWriter</code> los datos XML en el archivo
  * @see     javax.xml.stream.XMLStreamWriter
  */
 public void creaOrdering(XMLStreamWriter xsw) throws XMLStreamException{

    xsw.writeEmptyElement("ordering");     // Escribe el elemento
    xsw.writeAttribute("shuffle", Boolean.toString(shuffle));  // Atributo del elemento
 }
}
```

SectionPart.java

```
package xml.test;

import utilidades.Identificador;

/**
 * Clase que representa a la clase abstracta del mismo nombre según la norma QTI del IMS. Es la
 * superclase de otras clases que componen un Test. Contiene los atributos que se corresponden
 * con los definidos en la norma.
 *
 * @author David Domínguez
 *
 */
public abstract class SectionPart {

 /**
  * Identificador del elemento.
  */
 protected Identificador identificador;
 /**
  * boolean indicando si la subclase está requerida obligatoriamente en caso de que se use
  * algún criterio de selección.
  */
 protected boolean requerido;
 /**
  * boolean indicando si la subclase está fija en caso de que se use algún criterio de
  * ordenación.
  */
 protected boolean fijo;

 /**
  * Constructor de la clase que simplemente le da valores iniciales a las variables de clase.
  *
  * @param   identificador  identificador que contiene un identificador válido para la clase
  * @param   requerido      boolean indicando que este elemento es requerido
  * @param   fijo        boolean indicando que este elemento está en posición fija
  */
 public SectionPart(Identificador identificador, boolean requerido, boolean fijo) {
```

```
    this.identificador = identificador;
    this.requerido = requerido;
    this.fijo = fijo;
 }

 /**
  * Método "get" para obtener el identificador de la clase.
  *
  * @return     identificador que contiene el identificador válido de la clase
  */
 public Identificador getIdentificador() {
    return identificador;
 }
}
```

Selection.java

```
package xml.test;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

/**
 * Clase que representa el criterio de selección de los ítems de una sección de un Test según
 * la norma QTI del IMS. El único criterio definido por la norma es seleccionar un subconjunto
 * del conjunto total de ítems de la sección. Contiene unos atributos que se correspondes con
 * los definidos en la norma.
 *
 * @author David Domínguez
 *
 */
public class Selection {

 /**
  * Número de ítems a seleccionar del número total de ítems.
  */
 private int seleccion;
 /**
  * Si se pueden repetir los ítems escogiéndolos más de una vez o no.
  */
 private boolean conReemplazamiento;

 /**
  * Constructor de la clase que simplemente le da valores iniciales a las variables de la
  * clase.
  *
  * @param   seleccion         entero con el número de ítems a seleccionar del conjunto
  * @param   conReemplazamiento   boolean indicando si se puede escoger varias veces el mismo
  *                           ítem para formar el Test final
  */
 public Selection (int seleccion, boolean conReemplazamiento) {
    this.seleccion = seleccion;
    this.conReemplazamiento = conReemplazamiento;
 }

 /**
  * Método "get" que devuelve el número de ítems a seleccionar del conjunto de los ítems de
  * la sección.
  *
  * @return     entero indicando el número de ítems a seleccionar
  */
 public int getSeleccion() {
    return seleccion;
 }

 /**
  * Método que escribe este objeto <code>Selection</code> en el <code>XMLStreamWriter</code>
  * recibido como parámetro, según la norma QTI definida.
  *
  * @param   xsw                  xmlStreamWriter con el que escribir el elemento
  *                           correspondiente a esta clase en el archivo XML
  * @throws  XMLStreamException      si hay algún problema al escribir con el <code>
  *                           XMLStremaWriter</code> los datos XML en el archivo
  * @see     javax.xml.stream.XMLStreamWriter
```

```
  */
 public void creaSelection (XMLStreamWriter xsw) throws XMLStreamException{
    xsw.writeEmptyElement("selection ");      // Escribe el elemento
    xsw.writeAttribute("select", Integer.toString(seleccion)); // Atributo selección
    // Atributo con reemplazamiento
    xsw.writeAttribute("withReplacement", Boolean.toString(conReemplazamiento));
 }
}
```

TestPart.java

```
package xml.test;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

import utilidades.Identificador;

/**
 * Clase que representa una Parte de un Test según la norma QTI del IMS. Se utiliza para
 * agrupar Secciones. Contiene unos atributos que se corresponden con los definidos en la
norma.
 *
 * @author David Domínguez
 *
 */
public class TestPart {

 /**
  * Identificador de la Parte del Test.
  */
 private Identificador identificador;
 /**
  * Modo de navegación por la Parte del Test.
  */
 private String navegacion;
 /**
  * Modo de presentación de la Parte del Test.
  */
 private String presentacion;
 /**
  * Objeto de control de la Parte del Test.
  */
 private ItemSessionControl control;
 /**
  * Sección que incluye esta Parte del Test. Esta implementación sólo soporta una Sección por
  * Parte de Test.
  */
 private AssessmentSection seccion;

 /**
  * Constructor que simplemente inicializa las variables de la clase.
  *
  * @param   identificador  identificador con un identificador válido
  * @param   navegacion     string con el modo de navegación
  * @param   presentacion   string con el modo de presentación
  * @param   control        itemSessionControl con el control de la sesión
  * @param   seccion        assessmentSection con la sección que incluye
  */
 public TestPart(Identificador identificador, String navegacion, String presentacion,
       ItemSessionControl control, AssessmentSection seccion) {
    this.identificador = identificador;
    this.navegacion = navegacion;
    this.presentacion = presentacion;
    this.control = control;
    this.seccion = seccion;
 }

 /**
  * Método "get" para obtener la Sección, el objeto <code>AssessmentSection</code> que
  * incluye esta Parte del Test.
  *
  * @return              assessmentSection con la Sección
  */
```

```
 public AssessmentSection getSeccion() {
    return seccion;
 }

 /**
  * Método que escribe este objeto <code>TestPart</code> en el <code>XMLStreamWriter</code>
  * recibido como parámetro, según la norma QTI definida.
  *
  * @param   xsw                 xmlStreamWriter con el que escribir el elemento
  *                             correspondiente a esta clase en el archivo XML
  * @throws  XMLStreamException      si hay algún problema al escribir con el <code>
  *                             XMLStremaWriter</code> los datos XML en el archivo
  * @see     javax.xml.stream.XMLStreamWriter
  */
 public void creaTestPart(XMLStreamWriter xsw) throws XMLStreamException{
    xsw.writeStartElement("testPart");              // Escribe el elemento
    xsw.writeAttribute("identifier", identificador.getIdentificador()); // Identificador
    xsw.writeAttribute("navigationMode", navegacion); // Modo de navegación
    xsw.writeAttribute("submissionMode", presentacion);  // Modo de presentación
    control.creaItemSessionControl(xsw);            // Escribe el control de la sesión
    seccion.creaAssessmentSection(xsw);             // Escribe la sección
    xsw.writeEndElement();                        // Cierra "testPart"
 }
}
```

Weight.java

```
package xml.test;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

import utilidades.Identificador;

/**
 * Clase que representa un Peso de un ítem según la norma QTI del IMS. Se utiliza para aplicar
 * pesos distintos a los distintos ítems que componen el Test a la hora de evaluarlos. Contiene
 * unos atributos que se corresponden con los definidos en la norma.
 *
 * @author David Domínguez
 *
 */
public class Weight {

 /**
  * Identificador válido del peso.
  */
 private Identificador identificador;
 /**
  * Valor del peso que tiene el ítem dentro del Test.
  */
 private double peso;

 /**
  * Constructor que simplemente le da valores iniciales a las variables de la clase.
  *
  * @param   identificador  identificador con el identificador válido de la clase
  * @param   peso        double con el valor del peso del ítem en el Test
  */
 public Weight (Identificador identificador, double peso) {
    this.identificador = identificador;
    this.peso = peso;
 }

 /**
  * Método que escribe este objeto <code>Weight</code> en el <code>XMLStreamWriter</code>
  * recibido como parámetro, según la norma QTI definida.
  *
  * @param   xsw                 xmlStreamWriter con el que escribir el elemento
  *                             correspondiente a esta clase en el archivo XML
  * @throws  XMLStreamException      si hay algún problema al escribir con el <code>
  *                             XMLStremaWriter</code> los datos XML en el archivo
  * @see     javax.xml.stream.XMLStreamWriter
  */
```

```
 public void creaWeight (XMLStreamWriter xsw) throws XMLStreamException{
    xsw.writeStartElement("weight");          // Escribe el elemento
    xsw.writeAttribute("identifier", identificador.getIdentificador()); // Identificador
    xsw.writeCharacters(Double.toString(peso));     // Valor del peso
    xsw.writeEndElement();                    // Cierra "weight"
 }
}
```

3. Páginas Web JSP

A continuación se muestra el código de las JSPs que componen la Aplicación Web. Están compuestas de código HTML, etiquetas JSP (estándar y personalizadas) y llamadas a funciones de Javascript.

buscadorimagenes.jsp

```
<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Explorador de Im&aacute;genes</title>
 </head>

 <body bgcolor="#C3D3DF">

    <!-- Frame izquierdo donde se muestran las imágenes y los directorios disponibles -->
    <iframe name="Explorador" src="seleccionasignatura.jsp?Destino=explorador.jsp" align="left"
marginwidth="1%" marginheight="2%" width="33%" height="96%">
       El explorador no admite los marcos flotantes o no est&aacute; configurado actualmente
para mostrarlos.
    </iframe>

    <!-- Frame derecho donde se precargan las imágenes del frame izquierdo -->
    <iframe name="Imagen" src="imagen.jsp" align="right" marginwidth="1%" marginheight="2%"
width="65%" height="96%">
       El explorador no admite los marcos flotantes o no est&aacute; configurado actualmente
para mostrarlos.
    </iframe>

    <div style="position:absolute; top:95%; left:0; width:100%; height:3%">
       <center>
          <small>Herramienta de creaci&oacute;n de Examen QTI. Universidad de Sevilla.</small>
       </center>
    </div>
 </body>
</html>
```

cabecera.jsp

```
    <center>
       <a href="index.jsp"><img src="cabecera.jpg" alt="Volver al Inicio"/></a>
```

choice.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Tipo "Choice"</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/whizzywig.js"></script>
```

```
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/espanol.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/xhtml.js"></script>
    <script language="JavaScript" type="text/javascript" src="./Scripts/popup.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>
 <body onLoad="document.Formulario.${Focus}.focus();" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Tipo "Choice"</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <form method="post" onsubmit="syncTextarea();" name="Formulario">
       <p>Identificador*: <input type="text" name="Identif" size="26" value="${Identif}"></p>
       <p>T&iacute;tulo*: <input type="text" name="Tit" size="34" value="${Tit}"></p>
       <p><a href="seleccionasignatura.jsp?Destino=fileupload.jsp"
             onclick="return
popup('seleccionasignatura.jsp?Destino=fileupload.jsp','fileupload','height=400,width=460,left=
100,top=150,scrollbars = yes')">
             Subir Imagen (bmp, jpeg, png o gif)
       </a></p>
       <p>Instrucciones:</p>
       <p>
          <textarea name="Instruc" id="Instruc" cols="100" rows="20" style="width:1024px;
height:300px;">${Instruc}</textarea>
          <x:whizzywig textarea="Instruc"/>
       </p>
       <p>Pregunta: <input type="text" size="100" name="Preg" value="${Preg}"></p>
       <p>N&uacute;mero de respuestas*: <input type="text" name="Num_resp" size="8"
value="${Num_resp}"></p>
       <p>&#191;Mezclar aleatoriamente las respuestas? <input type="checkbox" name="Shuffle"
<c:if test="${Shuffle == true}">checked="true"</c:if>></p>
       <p><input type="submit" value="Introducir Respuestas" name="Form"/></a>
       <input type="button" value="Restablecer"
             onClick="resetValues(Formulario, 'Identif', '', '', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

creartest.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Nuevo Test</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/whizzywig.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/espanol.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/xhtml.js"></script>
    <script language="JavaScript" type="text/javascript" src="./Scripts/popup.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p align="center"><font size="5">Crear Nuevo Test</font></p>

    <x:aviso aviso="${Aviso}"/>
```

```
    <form method="post" onsubmit="syncTextarea();" name="Formulario">
       <p>Identificador del Test*: <input type="text" name="Identif" size="26"
value="${Identif}"></p>
       <p>T&iacute;tulo del test*: <input type="text" name="Tit" size="34" value="${Tit}"></p>
       <p><a href="seleccionasignatura.jsp?Destino=fileupload.jsp"
             onclick="return
popup('seleccionasignatura.jsp?Destino=fileupload.jsp','fileupload','height=400,width=460,left=
100,top=150,scrollbars = yes')">
             Subir Imagen (bmp, jpeg, png o gif)
       </a></p>
       <p>Instrucciones:</p>
       <p>
          <textarea name="Instruc" id="Instruc" cols="100" rows="20" style="width:1024px;
height:300px">${Instruc}</textarea>
          <x:whizzywig textarea="Instruc"/>
       </p>
       <p>Permitir revisar las respuestas del test al finalizar*:<br />
          <input type="radio" name="allowReview" <c:if test="${allowReview ==
true}">checked="true"</c:if> value="true"> <strong>S&iacute;</strong>, revisar el test tras
responderlo<br />
          <input type="radio" name="allowReview" <c:if test="${allowReview ==
false}">checked="false"</c:if> value="false"> <strong>No</strong>, no revistar el test tras
responderlo
       </p>
       <p>N&uacute;mero de &iacute;tems a seleccionar del conjunto: <input type="text"
name="Select" size="6" value="${Select}"><br/>
       En blanco o con un cero se seleccionan todos los &iacute;tems</p>
       <p>&#191;Mezclar aleatoriamente los &iacute;tems? <input type="checkbox" name="Shuffle"
<c:if test="${Shuffle == true}">checked="true"</c:if>></p>
       <p><input type="submit" value="Seleccionar &Iacute;tems para el Test" name="Form"/></a>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'Identif', '',
'', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

errorarchivonoencontrado.jsp

```
<%@ page isErrorPage="true" %>
<%@ taglib prefix="util" uri="http://www.QTI.us.es/util"%>

<html>
 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>¡Página no encontrada!</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <util:logger nivel="INFO" mensaje="Código de error HTTP 404" />
    <h3>Página no encontrada en el servidor</h3>
    La petición que ha realizado, <util:reqURI />, no existe en este servidor.<br />

    Si el error persiste envíe una descripción a
    <a href="mailto:<util:adminMail />?subject=Error creando examen QTI"><util:adminMail
/></a>,
    indicando que la p&aacute;gina que produjo el error fue "<util:reqURI />".
    <p align="center"><a href="index.jsp">Volver al Inicio</a></p>

 <jsp:include page="pie.jsp"/>
```

errorfinsesion.jsp

```
<%@ page isErrorPage="true" %>
<%@ taglib prefix="util" uri="http://www.QTI.us.es/util"%>
```

```
<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>¡La sesión ha expirado!</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <util:logger nivel="INFO" mensaje="Fin de sesión de usuario" />

    <h3>Su sesión ha expirado</h3>
    Su sesión en el sistema ha finalizado debido a que se ha excedido el tiempo de
    espera o a algún fallo interno en el sistema. <br />

    Si el error persiste envíe una descripción a
    <a href="mailto:<util:adminMail />?subject=Error creando examen QTI"><util:adminMail
/></a>.
    <p align="center"><a href="index.jsp">Volver al Inicio</a></p>

 <jsp:include page="pie.jsp"/>
```

errorgeneral.jsp

```
<%@ page isErrorPage="true" %>
<%@ taglib prefix="util" uri="http://www.QTI.us.es/util"%>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>&iexcl;Error inesperado!</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <util:logger nivel="SEVERE" mensaje="Excepción grave general lanzada" />

    <h3>Ha ocurrido un error inesperado</h3>
    Ha ocurrido un error inesperado de origen desconocido.<br />

    Si el error persiste env&iacute;e una descripci&oacute;n a
    <a href="mailto:<util:adminMail />?subject=Error creando examen QTI"><util:adminMail
/></a>,
    indicando que la p&aacute;gina que produjo el error fue <util:reqURI />.
    <p align="center"><a href="index.jsp">Volver al Inicio</a></p>

 <jsp:include page="pie.jsp"/>
```

errorio.jsp

```
<%@ page isErrorPage="true" %>
<%@ taglib prefix="util" uri="http://www.QTI.us.es/util"%>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>¡Error de Entrada/Salida!</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <util:logger nivel="SEVERE" mensaje="Excepción grave de I/O lanzada" />

    <h3>Ha ocurrido un error en el sistema de Entrada/Salida</h3>
```

```
    Ha ocurrido un error inesperado producido por operaciones de Entrada/Salida
    inesperadas o err&oacute;neas.<br />

    Si el error persiste env&iacute;e una descripci&oacute;n a
    <a href="mailto:<util:adminMail />?subject=Error creando examen QTI"><util:adminMail
/></a>,
    indicando que la p&aacute;gina que produjo el error fue <util:reqURI />.
    <p align="center"><a href="index.jsp">Volver al Inicio</a></p>

 <jsp:include page="pie.jsp"/>
```

errorjsp.jsp

```
<%@ page isErrorPage="true" %>
<%@ taglib prefix="util" uri="http://www.QTI.us.es/util"%>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>¡Error en la JSP!</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <util:logger nivel="SEVERE" mensaje="Excepción grave de JSP lanzada" />

    <h3>Ha ocurrido un error en la JSP</h3>
    Ha ocurrido un error inesperado producido por operaciones falladas en la JSP.<br />

    Si el error persiste env&iacute;e una descripci&oacute;n a
    <a href="mailto:<util:adminMail />?subject=Error creando examen QTI"><util:adminMail
/></a>,
    indicando que la p&aacute;gina que produjo el error fue <util:reqURI />.
    <p align="center"><a href="index.jsp">Volver al Inicio</a></p>

 <jsp:include page="pie.jsp"/>
```

errorservlet.jsp

```
<%@ page isErrorPage="true" %>
<%@ taglib prefix="util" uri="http://www.QTI.us.es/util"%>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>¡Error de Servlet!</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <util:logger nivel="SEVERE" mensaje="Excepción grave  de Servlet lanzada" />

    <h3>Ha ocurrido un error en las aplicaciones del Servidor</h3>
    Ha ocurrido un error inesperado producido por operaciones falladas en el servidor.<br />

    Si el error persiste env&iacute;e una descripci&oacute;n a
    <a href="mailto:<util:adminMail />?subject=Error creando examen QTI"><util:adminMail
/></a>,
    indicando que la p&aacute;gina que produjo el error fue <util:reqURI />.
    <p align="center"><a href="index.jsp">Volver al Inicio</a></p>

 <jsp:include page="pie.jsp"/>
```

explorador.jsp

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

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <script language="JavaScript" type="text/javascript" src="./Scripts/imagen.js"></script>
 </head>

 <body onload="cambiaImagen('blank.jpg', 'imagen','Imagen');" bgcolor="#ACC0C6">
    <p><font size="4">${DirectorioActual}:</font></p>

    <!-- Enlace para volver al directorio superior. Se muestra sólo si el enlace es distinto
    de null, lo que indica que no se puede subir un directorio. -->
    <c:if test="${DirectorioSuperior != null}">
       <p><img src="dir.png"/><a
href="explorador.jsp?DirectorioActual=${DirectorioSuperior}">/..</a></p>
    </c:if>

    <!-- Lista los directorios del directorio actual -->
    <c:forEach var="dir" items="${NombresDirectorios}">
       <p><img src="dir.png"/><a
href="explorador.jsp?DirectorioActual=${DirectorioActual}/${dir}">/${dir}</a></p>
    </c:forEach>

    <!-- Lista los archivos de imagen del directorio actual -->
    <c:forEach var="nombre" items="${NombresImagenes}" varStatus="estado">
       <p><img src="image.png"/><a href=""
       onMouseOver="cambiaImagen('${URLImagenes}${NombresImagenesCodif[estado.index]}',
'imagen','Imagen');"

 onclick="WantThis('${URLImagenes}${NombresImagenesCodif[estado.index]}');">${nombre}</a></p>
    </c:forEach>

    <p><a href="seleccionasignatura.jsp?Destino=explorador.jsp">Cambiar de Asignatura</a></p>

    <p><a href="javascript:window.parent.close()">Cerrar ventana</a></p>

 </body>
</html>
```

fileupload.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>File Upload</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.Archivo.focus()" bgcolor="#C3D3DF">

    <font color="#008000" size="4">${Informacion}<!-- Mensajes de información al usuario en la
página --></font>

    <x:aviso aviso="${Aviso}"/>

    <p><font size="4">Directorio Actual: ${DirectorioActual}</font></p>

    <!-- Enlace para volver al directorio superior. Se muestra sólo si el enlace es distinto
    de null, lo que indica que no se puede subir un directorio. -->
    <c:if test="${DirectorioSuperior != null}">
       <p><img src="dir.png"/><a
href="fileupload.jsp?DirectorioActual=${DirectorioSuperior}">/..</a></p>
    </c:if>
```

```
    <!-- Lista los directorios del directorio actual -->
    <c:forEach var="dir" items="${NombresDirectorios}">
       <p><img src="dir.png"/><a
href="fileupload.jsp?DirectorioActual=${DirectorioActual}/${dir}">/${dir}</a></p>
    </c:forEach>

    <p><a
href="nuevodirectorio.jsp?DirectorioActual=${DirectorioActual}&VolverA=fileupload.jsp">Crear
Nuevo Directorio</a></p>

    <br />

    <form method="post" enctype="multipart/form-data" name="Formulario">
       <input type="file" name="Archivo" size="50">
       <p><input type="submit" value="Subir Imagen"/>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, '', '', '',
'')"/></p>
    </form>

    <p><a href="seleccionasignatura.jsp?Destino=fileupload.jsp">Cambiar de Asignatura</a></p>

    <p><a href="javascript:self.close()">Cerrar ventana</a></p>

 <jsp:include page="pie.jsp"/>
```

fin.jsp

```
<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>${Titulo}</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p align="center">
       <font color="#008000" size="4">${Informacion}<!-- Mensajes de información al usuario en
la página --></font>
       <font color="#FF0000" size="4">${Aviso}<!-- Mensajes de aviso de error al usuario en la
página --></font>
    </p>
    <p align="center"><a href="index.jsp">INICIO</a></p>
 <jsp:include page="pie.jsp"/>
```

finitem.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Seleccione directorio</title>
 </head>

 <body onLoad="document.Formulario.Identif.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><font color="#008000" size="4">${Informacion}<!-- Mensajes de información al usuario en
la página --></font></p>

    <x:aviso aviso="${Aviso}"/>

    <p><font size="4">Directorio Actual: ${DirectorioActual}</font></p>
```

```
    <!-- Enlace para volver al directorio superior. Se muestra sólo si el enlace es distinto
    de null, lo que indica que no se puede subir un directorio. -->
    <c:if test="${DirectorioSuperior != null}">
       <p><img src="dir.png"/><a
href="finitem.jsp?DirectorioActual=${DirectorioSuperior}">/..</a></p>
    </c:if>

    <!-- Lista los directorios del directorio actual -->
    <c:forEach var="dir" items="${NombresDirectorios}">
       <p><img src="dir.png"/><a
href="finitem.jsp?DirectorioActual=${DirectorioActual}/${dir}">/${dir}</a></p>
    </c:forEach>

    <p><a
href="nuevodirectorio.jsp?DirectorioActual=${DirectorioActual}&VolverA=finitem.jsp">Crear Nuevo
Directorio</a></p>

    <br />

    <form method="post" name="Formulario">
       <!-- El campo para introducir un nuevo identificador del ítem se muestra si ya existe
       un ítem con el mismo identificador que ya se introdujo  -->
       <c:if test="${Nuevo == true}">
          <p>Introduzca el nuevo Nombre de archivo: <input type="text" name="Identif"
size="26"></p>
       </c:if>

       <p><input type="submit" value="Crear &Iacute;tem XML" name="Form"></p>
    </form>

    <p><a href="seleccionasignatura.jsp?Destino=finitem.jsp">Cambiar de Asignatura</a></p>

 <jsp:include page="pie.jsp"/>
```

fintest.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Seleccione directorio</title>
 </head>

 <body onLoad="document.Formulario.Identif.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><font color="#008000" size="4">${Informacion}<!-- Mensajes de información al usuario en
la página --></font></p>

    <x:aviso aviso="${Aviso}"/>

    <p><font size="4">Directorio Actual: ${DirectorioActual}</font></p>

    <!-- Enlace para volver al directorio superior. Se muestra sólo si el enlace es distinto
    de null, lo que indica que no se puede subir un directorio. -->
    <c:if test="${DirectorioSuperior != null}">
       <p><img src="dir.png"/><a
href="fintest.jsp?DirectorioActual=${DirectorioSuperior}">/..</a></p>
    </c:if>

    <!-- Lista los directorios del directorio actual -->
    <c:forEach var="dir" items="${NombresDirectorios}">
       <p><img src="dir.png"/><a
href="fintest.jsp?DirectorioActual=${DirectorioActual}/${dir}">/${dir}</a></p>
    </c:forEach>

    <p><a
href="nuevodirectorio.jsp?DirectorioActual=${DirectorioActual}&VolverA=fintest.jsp">Crear Nuevo
Directorio</a></p>
```

```
    <br />

    <form method="post" name="Formulario">
       <!-- El campo para introducir un nuevo identificador del ítem se muestra si ya existe
       un ítem con el mismo identificador que ya se introdujo  -->
       <c:if test="${Nuevo == true}">
          <p>Introduzca el nuevo Nombre de archivo: <input type="text" name="Identif"
size="26"></p>
       </c:if>

       <p><input type="submit" value="Crear Test" name="Form"></p>
    </form>

    <p><a href="seleccionasignatura.jsp?Destino=fintest.jsp">Cambiar de Asignatura</a></p>

 <jsp:include page="pie.jsp"/>
```

gapmatch.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Tipo "Gap Match"</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/whizzywig.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/espanol.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/xhtml.js"></script>
    <script language="JavaScript" type="text/javascript" src="./Scripts/popup.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Tipo "Gap Match"</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <form method="post" onsubmit="syncTextarea();" name="Formulario">
       <p>Identificador*: <input type="text" name="Identif" size="26" value="${Identif}"></p>
       <p>T&iacute;tulo*: <input type="text" name="Tit" size="34" value="${Tit}"></p>
       <p><a href="seleccionasignatura.jsp?Destino=fileupload.jsp"
             onclick="return
popup('seleccionasignatura.jsp?Destino=fileupload.jsp','fileupload','height=400,width=460,left=
100,top=150,scrollbars = yes')">
             Subir Imagen (bmp, jpeg, png o gif)
       </a></p>
       <p>Instrucciones:</p>
       <p>
          <textarea name="Instruc" id="Instruc" cols="100" rows="20" style="width:1024px;
height:300px">${Instruc}</textarea>
          <x:whizzywig textarea="Instruc"/>
       </p>
       <p>Pregunta:<input type="text" size="100" name="Preg" value="${Preg}"></p>
       <p>N&uacute;mero de opciones*: <input type="text" name="Num_opc" size="8"
value="${Num opc}"></p>
       <p>N&uacute;mero de huecos*: <input type="text" name="Num_huec" size="8"
value="${Num_huec}"></p>
       <p>&#191;Mezclar aleatoriamente las opciones? <input type="checkbox" name="Shuffle"
<c:if test="${Shuffle == true}">checked="true"</c:if>></p>
       <p><input type="submit" value="Introducir Respuestas" name="Form"/></a>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'Identif', '',
'', '')"/></p>
    </form>
```

```
    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

hottext.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Tipo "Hot Text"</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/whizzywig.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/espanol.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/xhtml.js"></script>
    <script language="JavaScript" type="text/javascript" src="./Scripts/popup.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Tipo "Hot Text"</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <form method="post" onsubmit="syncTextarea();" name="Formulario">
       <p>Identificador*: <input type="text" name="Identif" size="26" value="${Identif}"></p>
       <p>T&iacute;tulo*: <input type="text" name="Tit" size="34" value="${Tit}"></p>
       <p><a href="seleccionasignatura.jsp?Destino=fileupload.jsp"
             onclick="return
popup('seleccionasignatura.jsp?Destino=fileupload.jsp','fileupload','height=400,width=460,left=
100,top=150,scrollbars = yes')">
             Subir Imagen (bmp, jpeg, png o gif)
       </a></p>
       <p>Instrucciones:</p>
       <p>
          <textarea name="Instruc" id="Instruc" cols="100" rows="20" style="width:1024px;
height:300px">${Instruc}</textarea>
          <x:whizzywig textarea="Instruc"/>
       </p>
       <p>Pregunta:<input type="text" size="100" name="Preg" value="${Preg}"></p>
       <p>N&uacute;mero de Hot Texts*: <input type="text" name="Num_hot" size="8"
value="${Num_hot}"></p>
       <p><input type="submit" value="Introducir Respuestas" name="Form"/></a>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'Identif', '',
'', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

imagen.jsp

```
<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
    <meta http-equiv="Content-Language" content="es">
 </head>

 <body>
```

```
       <img src="blank.jpg" name="imagen" align="left"/>
 </body>

</html>
```

index.jsp

```
<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Crear Examen QTI</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p align="center">
       <a href="creartest.jsp"><font size="6">Crear Nuevo Test</font></a>
    </p>

    <p align="center">
       <a href="tipoitem.jsp"><font size="6">Crear Nuevo Item</font></a>
    </p>

 <jsp:include page="pie.jsp"/>
```

inlinechoice.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Tipo "Inline Choice"</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/whizzywig.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/espanol.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/xhtml.js"></script>
    <script language="JavaScript" type="text/javascript" src="./Scripts/popup.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Tipo "Inline Choice"</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <form method="post" onsubmit="syncTextarea();" name="Formulario">
       <p>Identificador*: <input type="text" name="Identif" size="26" value="${Identif}"></p>
       <p>T&iacute;tulo*: <input type="text" name="Tit" size="34" value="${Tit}"></p>
       <p><a href="seleccionasignatura.jsp?Destino=fileupload.jsp"
             onclick="return
popup('seleccionasignatura.jsp?Destino=fileupload.jsp','fileupload','height=400,width=460,left=
100,top=150,scrollbars = yes')">
             Subir Imagen (bmp, jpeg, png o gif)
       </a></p>
       <p>Instrucciones:</p>
       <p>
          <textarea name="Instruc" id="Instruc" cols="100" rows="20" style="width:1024px;
height:300px">${Instruc}</textarea>
```

```
          <x:whizzywig textarea="Instruc"/>
       </p>
       <p>N&uacute;mero de opciones posibles*: <input type="text" name="Num_opc" size="8"
value="${Num_opc}"></p>
       <p>&#191;Mezclar aleatoriamente las opciones? <input type="checkbox" name="Shuffle"
<c:if test="${Shuffle == true}">checked="true"</c:if>></p>
       <p><input type="submit" value="Introducir Opciones" name="Form"/></a>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'Identif', '',
'', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

introducirpesos.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Introduzca los pesos de los &Iacute;tems</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/confirmSubmit.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Introduzca los pesos de cada ítem</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <p>Los pesos en caso de ítem correcto que se dejen vacíos se considerarán igual a 1.0 y los
    de ítem incorrecto vacíos igual a 0.0</p>

    <form method="post" name="Formulario">
          <table border="1">
             <tr>
                <td width="230px" bgcolor="#ACC0C6"><p align="center"><b>Item</b></p></td>
                <td width="160px" bgcolor="#ACC0C6"><p align="center"><b>Título</b></p></td>
                <td width="120px" bgcolor="#ACC0C6"><p
align="center"><b>Identificador</b></p></td>
                <td width="120px" bgcolor="#ACC0C6"><p align="center"><b>Peso
Correcto</b></p></td>
                <td width="120px" bgcolor="#ACC0C6"><p align="center"><b>Peso
Incorrecto</b></p></td>
                <c:if test="${Shuffle == true}">
                   <td width="120px" bgcolor="#ACC0C6"><p align="center"><b>Fija</b></p></td>
                </c:if>
                <c:if test="${Select == true}">
                   <td width="120px" bgcolor="#ACC0C6"><p
align="center"><b>Requerido</b></p></td>
                </c:if>
             </tr>

             <!-- Una fila de la tabla por cada ítem añadido al test -->
             <c:forEach var="item" items="${Items_A_Añadir}"  varStatus="estado">
                <tr>
                   <td><p align="center">${item.nombre}</p></td>
                   <td><p align="center">${item.titulo}</p></td>
                   <td><p align="center"><input type="text" name="Identif_${estado.index}"
size="10" value="${Identif[estado.index].identificador}"></p></td>
                   <td><p align="center"><input type="text" name="Ok_${estado.index}" size="3"
value="${Ok[estado.index]}"></p></td>
                   <td><p align="center"><input type="text" name="NoOk_${estado.index}" size="3"
value="${NoOk[estado.index]}"></p></td>
```

```
                  <c:if test="${Shuffle == true}">
                     <td><p align="center"><input type="checkbox" name="Fija_${estado.index}"
<c:if test="${Fija[estado.index] == true}">checked="true"</c:if>></p></td>
                  </c:if>
                  <c:if test="${Select == true}">
                     <td><p align="center"><input type="checkbox"
name="Requerido ${estado.index}" <c:if test="${Requerido[estado.index] ==
true}">checked="true"</c:if>></p></td>
                  </c:if>
               </tr>
            </c:forEach>
         </table>

         <input type="submit" value="Crear Test" name="Crear" onClick="return
confirmSubmit('¿Está conforme a los pesos que ha introducido?')"/>
         <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'Identif_',
'${IdentificadorBase}', '', '')"/></p>

   </form>


 <jsp:include page="pie.jsp"/>
```

match.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
   <meta http-equiv="Content-Language" content="es"/>
   <title>Tipo "Match"</title>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/whizzywig.js"></script>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/espanol.js"></script>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/xhtml.js"></script>
   <script language="JavaScript" type="text/javascript" src="./Scripts/popup.js"></script>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

   <jsp:include page="cabecera.jsp"/>

   <p><h2>Tipo "Match"</h2></p>

   <x:aviso aviso="${Aviso}"/>

   <form method="post" onsubmit="syncTextarea();" name="Formulario">
      <p>Identificador*: <input type="text" name="Identif" size="26" value="${Identif}"></p>
      <p>T&iacute;tulo*: <input type="text" name="Tit" size="34" value="${Tit}"></p>
      <p><a href="seleccionasignatura.jsp?Destino=fileupload.jsp"
            onclick="return
popup('seleccionasignatura.jsp?Destino=fileupload.jsp','fileupload','height=400,width=460,left=
100,top=150,scrollbars = yes')">
            Subir Imagen (bmp, jpeg, png o gif)
      </a></p>
      <p>Instrucciones:</p>
      <p>
         <textarea name="Instruc" id="Instruc" cols="100" rows="20" style="width:1024px;
height:300px">${Instruc}</textarea>
         <x:whizzywig textarea="Instruc"/>
      </p>
      <p>Pregunta:<input type="text" size="100" name="Preg" value="${Preg}"></p>
      <p>N&uacute;mero de opciones del grupo 1 (filas)*: <input type="text" name="NumGrupo1"
size="8" value="${NumGrupo1}"></p>
      <p>N&uacute;mero de opciones del grupo 2 (columnas)*: <input type="text"
name="NumGrupo2" size="8" value="${NumGrupo2}"></p>
```

```
       <p>&#191;Mezclar aleatoriamente las opciones? <input type="checkbox" name="Shuffle"
<c:if test="${Shuffle == true}">checked="true"</c:if>></p>
       <p><input type="submit" value="Introducir Opciones" name="Form"/></a>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'Identif', '',
'', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

nuevodirectorio.jsp

```
<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Crear Nuevo Directorio</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.DirNombre.focus()" bgcolor="#C3D3DF">

    <p>Creando directorio en: ${DirectorioActual}
    <form method="POST" name="Formulario">
       <p>Nombre: <input type="text" name="DirNombre" size="43"></p>
       <p><input type="submit" value="Crear">
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, '', '', '',
'')"/></p>
       <p><a href="${VolverA}">Cancelar</a></p>
    </form>

 </body>

</html>
```

pie.jsp

```
       <small>Herramienta de Creaci&oacute;n de Examen QTI. Universidad de Sevilla.</small>
    </center>
 </body>
</html>
```

respuestaschoice.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Introducir respuestas tipo "Choice"</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/confirmSubmit.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Introducir respuestas tipo "Choice"</h2></p>

    <x:aviso aviso="${Aviso}"/>
```

```
    <form method="post" name="Formulario">
       <p>N&uacute;mero m&aacute;ximo de elecciones del candidato*: <input type="text"
name="MaxChoices" size="6" value="${MaxChoices}"><br />
       (si es cero no hay restricciones en el n&uacute;mero m&aacute;ximo de elecciones)</p>
       <table border="1" width="600">
          <!-- Un elemento por cada respuesta del ítem -->
          <c:forEach var="numero" begin="0" end="${Num_Resp - 1}">
             <tr>
                <td width="300">Respuesta ${numero + 1}*: <input type="text"
name="Respuesta${numero}" size="45" value="${Respuestas[numero]}"></td>
                <td width="103">Identificador*: <input type="text" name="Identificador${numero}"
size="12" value="${Identificador[numero].identificador}"></td>
                <td><center>&#191;Correcta? <input type="checkbox" name="Correcto${numero}"
<c:if test="${Correcto[numero] == true}">checked="true"</c:if>></center></td>
                <c:if test="${Shuff == true}">
                   <td><center>&#191;Fija? <input type="checkbox" name="Fija${numero}" <c:if
test="${Fija[numero] == true}">checked="true"</c:if>></center></td>
                </c:if>
             </tr>
          </c:forEach>

       </table>

       <p><input type="submit" value="Crear &Iacute;tem XML" name="Form" onClick="return
confirmSubmit('&#191;Est&aacute; conforme con los datos introducidos para crear el &iacute;tem
Choice?')"/>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario,
'Identificador', '${IdentificadorBase}', '', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

respuestasgapmatch.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Introducir huecos y opciones "Gap Match"</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/confirmSubmit.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Introducir los textos, los huecos y las opciones</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <form method="post" name="Formulario">
       <p>Texto hasta el primer hueco: <textarea name="Texto0" cols="50"
rows="3">${Texto[0]}</textarea></p>
       <!-- Los textos intermedios. Su número es igual al número de huecos menos uno -->
       <c:forEach var="huec" begin="1" end="${NumHuec - 1}">
          <p>Texto del hueco ${huec} al hueco ${huec+1}: <textarea name="Texto${huec}"
cols="50" rows="3">${Texto[huec]}</textarea></p>
       </c:forEach>
       <p>Texto tras el &uacute;ltimo hueco: <textarea name="Texto${NumHuec}" cols="50"
rows="3">${Texto[NumHuec]}</textarea></p>

       <p>Para los n&uacute;meros m&aacute;ximos de elecciones un valor de cero indica que no
hay restricciones
       en el n&uacute;mero m&aacute;ximo de elecciones</p>
```

```
      <table border="1">
         <tr>
            <td></td>
         <!-- Un elemento por cada opción del ítem -->
            <c:forEach var="opc" begin="0" end="${NumOpc - 1}">
               <td>
                  Opci&oacute;n ${opc+ 1}*: <input type="text" name="Opcion${opc}" size="15"
value="${Opcion[opc]}"><br />
                  Identificador*: <input type="text" name="IdentOpc${opc}" size="10"
value="${IdentOpc[opc].identificador}"><br />
                  M&aacute;ximas asociaciones*: <input type="text" name="Max${opc}" size="5"
value="${Max[opc]}">
                  <c:if test="${Shuffle == true}">
                     <br />&#191;Fija? <input type="checkbox" name="Fija${opc}" <c:if
test="${Fija[opc] == true}">checked="true"</c:if>>
                  </c:if>
               </td>
            </c:forEach>
         </tr>

         <!-- Un elemento por cada hueco del ítem -->
         <c:forEach var="huec" begin="0" end="${NumHuec - 1}">
            <tr>
               <td>
                  Hueco ${huec + 1}<br />
                  Identificador*: <input type="text" name="IdentHuec${huec}" size="10"
value="${IdentHuec[huec].identificador}"></p>
               </td>
               <!-- Un elemento por cada opción del ítem -->
               <c:forEach var="opc" begin="0" end="${NumOpc- 1}">
                  <td><center>&#191;Combinaci&oacute;n Correcta? <input type="radio"
name="Correcto${huec}" <c:if test="${Correcto[huec] == opc}">checked="true"</c:if>
value="${opc}"></center></td>
               </c:forEach>
            </tr>
         </c:forEach>

      </table>

      <p><input type="submit" value="Crear &Iacute;tem XML" name="Form" onClick="return
confirmSubmit('&#191;Est&aacute; conforme con los datos introducidos para crear el &iacute;tem
Gap Match?')"/>
      <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'IdentOpc',
'${IdentificadorBaseOpcion}', 'IdentHuec', '${IdentificadorBaseHueco}')"/></p>
   </form>

   <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

respuestashottext.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
   <meta http-equiv="Content-Language" content="es"/>
   <title>Introducir textos y "Hot Texts"</title>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/confirmSubmit.js"></script>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

   <jsp:include page="cabecera.jsp"/>

   <p><h2>Introducir los textos y los "Hot Texts"</h2></p>

   <x:aviso aviso="${Aviso}"/>
```

```
    <form method="post" name="Formulario">
       <p>N&uacute;mero m&aacute;ximo de elecciones*: <input type="text" name="Max" size="5"
value="${Max}"></p>
       <p>Texto hasta el primer "Hot Text": <textarea name="Texto0" cols="50"
rows="3">${Texto[0]}</textarea></p>
       <!-- Los hot texts y textos intermedios. Su número es igual al número de huecos
       menos uno -->
       <c:forEach var="hot" begin="1" end="${NumHot}">
          <p>Hot Text ${hot}*: <input type="text" name="Hottext${hot - 1}" size="15"
value="${Hottexts[hot - 1]}"></p>
          <p>Identificador*: <input type="text" name="IdentHot${hot - 1}" size="10"
value="${IdentHot[hot - 1].identificador}"></p>
          <p>&#191;"Hot Text" Correcto? <input type="checkbox" name="Correcto${hot - 1}" <c:if
test="${Correcto[hot - 1] == true}">checked="true"</c:if>>
          <!-- Si es el último hottext pide el último texto, no el texto siguiente -->
          <c:if test="${hot != NumHot}">
             <p>Texto del "Hot Text" ${hot} al "Hot Text" ${hot + 1}: <textarea
name="Texto${hot}" cols="50" rows="3">${Texto[hot]}</textarea></p>
          </c:if>
       </c:forEach>
       <!-- El último texto no se pide en el bucle anterior, sino aquí -->
       <p>Texto tras el &uacute;ltimo "Hot Text": <textarea name="Texto${NumHot}" cols="50"
rows="3">${Texto[NumHot]}</textarea></p>

       <p><input type="submit" value="Crear &Iacute;tem XML" name="Form" onClick="return
confirmSubmit('&#191;Est&aacute; conforme con los datos introducidos para crear el &iacute;tem
Hot Text?')"/>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'IdentHot',
'${IdentificadorBase}', '', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

respuestasinlinechoice.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Introducir textos y opciones "Inline Choice"</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/confirmSubmit.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Itroducir los textos y las opciones tipo "Inline Choice"</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <form method="post" name="Formulario">
       <p>Texto hasta las opciones:<br /><textarea name="Texto1" rows="6"
cols="40">${Texto1}</textarea></p>
       <table border="1" width="600" id="table1">
          <!-- Una fila por cada opción del ítem -->
          <c:forEach var="numero" begin="0" end="${Num_Opc - 1}">
             <tr>
                <td width="300">Opci&oacute;n ${numero + 1}*: <input type="text"
name="Opciones${numero}" size="45" value="${Opciones[numero]}"></td>
                <td width="103">Identificador*: <input type="text" name="Identificador${numero}"
size="12" value="${Identificador[numero].identificador}"></td>
                <td><center>&#191;Correcta? <input type="radio" name="Correcto" <c:if
test="${Correcto == numero}">checked="true"</c:if> value="${numero}"></center></td>
```

```
                <c:if test="${Shuff == true}">
                   <td><center>&#191;Fija? <input type="checkbox" name="Fija${numero}" <c:if
test="${Fija[numero] == true}">checked="true"</c:if>></center></td>
                </c:if>
             </tr>
          </c:forEach>

       </table>
       <p>Texto tras las opciones:<br /><textarea name="Texto2" rows="6"
cols="40">${Texto2}</textarea></p>

       <p><input type="submit" value="Crear &Iacute;tem XML" name="Form" onClick="return
confirmSubmit('&#191;Est&aacute; conforme con los datos introducidos para crear el &iacute;tem
Inline Choice?')"/>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario,
'Identificador', '${IdentificadorBase}', '', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

respuestasmatch.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Introducir respuestas tipo "Match"</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/confirmSubmit.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Introducir ambos grupos de opciones tipo "Match"</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <form method="post" name="Formulario">
       <p>N&uacute;mero m&aacute;ximo de asociaciones totales*: <input type="text"
name="NumMax" size="8" value="${NumMax}"></p>
       <p>(si es cero no hay restricciones en el n&uacute;mero m&aacute;ximo de elecciones)</p>

       <p>Para los n&uacute;meros m&aacute;ximos de elecciones un valor de cero indica que no
hay restricciones
       en el n&uacute;mero m&aacute;ximo de elecciones</p>
       <table border="1" id="table1">
          <tr>
             <td></td>
             <!-- Una columna por cada opción del grupo 2 del ítem -->
             <c:forEach var="num2" begin="0" end="${NumGrupo2-1}">
                <td>
                   Opci&oacute;n 2.${num2 + 1}*: <input type="text" name="Grupo2_${num2}"
size="15" value="${Grupo2[num2]}"><br />
                   Identificador*: <input type="text" name="Identificador2_${num2}" size="10"
value="${Identificador2[num2].identificador}"><br />
                   M&aacute;ximas asociaciones*: <input type="text" name="Max2_${num2}" size="5"
value="${Max2[num2]}">
                   <c:if test="${Shuffle == true}">
                      <br />&#191;Fija? <input type="checkbox" name="Fija2_${num2}" <c:if
test="${Fija2[num2] == true}">checked="true"</c:if>>
                   </c:if>
                </td>
             </c:forEach>
          </tr>
```

```
         <!-- Una fila por cada opción del grupo 1 del ítem -->
         <c:forEach var="num1" begin="0" end="${NumGrupo1 - 1}">
            <tr>

               <td>
                  Opci&oacute;n 1.${num1 + 1}*: <input type="text" name="Grupo1_${num1}"
size="15" value="${Grupo1[num1]}"><br />
                  Identificador*: <input type="text" name="Identificador1_${num1}" size="10"
value="${Identificador1[num1].identificador}"><br />
                  M&aacute;ximas asociaciones*: <input type="text" name="Max1_${num1}" size="5"
value="${Max1[num1]}">
                  <c:if test="${Shuffle == true}">
                     <br />&#191;Fija? <input type="checkbox" name="Fija1_${num1}" <c:if
test="${Fija1[num1] == true}">checked="true"</c:if>>
                  </c:if>
               </td>
               <!-- Una columna por cada opción del grupo 2 del ítem -->
               <c:forEach var="num2" begin="0" end="${NumGrupo2 - 1}">
                  <td><center>&#191;Combinaci&oacute;n Correcta? <input type="checkbox"
name="Correcto${num1} ${num2}" <c:if test="${Correcto[num1][num2] ==
true}">checked="true"</c:if>></center></td>
               </c:forEach>

            </tr>
         </c:forEach>

      </table>

      <p><input type="submit" value="Crear &Iacute;tem XML" name="Form" onClick="return
confirmSubmit('&#191;Est&aacute; conforme con los datos introducidos para crear el &iacute;tem
Match?')"/>
      <input type="button" value="Restablecer" onClick="resetValues(Formulario,
'Identificador1_', '${IdentificadorBaseGrupo1}', 'Identificador2_',
'${IdentificadorBaseGrupo2}')"/></p>
   </form>

   <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

respuestastextentry.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
   <meta http-equiv="Content-Language" content="es"/>
   <title>Introducir respuesta tipo "Text Entry"</title>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/confirmSubmit.js"></script>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

   <jsp:include page="cabecera.jsp"/>

   <p align="center"><u><font size="5">Introducir textos y respuesta tipo "Text
Entry"</font></u></p>

   <x:aviso aviso="${Aviso}"/>

   <form method="post" name="Formulario">
      <p>Texto hasta la respuesta:<br /><textarea name="Texto1" rows="6"
cols="40">${Texto1}</textarea></p>
      <p>Respuesta correcta*: <input type="text" name="Respuesta" size="45"
value="${Respuesta}"></p>
      <p>Longitud esperada: <input type="text" name="Longitud" size="5"
value="${Longitud}"></p>
```

```
      <p>Texto tras la respuesta:<br /><textarea name="Texto2" rows="6"
cols="40">${Texto2}</textarea></p>

      <p><input type="submit" value="Crear &Iacute;tem XML" name="Form" onClick="return
confirmSubmit('&#191;Est&aacute; conforme con los datos introducidos para crear el &iacute;tem
Text Entry?')"/>
      <input type="button" value="Restablecer" onClick="resetValues(Formulario, '', '', '',
'')"/></p>
   </form>

   <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

seleccionaritems.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>

<html>

 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
   <meta http-equiv="Content-Language" content="es"/>
   <title>Seleccionar items</title>
   <script language="JavaScript" type="text/javascript" src="./Scripts/checkbox.js"></script>
   <script language="JavaScript" type="text/javascript"
src="./Scripts/confirmSubmit.js"></script>
 </head>

 <body bgcolor="#C3D3DF">

   <jsp:include page="cabecera.jsp"/>

   <form method="post" name="Formulario">

      <!-- Capa izquierda de la página. Muestra los ítems disponibles para añadir al test -->
      <div style="position:absolute; left:3%; top:185px; width:45%; height:462px;
overflow:scroll">
         <p><font size="4">Directorio Actual: ${DirectorioActual}</font></p>

         <!-- Enlace para volver al directorio superior. Se muestra sólo si el enlace es
         distinto de null, lo que indica que no se puede subir un directorio. -->
         <c:if test="${DirectorioSuperior != null}">
            <p><img src="dir.png"/><a
href="seleccionaritems.jsp?DirectorioActual=${DirectorioSuperior}">/..</a></p>
         </c:if>

         <!-- Lista los directorios del directorio actual -->
         <c:forEach var="dir" items="${NombresDirectorios}">
            <p><img src="dir.png"/><a
href="seleccionaritems.jsp?DirectorioActual=${DirectorioActual}/${dir}">/${dir}</a></p>
         </c:forEach>

         <p><h2>&Iacute;tems disponibles</h2></p>

         <table border="1" width="100%">
            <tr>
               <td width="42%" bgcolor="#ACC0C6"><p align="center"><b>Item</b></p></td>
               <td width="42%" bgcolor="#ACC0C6"><p
align="center"><b>T&iacute;tulo</b></p></td>
               <td width="16%" bgcolor="#ACC0C6"><p align="center"><b><input type="submit"
value="Añadir" name="Añadir"/></b></p></td>
            </tr>

            <!-- La fila con el checkbox para seleccionar todos se muestra sólo si la
            lista de ítems no está vacía -->
            <c:if test="${fn:length(Items) != 0}">
               <tr bgcolor="#6F9AD3">
                  <td></td>
                  <td></td>
```

```
                  <td><p align="center"><input type="checkbox" name="AñadeTodos1"
value="checkboxTodos" onClick="SeleccionarTodos(this, 'Añade_', AñadeTodos2);">Seleccionar
Todos</p></td>
               </tr>
            </c:if>

            <!-- Una fila por cada ítem disponible para añadir al test -->
            <c:forEach var="item" items="${Items}" varStatus="estado">
               <!-- Si el ítem ya está añadido se marca de un color la fila -->
               <tr <c:if test="${Seleccionados[estado.index] ==
true}">bgcolor="#BED6DB"</c:if>>
                  <td ><p align="center">${item.nombre}</p></td>
                  <td><p align="center">${item.titulo}</p></td>
                  <td><p align="center">
                  <!-- Si el ítem ya está añadido se escribe "Añadido" -->
                     <c:if test="${Seleccionados[estado.index] == true}">Añadido</c:if>
                     <input type="checkbox" name="Añade_${estado.index}">
                  </p></td>
               </tr>
            </c:forEach>

            <!-- La fila con el checkbox para seleccionar todos se muestra sólo si la
            lista de ítems no está vacía -->
            <c:if test="${fn:length(Items) != 0}">
               <tr bgcolor="#6F9AD3">
                  <td></td>
                  <td></td>
                  <td><p align="center"><input type="checkbox" name="AñadeTodos2"
value="checkboxTodos" onClick="SeleccionarTodos(this, 'Añade_', AñadeTodos1);">Seleccionar
Todos</p></td>
               </tr>
            </c:if>
         </table>

         <p><a href="seleccionasignatura.jsp?Destino=seleccionaritems.jsp">Cambiar de
Asignatura</a></p>

      </div>

      <!-- Capa derecha de la página. Muestra los ítems seleccionados para añadir al test -->
      <div style="position:absolute; left:52%; top:185px; width:45%; height:462px;
overflow:scroll">
         <p align="center"><u><font size="5">ITEMS SELECCIONADOS ACTUALMENTE</font></u></p>
         <p><font color="#008000" size="4">${Informacion}<!-- Mensajes de información al
usuario en la página --></font></p>
         <x:aviso aviso="${Aviso}"/>
         <p>N&uacute;mero de &iacute;tems seleccionados:
<strong>${fn:length(Items A Añadir)}</strong><p/>
         <table border="1" width="100%">
            <tr>
               <td width="42%" bgcolor="#ACC0C6"><p align="center"><b>Item</b></p></td>
               <td width="42%" bgcolor="#ACC0C6"><p
align="center"><b>T&iacute;tulo</b></p></td>
               <td width="16%" bgcolor="#ACC0C6"><p align="center"><input type="submit"
value="Eliminar" name="Eliminar"/></p></td>
            </tr>

            <!-- La fila con el checkbox para seleccionar todos se muestra sólo si la
            lista de ítems no está vacía -->
            <c:if test="${fn:length(Items_A_Añadir) != 0}">
               <tr bgcolor="#6F9AD3">
                  <td></td>
                  <td></td>
                  <td><p align="center"><input type="checkbox" name="EliminaTodos1"
value="checkboxTodos" onClick="SeleccionarTodos(this, 'Elimina_', EliminaTodos2);">Seleccionar
Todos</p></td>
               </tr>
            </c:if>

            <!-- Una fila por cada ítem seleccionado para añadir al test -->
            <c:forEach var="item" items="${Items_A_Añadir}"  varStatus="estado">
               <tr>
                  <td><p align="center"></p>${item.nombre}</td>
                  <td><p align="center"></p>${item.titulo}</td>
                  <td><p align="center"><input type="checkbox"
name="Elimina_${estado.index}"></p></td>
```

```
               </tr>
            </c:forEach>

            <!-- La fila con el checkbox para seleccionar todos se muestra sólo si la
            lista de ítems no está vacía -->
            <c:if test="${fn:length(Items_A_Añadir) != 0}">
               <tr bgcolor="#6F9AD3">
                  <td></td>
                  <td></td>
                  <td><p align="center"><input type="checkbox" name="EliminaTodos2"
value="checkboxTodos" onClick="SeleccionarTodos(this, 'Elimina_', EliminaTodos1);">Seleccionar
Todos</p></td>
               </tr>
            </c:if>
         </table>

         <input type="submit" value="Introducir Pesos" name="Pesos" onClick="return
confirmSubmit('¿Está conforme con los ítems seleccionados a incluir en el Test?')"/>
      </div>

   </form>

   <div style="position:absolute; top:650px; left:0; width:100%; height:20px">
      <center>
         <small>Herramienta de Creaci&oacute;n de Examen QTI. Universidad de Sevilla.</small>
      </center>
   </div>

 </body>
</html>
```

seleccionasignatura.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
   <meta http-equiv="Content-Language" content="es"/>
   <title>Seleccione la asignatura</title>
   <script language="JavaScript" type="text/javascript" src="./Scripts/popup.js"></script>
 </head>

 <body bgcolor="#C3D3DF">

   <center>

   <p><h2>Seleccionar directorio de asignatura</h2></p>

   <font color="#008000" size="4">${Informacion}<!-- Mensajes de información al usuario en la
página --></font>

   <x:aviso aviso="${Aviso}"/>

   <!-- Lista las asignaturas disponibles -->
   <c:forEach var="dir" items="${NombresAsignaturas}">
      <p><img src="dir.png"/><a href="${Destino}?DirectorioAsignatura=${dir}">/${dir}</a></p>
   </c:forEach>

   <!-- El directorio actual es "/" en la llamada a "nuevodirectorio.jsp" porque está en el
propio
   directorio de trabajo -->
   <p><a
href="nuevodirectorio.jsp?DirectorioActual=/&VolverA=seleccionasignatura.jsp?Destino=${Destino}
">
      Crear Nuevo Directorio de Asignatura
   </a></p>

   <br />

 <jsp:include page="pie.jsp"/>
```

textentry.jsp

```
<%@ taglib prefix="x" tagdir="/WEB-INF/tags/"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Tipo TextEntry</title>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/whizzywig.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/espanol.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/whizzery/xhtml.js"></script>
    <script language="JavaScript" type="text/javascript" src="./Scripts/popup.js"></script>
    <script language="JavaScript" type="text/javascript"
src="./Scripts/resetValues.js"></script>
 </head>

 <body onLoad="document.Formulario.${Focus}.focus()" bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p><h2>Tipo "Text Entry"</h2></p>

    <x:aviso aviso="${Aviso}"/>

    <form method="post" onsubmit="syncTextarea();" name="Formulario">
       <p>Identificador*: <input type="text" name="Identif" size="26" value="${Identif}"></p>
       <p>T&iacute;tulo*: <input type="text" name="Tit" size="34" value="${Tit}"></p>
       <p><a href="seleccionasignatura.jsp?Destino=fileupload.jsp"
             onclick="return
popup('seleccionasignatura.jsp?Destino=fileupload.jsp','fileupload','height=400,width=460,left=
100,top=150,scrollbars = yes')">
             Subir Imagen (bmp, jpeg, png o gif)
       </a></p>
       <p>Instrucciones:</p>
       <p>
          <textarea name="Instruc" id="Instruc" cols="100" rows="20" style="width:1024px;
height:300px">${Instruc}</textarea>
          <x:whizzywig textarea="Instruc"/>
       </p>
       <p><input type="submit" value="Introducir Respuesta" name="Form"/></a>
       <input type="button" value="Restablecer" onClick="resetValues(Formulario, 'Identif', '',
'', '')"/></p>
    </form>

    <p>* Campo obligatorio</p>

 <jsp:include page="pie.jsp"/>
```

tipoitem.jsp

```
<html>

 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
    <meta http-equiv="Content-Language" content="es"/>
    <title>Elegir tipo de Item</title>
 </head>

 <body bgcolor="#C3D3DF">

    <jsp:include page="cabecera.jsp"/>

    <p>
       <a href="choice.jsp"><font size="5">Respuesta Simple o M&uacute;ltiple</font></a>
    </p>

    <p>
       <a href="match.jsp"><font size="5">Correspondencia entre dos grupos</font></a>
    </p>
```

```
    <p>
       <a href="gapmatch.jsp"><font size="5">Correspondencia entre grupo de opciones y huecos
de texto</font></a>
    </p>

    <p>
       <a href="inlinechoice.jsp"><font size="5">Elecci&oacute;n de palabra mediante
desplegable en un texto</font></a>
    </p>

    <p>
       <a href="textentry.jsp"><font size="5">Inserci&oacute;n manual de respuesta en un
texto</font></a>
    </p>

    <p>
       <a href="hottext.jsp"><font size="5">Selecci&oacute;n de palabras en un texto</font></a>
    </p>

 <jsp:include page="pie.jsp"/>
```

4. Funciones JavaScript

Las funciones de JavaScript usadas en las páginas Web de la Aplicación se encuentran en el directorio `Scripts`, y son las diversas funciones de utilidades que se utilizan en las JSPs de la Aplicación. A continuación se muestra su código. Los comentarios están hechos en el formato "Javadoc" de Java, aunque no se vaya a generar dicha documentación de las funciones JavaScript.

checkbox.js

```
/**
 * Función que pone todos los elementos "checkbox" de un formulario cuyo nombre empieza
 * con un nombre base al mismo valor que otro checkbox. También pone al mismo valor otro
 * checkbox recibido como parámetro.
 *
 * Representa la función que hace funcionar un checkbox del tipo "Seleccionar Todos" que
 * ayuda a que cuando se marca se seleccionen todos los checkboxes del formulario
 * facilitando al usuario esa labor.
 *
 * @author  David Domínguez
 * @param   chkbox         elemento checkbox a cuyo valor hay que colocar los demás
 * @param   nombreBase     string con el nombre base común de los checkboxes del
 *                       formulario que se deben poner al mismo valor que "chkbox"
 * @param   elementoPar    elemento checkbox que se tiene que poner al mismo valor que
 *                   "chkbox"
 */
function SeleccionarTodos(chkbox, nombreBase, elementoPar) {

   // Comprueba todos los elementos del formulario
   for (var i = 0; i < document.forms[0].elements.length; i++) {

      // Elemento i del formulario
      var elemento = document.forms[0].elements[i];

      // Comprueba que sea de tipo "checkbox"
      if (elemento.type == "checkbox") {
         // Comprueba que el nombre del checkbox empiece por "nombreBase"
         if (elemento.name.substring(-1, nombreBase.length) == nombreBase) {
            // Pone el valor del checkbox al mismo valor que el del elemento chkbox
            elemento.checked = chkbox.checked;
         }
      }
   }

   // Pone el valor del checkbox par al del elemento chkbox
   elementoPar.checked = chkbox.checked;
}
```

confirmSubmit.js

```
/**
 * Función que pide una confirmación al usuario mediante una pequeña ventana mostrándole
 * un mensaje para ayudarle a tomar la decisión. Devuelve true o false en función de si
 * aceptó o no respectivamente.
 *
 * @author  David Domínguez
 * @param   mensaje        string con la pregunta que se le hace al usuario
 * @return                 true o false en función de si el usuario aceptó o no la
 *                         confirmación, respectivamente.
 */
function confirmSubmit(mensaje) {
   // Variable con la respuesta del usuario a la petición de confirmación con el
   // mensaje
   var confirmacion = confirm(mensaje);

   // Devuelve true o false en función de la respuesta del usuario
   if (confirmacion)
      return true ;
   else
      return false ;
}
```

imagen.js

```
/**
 * Función que devuelve la URL deseada de la imagen para incluir en las instrucciones.
 * Establece el valor de la constante definida en el JavaScript whizzywig.js. Luego
 * cierra la ventana de tipo 'pop up' para elegir imagen
 *
 * @author  David Domínguez
 * @param   url            string con la URL de la imagen a incluir en las
 *                         instrucciones
 */
function WantThis(url) {
   window.parent.opener.document.getElementById('if_url').value = url;
   window.parent.close();
}

/**
 * Función que cambia la imagen mostrada en el campo "nombre" del frame "frame" de la
 * página por la imagen recibida en el parámetro "imag". Se cosidera que la llamada a
 * esta función se realiza desde otro frame de la misma página.
 *
 * @author  David Domínguez
 * @param   imag           dirección de la imagen nueva a mostrar
 * @param   nombre         nombre del campo de imagen a modificar
 * @param   frame          nombre del frame que contiene el campo imagen a modificar
 */
function cambiaImagen(imag, nombre, frame) {
   window.parent.frames[frame].document.images[nombre].src=imag;
}
```

popup.js

```
/**
 * Función que abre una nueva ventana del navegador de tipo 'pop up', especificando su
 * URL, su nombre y los parámetros de la ventana.
 *
 * @author  David Domínguez
 * @param   url            dirección de la página a abrir en la nueva ventana
 * @param   nombre         nombre de la nueva ventana
 * @param   params         parámetros de la nueva ventana, como tamaño, si tiene barra
 *                         de
 *                         desplazamiento, etc.
 */
function popup(url, nombre, params) {
   // Abre la nueva ventana con los parámetros recibidos
   nuevaVentana = window.open(url,nombre,params);

   // Si el 'focus' está en la ventana original
   if (window.focus)
```

```
       {nuevaVentana.focus()} // Coloca el 'focus' en la nueva ventana de 'po-pup'

   return false;  //Para no abrir la página en la ventana principal
}
```

resetValues.js

```
/**
 * Función que se encarga de restablecer los valores de los campos de un formulario. Los
 * campos de tipo 'checkbox' y 'radio' los desmarca, y los campos de tipo 'textarea' los
 * pone en blanco. Los campo de tipo 'text' reciben un tratamiento especial:si el campo
 * no es de un identificador, se pone en blanco, pero si el campo es un identificador
 * hay que ponerlo a su valor inicial por defecto, si lo tiene, seguido del número que
 * corresponda a la posición de ese campo particular entre todos los de ese mismo tipo.
 * Puede haber en una misma página dos conjuntos de campos identificador, cada uno con
 * un valor inicial por defecto. Recibe el objeto representando al formulario de la
 * página cuyos campos hay que poner a su valor inicial, el nombre del primer tipo de
 * campo identificador, el valor del identificador por defecto de ese campo, el nombre
 * del segundo tipo de campo identificador y el valor del identificador por defecto de
 * ese campo.
 *
 * Puede que no todas las páginas tengan dos conjuntos de campos identificador. Puede
 * que no tengan ni un primer conjunto de campos identificador, en cuyo caso, los campos
 * correspondientes llegarán igual a la cadena vacía.
 *
 * @author  David Domínguez
 * @param   formulario     objeto representando al formulario de la página cuyos
 *                         campos se quieren poner a sus valores iniciales
 * @param   campo1         nombre común del primer conjunto de identificadores de la
 *                         página
 * @param   identificador1 nombre base por defecto que toman los identificadores del
 *                         primer grupo de la página. Deberá estar seguido de un
 *                         número con la posición de cada elemento particular.
 * @param   campo2         nombre común del segundo conjunto de identificadores de la
 *                         página
 * @param   identificador2 nombre base por defecto que toman los identificadores del
 *                         segundo  grupo de la página. Deberá estar seguido de un
 *                         número con la posición de cada elemento particular.
 */
function resetValues(formulario, campo1, identificador1, campo2, identificador2) {

   // Recorre todos los elementos del formulario
   for (var i = 0; i < formulario.elements.length; i++) {
      // Elemento del formulario a poner a su valor inicial
      var elemento = formulario.elements[i];

      // Si es de tipo 'checkbox' lo desmarca
      if (elemento.type == "checkbox") {
         elemento.checked = false;
      }
      // Si es de tipo 'text' hay que iniciar bien los campos que sean de
      // Identificadores
      else if (elemento.type == "text") {
         // Comprueba si es un identificador buscando que su nombre contenga el
         // nombre base
         // del primer grupo de identificadores a comprobar
         if (elemento.name.lastIndexOf(campo1, elemento.name.length - 1) != -1) {
            // Si el valor por defecto del identificador es vacío, pone el campo
            // vacío
            if (identificador1 == "") {
               elemento.value = "";
            }
            // Si el valor por defecto del identificador no es vacío
            else {
               // Posición contando desde el final del nombre del campo que
               // contiene un número
               var pos;
               // Caracter del nombre del campo a comprobar si es un número
               var numero;
               // Comprueba cuántos caracteres del final del nombre del campo son
               // números
               for (pos = 1, numero = elemento.name.charAt(elemento.name.length - pos);
                     numero <= '9' && numero >= '0';
                     numero = elemento.name.charAt(elemento.name.length - pos)){
```

```
                  pos++;
               }
               // Número del final del nombre del campo que indica la posición de
               // éste
               // entre todos los campos con el mismo nombre base
               posicion =
                     new Number(elemento.name.substring(elemento.name.length - pos + 1,
                     elemento.name.length)) + 1;
               // Nombre por defecto del campo igual al nombre base seguido de la
               // posición
               elemento.value = identificador1 + posicion;
            }
         }
         // Comprueba si es un identificador buscando que su nombre contenga el
         // nombre base del segundo grupo de identificadores a comprobar
         else if (elemento.name.lastIndexOf(campo2, elemento.name.length) != -1){
            // Si el valor por defecto del identificador es vacío, pone el campo
            // vacío
            if (identificador2 == "") {
               elemento.value = "";
            }
            // Si el valor por defecto del identificador no es vacío
            else {
               // Comprueba cuántos caracteres del final del nombre del campo son
               // números
               for (pos = 1, numero = elemento.name.charAt(elemento.name.length - pos);
                     numero <= '9' && numero >= '0';
                     numero = elemento.name.charAt(elemento.name.length - pos)){
                  pos++;
               }
               // Número del final del nombre del campo que indica la posición de
               // éste entre todos los campos con el mismo nombre base
               posicion =
                     new Number(elemento.name.substring(elemento.name.length - pos + 1,
                     elemento.name.length)) + 1;
               // Nombre por defecto del campo igual al nombre base seguido de la
               // posición
               elemento.value = identificador2 + posicion;
            }
         }
         // Si el campo no es de identificador, se hace su valor igual a vacío
         else {
            elemento.value = "";
         }
      }
      // Si es de tipo 'textarea' la pone igual a vacío
      else if (elemento.type == "textarea") {
         elemento.value = "";
      }
      // si es de tipo 'radio' lo desmarca
      else if (elemento.type == "radio") {
         elemento.checked = false;
      }
   }
}
```

5. Biblioteca de etiquetas `util.tld`

Descriptor de la biblioteca de etiquetas llamada *util*. Mediante su nombre se incluye en las JSPs para poder usar las etiquetas que agrupa. Esta biblioteca agrupa tres etiquetas en ella: la etiqueta *adminMail*, la etiqueta *reqURI* y la etiqueta *logger*. El código se puede ver a continuación.

```
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
      version="2.0">
```

```
    <!-- Versión de la especificación de bibliotecas de etiquetas utilizada -->
    <tlib-version>1.0</tlib-version>

    <!-- Nombre de la biblioteca -->
    <short-name>Util TLD</short-name>

    <!-- Versión de JSP utilizada -->
    <jsp-version>2.0</jsp-version>

    <!-- URI ficticia definida para acceder a ella directamente en las JSP
        sin tener en cuenta su localización real -->
    <uri>http://www.QTI.us.es/util</uri>

    <!-- Definición de la etiqueta que imprime en pantalla la dirección de
        correo electrónico del administrador del sistema-->
    <tag>
       <!-- Nombre de la etiqueta para acceder a ella -->
       <name>adminMail</name>
       <!-- Clase que maneja la etiqueta -->
       <tag-class>utilidades.etiquetas.util.AdminMailTag</tag-class>
       <!-- La etiqueta tiene el cuerpo vacío -->
       <body-content>empty</body-content>
    </tag>

    <!-- Definición de la etiqueta que imprime en pantalla la URL que ha
        provocado un error -->
    <tag>
       <!-- Nombre de la etiqueta para acceder a ella -->
       <name>reqURI</name>
       <!-- Clase que maneja la etiqueta -->
       <tag-class>utilidades.etiquetas.util.RequestedURITag</tag-class>
       <!-- La etiqueta tiene el cuerpo vacío -->
       <body-content>empty</body-content>
    </tag>

    <!-- Definición de la etiqueta que registra un error utilizando el
        sistema de registro de la Aplicación Web -->
    <tag>
       <!-- Nombre de la etiqueta para acceder a ella -->
       <name>logger</name>
       <!-- Clase que maneja la etiqueta -->
       <tag-class>utilidades.etiquetas.util.logging.LoggerTag</tag-class>
       <!-- La etiqueta tiene el cuerpo vacío -->
       <body-content>empty</body-content>
       <!-- Atributo de la etiqueta -->
       <attribute>
          <!-- Nombre del atributo -->
          <name>nivel</name>
          <!-- El atributo es obligatorio -->
          <required>true</required>
          <!-- Tipo cadena, pero debe ser un nivel de importancia de
              registro -->
          <type>java.lang.String</type>
       </attribute>
       <!-- Otro atributo de la etiqueta -->
       <attribute>
          <!-- Nombre del atributo -->
          <name>mensaje</name>
          <!-- El atributo es obligatorio -->
          <required>true</required>
          <!-- Tipo cadena -->
          <type>java.lang.String</type>
       </attribute>
    </tag>
</taglib>
```

6. Archivos “tag” de etiquetas personalizadas

Son etiquetas personalizadas de tipo especial que se pueden implementar utilizando lenguaje JSP. Seguidamente se muestra el código de las dos etiquetas de este tipo que contiene la Aplicación.

aviso.tag

```
<!-- Atributo de la etiqueta, requerido -->
<!-- Es el mensaje de aviso a mostrar al usuario -->
<%@ attribute name="aviso" required="true" %>

<!-- Mensajes de aviso al usuario en la página -->
<p><font color="#FF0000" size="4">${aviso}</font></p>


<script  language="JavaScript" type="text/javascript">
<!--
   var cadena = "${aviso}";
   if (cadena.length != 0) {
      // Sustituye los caracteres <br /> HTML por caracteres nueva
      // línea
      var cadena_array = cadena.split("<br />");
      salida = "";
      for (i = 0; i < cadena_array.length - 1; i++) {
         salida += cadena_array[i] + "\n";
      }
      salida += cadena_array[cadena_array.length - 1];

      alert(salida);
   }
// -->
</script>
```

whizzywig.tag

```
<!-- Atributo de la etiqueta, requerido -->
<!-- Es el nombre del textarea de la página que tendrá el editor -->
<%@ attribute name="textarea" required="true" %>

<script language="JavaScript" type="text/javascript">
<!--
   // Página del explorador de imágenes a utilizar
   imageBrowse = "buscadorimagenes.jsp";
   // Declaración de botones personalizados (minúsculas)
   buts = 'nuevalinea cita definicion salidaprograma subindice
superindice';
   // Acción de los botones personalizados (sin etiqueta de cierre):
   dobut['nuevalinea'] = '<br>';
   dobut['cita'] = '<cite>';
   dobut['definicion'] = '<dfn>';
   dobut['salidaprograma'] = '<samp>';
   dobut['subindice'] = '<sub>';
   dobut['superindice'] = '<sup>';
   // LLamada a la función que carga el editor, con el nombre del
   // textarea y el orden de los elementos que va a incluir
   makeWhizzyWig("${textarea}", "formatblock fontsize | bold italic
underline cita definicion salidaprograma | newline subindice
```

```
superindice | left center right | number bullet | newline color hilite
rule | nuevalinea | undo redo | link image table | clean html");
// -->
</script>
```