

IES CHAN DO MONTE

C.S. de Desarrollo de Aplicaciones Multiplataforma

UNIDAD 1 parte 2: Gestión de Ficheros XML SAX

Índice

1.	<i>Acceso a ficheros XML con SAX</i>	2
2.	<i>APIs para trabajar con SAX</i>	3
2.1	API SAX pura (org.xml.sax)	3
2.2	API JAXP (javax.xml.parsers)	3
2.3	Uso de XMLReader desde SAXParserFactory	5
2.4	Excepciones comunes en XMLReader y SAXParser	7
2.5	Validación de documentos XML con SAX: DTD y XSD	8
2.6	Interfaz ContentHandler y clase DefaultHandler	10

1. Acceso a ficheros XML con SAX

SAX (Simple API for XML) es una API para procesar documentos XML de forma **secuencial y basada en eventos**. A diferencia de DOM, **no construye un árbol en memoria**, sino que lanza eventos a medida que recorre el documento.

Características principales:

- Procesamiento **rápido y eficiente**, ideal para documentos grandes.
- **No permite modificar** el XML directamente (solo lectura).
- El programador debe implementar un **manejador de eventos** que reaccione a las etiquetas, atributos y texto.
- SAX lanza eventos como:
 - `startDocument()` / `endDocument()`
 - `startElement(...)` / `endElement(...)`
 - `characters(...)`

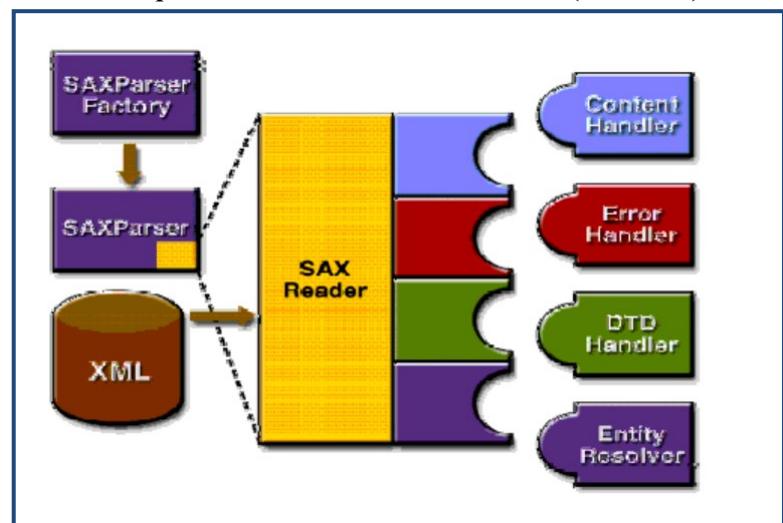
Se necesita para trabajar con SAX:

- La **creación de un modelo de objetos en memoria** para almacenar la información por parte del programador.
- Una **clase que reciba los eventos SAX** y cree objetos en el modelo de objetos. SAX lanza eventos por cada etiqueta de apertura y de cierre, para los DTD's, comentarios, etc.
- El programador debe ser **consciente de los eventos que se producen y del orden de los mismos**. Los métodos permiten procesar el documento guiado por eventos.

Funcionamiento:

Durante el reconocimiento del documento, **cada vez que se identifica una estructura (elemento)**, se mira si **hay un método que manipula ese elemento**, si es así, **se invoca a este método**. Al terminar, se continúa con el reconocimiento hasta el final del documento.

SAX define cuatro interfaces básicas:



Interfaces clave en SAX:

Interfaz	Función principal
<code>ContentHandler</code>	Gestiona eventos del documento (inicio, etiquetas, texto).
<code>ErrorHandler</code>	Gestiona errores y advertencias durante el análisis.
<code>DTDHandler</code>	Gestiona eventos relacionados con DTDs.
<code>EntityResolver</code>	Resuelve referencias a entidades externas.

- **ContentHandler:** se utiliza para tratar eventos generales del documento, como apertura y cierre de etiquetas o cuando aparecen bloques de texto. Algunos métodos:
 - startDocument(): invocado cuando comienza un documento.
 - endDocument(): invocado cuando finaliza un documento.
 - startElement(nombre, atributos): invocado cuando se abre una etiqueta.
 - endElement(nombre): invocado cuando se cierra una etiqueta.
 - characters(texto): invocado cuando aparece un bloque de texto.
- **ErrorHandler:** maneja los errores y warnings. Ya se vió está interfaz en el tema de validación de un documento XML con DOM.
 - **error:** invocado cuando el parser encuentra un error recuperable. El análisis del documento puede continuar.
 - **fatalError(excepción):** invocado cuando el parser encuentra un error irrecuperable. El análisis debe finalizar.
 - **warning(excepción):** invocado cuando el parser da “avisos”, el análisis debe continuar.
- DTDHandler: invocado para tratar eventos relacionados con los DTD's.
- EntityResolver: se utiliza para resolver referencias a entidades externas, si éstas no se utilizan no es necesario implementar este interfaz.

Componentes necesarios

- Un parser XML
- Las clases de SAX
- Un documento XML

2. APIs para trabajar con SAX

Existen varias formas de trabajar con SAX en Java, desde la API pura hasta las API modernas de JAXP:

2.1 API SAX pura (org.xml.sax)

- Es la API original de SAX: Es la forma directa y clásica de usar SAX.
- Permite procesar XML de manera secuencial y basada en eventos mediante un XMLReader y un ContentHandler.
- Usa XMLReaderFactory para crear el parser.
- Algunos métodos y clases (como XMLReaderFactory) están deprecated en Java moderno.
- Requiere configurar manualmente el manejador.

```
XMLReader procesador = XMLReaderFactory.createXMLReader();
procesador.setContentHandler(new MiManejador());
procesador.parse("documento.xml");
```

2.2 API JAXP (javax.xml.parsers)

- Es la forma más moderna moderna.
- Proporciona una interfaz unificada para SAX, DOM y XSLT.

- Permite configurar validación, namespaces, etc.

Clases clave:

- SAXParserFactory**: crea instancias de SAXParser y permite configurarlas.
- SAXParser**: parser SAX que implementa la interfaz de análisis secuencial.

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
ContentHandler manejador = new MiManejador();
parser.parse("documento.xml", manejador);
```

Métodos importantes

Método	Descripción
<code>newSAXParser()</code>	Crea una instancia de SAXParser .
<code>parse(String uri, DefaultHandler handler)</code>	Analiza el documento XML con el manejador indicado.
<code>getXMLReader()</code>	Devuelve el XMLReader interno para configuración avanzada.
<code>isNamespaceAware()</code>	Indica si el parser reconoce espacios de nombres.
<code>isValidating()</code>	Indica si el parser realiza validación.

Características del Analizador SAX XMLREADER

Característica	Método	Valor
Validación con DTD	<code>setValidating(true)</code>	<code>boolean</code>
Espacios de nombres	<code>setNamespaceAware(true)</code>	<code>boolean</code>

```
try {
    // Creamos la fábrica de parsers SAX
    SAXParserFactory factory = SAXParserFactory.newInstance();

    // Configuramos características del parser
    factory.setNamespaceAware(true); // Reconoce espacios de nombres
    factory.setValidating(true); // Activa validación con DTD o XSD

    // Creamos el parser SAX
    SAXParser parser = factory.newSAXParser();

    // Asignamos el manejador de eventos
    MiManejador manejador = new MiManejador(); // Extiende DefaultHandler

    // Procesamos el documento XML
    parser.parse("documento.xml", manejador);

} catch (Exception e) {
    System.err.println("Error: " + e.getMessage());
}
```

2.3 Uso de XMLReader desde SAXParserFactory

Aunque **XMLReaderFactory** está **deprecated**, **XMLReader** sigue siendo la interfaz central de SAX para procesar eventos de XML.

Con JAXP, podemos obtener un **XMLReader** moderno a partir de un **SAXParser**, lo que nos permite mantener compatibilidad con la API SAX pura, pero sin usar código obsoleto.

Pasos para usar XMLReader con JAXP:

- Crear un **SAXParser** mediante la fábrica de JAXP:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
```

- Obtener el **XMLReader** del **SAXParser**:

```
XMLReader xmlReader = saxParser.getXMLReader();
```

- Configurar el **ContentHandler** que responderá a los eventos SAX:

```
MiManejador manejador = new MiManejador();
xmlReader.setContentHandler(manejador);
```

- Parsear el documento XML:

```
xmlReader.parse(new InputSource(ruta));
```

Métodos importantes

Métodos	
ContentHandler	getContentHandler() Devuelve el controlador ContentHandler de eventos actual.
ErrorHandler	getErrorHandler() Devuelve el controlador ErrorHandler de manejo de errores actual.
boolean	getFeature(java.lang.String name) throws SAXNotRecognizedException, SAXNotSupportedException Devuelve true si la característica es soportada por el analizador XML.
java.lang.Object	getProperty(java.lang.String name) Devuelve el valor de una propiedad
void	parse (InputSource input) throws java.io.IOException, SAXException Analiza un documento XML de cualquier fuente de entrada válida (un flujo de carácter, un flujo de bytes, o un URI).
void	parse(java.lang.String systemId) throws java.io.IOException, SAXException Analiza un documento XML de un identificador de sistema (URI). Es equivalente <code>parse (InputSource (SistemaURI))</code>
void	setContentHandler (ContentHandler handler) Permite que una aplicación registre un controlador de eventos.
void	setErrorHandler (ErrorHandler handler) Permite que una aplicación registre un controlador de eventos de error
void	setFeature(java.lang.String name, boolean value) throws SAXNotRecognizedException, SAXNotSupportedException Establece el valor de una característica del analizador
void	setProperty(java.lang.String name, java.lang.Object value) throws SAXNotRecognizedException, SAXNotSupportedException Establece el valor de una propiedad.

```

package leerficheroxmlsax;
import java.io.FileInputStream;
import java.io.IOException;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;

public class LeerFicheroXMLSAX {
    public static void main(String[] args) {
        try {
            // Creamos la fábrica de parsers SAX (JAXP)
            SAXParserFactory factory = SAXParserFactory.newInstance();
            // Creamos el parser SAX
            SAXParser saxParser = factory.newSAXParser();

            // Obtenemos el XMLReader desde el SAXParser
            XMLReader Analizador = saxParser.getXMLReader();

            // Añadimos nuestro manejador de eventos al analizador
            /*Manejador es una clase que implementa ContentHandler (aquí hay que implementar
            todos su métodos, pero para solo implementar los que necesitemos se extiende de
            DefaultHandler que implementa la interfaz ContentHandler*/
            Analizador.setContentHandler(new Manejador());
            // Procesamos el xml de ejemplo
            Analizador.parse(new InputSource(new
                FileInputStream("Actores.xml")));
        } catch (SAXException e) { ..... }
        catch (IOException e) { .... }
    }
}

```

Características del Analizador SAX XMLREADER

El Analizador XMLReader, se configura mediante el establecimiento de sus características.

- Una característica tiene un valor booleano **verdadero / falso**.
- **Las características se nombran por URI absolutos.**

Obtener y establecer Características

La interfaz XMLReader proporciona estos dos métodos para activar y desactivar funciones:

```

public void setFeature(String name, boolean value) throws SAXNotRecognizedException,
                                                       SAXNotSupportedException;
public boolean getFeature(String name) throws SAXNotRecognizedException,
                                             SAXNotSupportedException;

```

El primer argumento es el nombre de la función para establecer u obtener una característica.

Las características estándar que son compatibles con varios analizadores tienen nombres que comienzan con <http://xml.org/sax/features>.

Sin embargo, también se admiten características personalizadas de diferentes analizadores. Los nombres de estas características comienzan con las URLs en el dominio del proveedor del analizador. Por ejemplo, las características no estándar del analizador Xerces del proyecto Apache XML comienzan con <http://apache.org/xml/features/>.

Por ejemplo:

```

public static void main(String[] args) {
    try {
        //Características del analizador
        String val="http://xml.org/sax/features/validation";
        String namespaces="http://xml.org/sax/features/namespaces";
        String esquemas="http://apache.org/xml/features/validation/schema";
        // Creamos la fábrica de parsers SAX (JAXP)
        SAXParserFactory factory = SAXParserFactory.newInstance();
        // Creamos el parser SAX
        SAXParser saxParser = factory.newSAXParser();

        // Obtenemos el XMLReader desde el SAXParser
        XMLReader Analizador = saxParser.getXMLReader();
        if (Analizador.getFeature(val))
            System.out.println("Esta activada la validación");
        else{
            System.out.println("NO Esta activada la validación. Se va a establecer");
            Analizador.setFeature(val,true);
        }
        if (Analizador.getFeature(namespaces))
            System.out.println("Soporta espacios de nombres");
        else{
            System.out.println("NO soporta espacios de nombres. Se van a establecer");
            Analizador.setFeature(namespaces,true);
        }
        if (Analizador.getFeature(esquemas))
            System.out.println("Soporta validacion con esquemas");
        else{
            System.out.println("NO soporta validación con esquemas. Se va a establecer");
            Analizador.setFeature(esquemas,true);
        }
    }
}

```

Salida:

```

run:
NO Esta activada la validación. Se va a establecer
Soporta espacios de nombres
NO soporta validadcción con esquemas. Se va a establecer

```

2.4 Excepciones comunes en XMLReader y SAXParser

Excepción	Cuándo se lanza	Superclase
SAXNotRecognizedException	Cuando el nombre de la característica no es reconocido por el parser.	SAXException
SAXNotSupportedException	Cuando la característica es reconocida pero no puede activarse en ese momento o no está soportada.	SAXException

2.5 Validación de documentos XML con SAX: DTD y XSD

Validación con DTD usando SAXParser (JAXP)

Requisitos:

- El XML debe declarar el DTD en la cabecera:

```
<!DOCTYPE Actores SYSTEM "actores.dtd">
```

- Se debe activar la validación con:

```
factory.setValidating(true);
```

Ejemplo :

```
public class ValidarConDTD_SAXParser {
    public static void main(String[] args) {
        try {
            // Crear la fábrica y activar validación
            SAXParserFactory factory = SAXParserFactory.newInstance();
            factory.setValidating(true); // Activar validación con DTD
            factory.setNamespaceAware(false); // No es necesario para DTD

            // Crear el parser
            SAXParser parser = factory.newSAXParser();

            // Analizar el XML con un manejador personalizado
            parser.parse("actores.xml", new Manejador());

            } catch (Exception e) {
                System.err.println("Error: " + e.getMessage());
            }
        }
    }
```

Validación con XSD usando SAXParser (JAXP)

Requisitos

- El XML debe declarar el esquema:

```
<Actores xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="actores.xsd">
```

- Se debe activar:

```
factory.setValidating(true);
factory.setNamespaceAware(true);
```

Ejemplo :

```
try {
    // Crear la fábrica y configurar validación con XSD
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setValidating(true); // Necesario para activar validación
    factory.setNamespaceAware(true); // Obligatorio para XSD

    // Crear el parser
    SAXParser parser = factory.newSAXParser();

    // Analizar el XML con el manejador
    parser.parse("actores.xml", new Manejador());

    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
}
```

Validación con DTD usando XMLReader (desde SAXParser)

Requisitos

- El XML debe declarar el DTD.
- Se activa la característica:

```
reader.setFeature("http://xml.org/sax/features/validation", true);
```

Ejemplo:

```
try {
    // Crear el parser y obtener el XMLReader
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser parser = factory.newSAXParser();
    XMLReader reader = parser.getXMLReader();

    // Activar validación con DTD
    reader.setFeature("http://xml.org/sax/features/validation", true);

    // Asignar manejador
    reader.setContentHandler(new Manejador());

    // Analizar el documento
    reader.parse(new InputSource("actores.xml"));

} catch (Exception e) {
    System.err.println("Error: " + e.getMessage());
}
```

Validación con XSD usando XMLReader (desde SAXParser)

Requisitos

- El XML debe declarar el esquema con xsi:noNamespaceSchemaLocation.
- Se activan dos características:

```
"http://xml.org/sax/features/validation"
"http://apache.org/xml/features/validation/schema"
```

Ejemplo completo

```
try {
    // Crear el parser y obtener el XMLReader
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setNamespaceAware(true); // Obligatorio para XSD
    factory.setValidating(true); // Activar validación

    SAXParser parser = factory.newSAXParser();
    XMLReader reader = parser.getXMLReader();

    // Activar validación con XSD
    reader.setFeature("http://xml.org/sax/features/validation", true);
    reader.setFeature("http://apache.org/xml/features/validation/schema", true);

    // Asignar manejador
    reader.setContentHandler(new Manejador());

    // Analizar el documento
    reader.parse(new InputSource("actores.xml"));

} catch (Exception e) {
    System.err.println("Error: " + e.getMessage());
}
```

2.6 Interfaz ContentHandler y clase DefaultHandler

Esta interfaz es el centro de todo el proceso de XML con SAX. Aquí se **definen todos los eventos** que se producen a lo largo del procesamiento del documento XML, así que nuestras clases deberán implementar este interface para que el analizador XML realice lo que necesitemos

org.xml.sax

Interface ContentHandler

All Known Subinterfaces:
[TemplatesHandler](#), [TransformerHandler](#), [UnmarshallerHandler](#)
All Known Implementing Classes:
[DefaultHandler](#), [DefaultHandler2](#), [ValidatorHandler](#), [XMLFilterImpl](#), [XMLReaderAdapter](#)

Métodos	
Modificador y tipo	Método y descripción
void	startDocument () throws SAXException El evento que se produce al comenzar a procesar un documento XML
void	endDocument () throws SAXException Recibe notificación del final de un documento
void	startElement (String uri, String localName, String qName, Attributes atts) throws SAXException Este método se ejecuta cuando empieza un elemento del documento XML. Los argumentos que recibe son la dirección URI del espacio de nombres asociado al elemento, el nombre del elemento (o etiqueta) sin el prefijo del espacio de nombres, el nombre del elemento en la versión 1.0 de la especificación de XML, y los atributos que contiene la etiqueta en forma de una instancia de la clase Attributes.
void	endElement (String uri, String localName, String qName) throws SAXException Recibe notificación del final de un elemento.
void	characters (char[] ch, int start, int length) throws SAXException Este método consigue el valor del elemento, es decir, el contenido entre las etiquetas de inicio y final del elemento. Los parámetros que recibe son un array de caracteres, así como la indicación del inicio y la longitud.
void	ignorableWhitespace (char[] ch, int start, int length) throws SAXException Este método indica los espacios en blanco que pueden ser ignorados en el documento, pero normalmente solo se utiliza cuando se valida el documento durante el procesamiento del fichero. Los parámetros tienen el mismo significado que en el método characters
void	processingInstruction (String target, String data) throws SAXException Este evento se produce cuando se encuentra una instrucción de proceso de XML (llamadas también PI) distinta a la declaración de comienzo XML. Los argumentos de este método son el destino de la instrucción y los atributos de dicha instrucción.
void	setDocumentLocator (Locator locator) Este método recibe una instancia de la clase Locator que contendrá la información referente a la posición del documento donde sucede un evento.
void	skippedEntity (String name) throws SAXException Este método indica que una entidad externa se ha ignorado al procesar el fichero. Recibe como parámetros el nombre de dicha entidad.
void	startPrefixMapping (String prefix, String uri) throws SAXException

SAXException

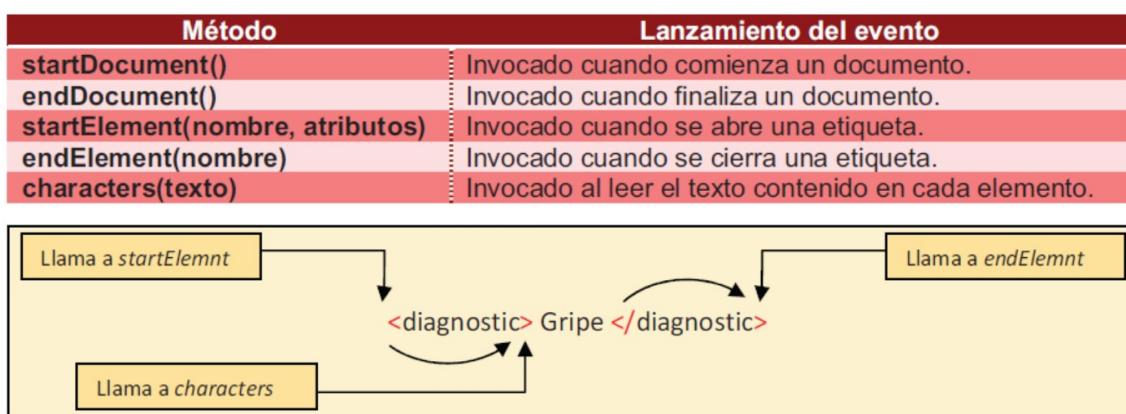
Indica el comienzo de un prefijo de un espacio de nombre (Namespace). Los parámetros de este método son el prefijo del espacio de nombres y su dirección URI asociada

void

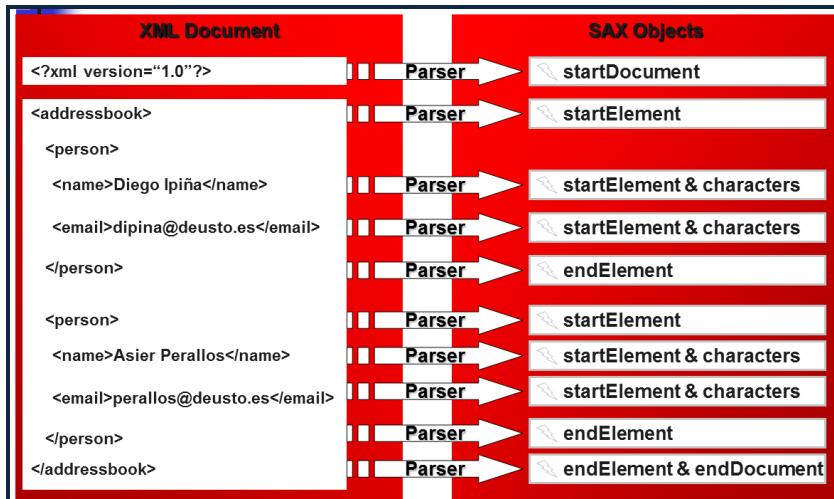
endPrefixMapping (String prefix) throws SAXException

Indica cuando se ha terminado la petición de inicio del prefijo del espacio de nombres.

Los métodos más utilizados son:



Ejemplo:



Si utilizamos la anterior interfaz, tenemos que implementar todos los métodos, pero puede que solo necesitemos unos cuantos métodos, entonces para ahorrar esfuerzo, existe una clase que implementa esta interface, como siempre, pero con los métodos vacíos para que se tenga que extenderla y sobreescribir los que métodos que nos hagan falta. Esta clase se llama **DefaultHandler**.

org.xml.sax.helpers

Class DefaultHandler

```
java.lang.Object
└ org.xml.helpers.DefaultHandler
```

All Implemented Interfaces:

[ContentHandler](#), [DTDHandler](#), [EntityResolver](#), [ErrorHandler](#)

¿Qué son localName y qName?

Cuando SAX analiza un documento XML, lanza eventos como:

```
startElement(String uri, String localName, String qName, Attributes attributes)
```

Estos parámetros representan:

Parámetro	Significado
uri	URI del espacio de nombres (namespace) asociado al elemento.
localName	Nombre local del elemento, sin prefijo.
qName	Nombre calificado del elemento, con prefijo si lo hay. Es el nombre tal como aparece en el XML.

Ejemplo con namespaces

Supón que tienes este XML:

```
<libro xmlns:bk="http://ejemplo.org/biblioteca">
    <bk:titulo>La sombra del viento</bk:titulo>
</libro>
```

Parámetro	Valor
uri	"http://ejemplo.org/biblioteca"
localName	"titulo"
qName	"bk:titulo"

¿Cuándo puede ser localName null o vacío?

Si el parser no está configurado para usar namespaces

Si no activas esta línea:

```
factory.setNamespaceAware(true);
```

entonces el parser no separa prefijos y espacios de nombres, y localName puede venir vacío o null. En ese caso, solo qName tendrá valor.

Clase Manejador.java

```
/*
 * Gestiona los eventos cuando se procesa un documento XML con SAX.
 * Esta clase extiende DefaultHandler y reacciona a los eventos SAX:
 * - Inicio y fin del documento
 * - Inicio y fin de cada etiqueta
 * - Texto entre etiquetas
 */
package learficheromlsax;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class Manejador extends DefaultHandler {

    // Se ejecuta al comenzar el análisis del documento XML
    @Override
    public void startDocument() throws SAXException {
        System.out.print("Principio del documento...");
    }
}
```

```

// Se ejecuta al finalizar el análisis del documento XML
@Override
public void endDocument() throws SAXException {
    System.out.println("Fin del documento...");
}

// Se ejecuta al encontrar una etiqueta de apertura
@Override
public void startElement(String uri, String localName, String name, Attributes
attributes) throws SAXException {
    System.out.println("\nProcesando etiqueta de apertura...");

    // uri: espacio de nombres (namespace) si se usa
    System.out.println("\tNamespace URI: " + uri);

    // localName: nombre local del elemento (sin prefijo)
    System.out.println("\tNombre local: " + localName);

    // name (qName): nombre tal como aparece en el XML, con prefijo si lo hay
    System.out.println("\tNombre con prefijo (qName): " + name);

    // Procesar atributos de la etiqueta
    System.out.println("\tProcesando " + attributes.getLength() + " "
atributos...");
    for (int i = 0; i < attributes.getLength(); i++) {
        System.out.println("\t\tNombre: " + attributes.getQName(i));
        System.out.println("\t\tValor: " + attributes.getValue(i));
    }

    // Acceso directo a un atributo concreto por nombre
    String valorId = attributes.getValue("id");
    if (valorId != null) {
        System.out.println("\tId detectado: " + valorId);
    }
}

// Se ejecuta al encontrar texto entre etiquetas
@Override
public void characters(char[] ch, int start, int length) throws SAXException {
    // El texto puede venir fragmentado, por eso se acumula o se filtra
    String texto = String.valueOf(ch, start, length).trim();
    if (!texto.isEmpty()) {
        System.out.println("Texto dentro de etiqueta: " + texto);
    }
}

// Se ejecuta al encontrar una etiqueta de cierre
@Override
public void endElement(String uri, String localName, String name) throws
SAXException {
    System.out.println("Fin de etiqueta:");
    System.out.println("\tNamespace URI: " + uri);
    System.out.println("\tNombre local: " + localName);
    System.out.println("\tNombre con prefijo (qName): " + name);
}

```

} Manejo de errores SAX

La interfaz ErrorHandler es la **interface básica para manejar errores SAX**. Ya se vió en el apartado de validación de un documento XML mediante DOM, ya que también es utilizada por DOM para el control de errores en el análisis de un documento XML.

Como ya se comentó, para tratar **los errores de procesamiento de XML**, se debe utilizar la interfaz **ErrorHandler**, en lugar de lanzar una excepción.

Después se debe registrar una instancia con el parser XML mediante el método **setErrorHandler**.

Ejemplo:

```
package leerficheroxmlsax;
```

```
import java.io.FileInputStream;
import java.io.IOException;
import org.xml.sax.*;
import org.xml.sax.helpers.XMLReaderFactory;
public class LeerFicheroXMLSAX {
public static void main(String[] args) {
try {
    // Creamos la fábrica de parsers SAX (JAXP)
    SAXParserFactory factory = SAXParserFactory.newInstance();
    // Creamos el parser SAX
    SAXParser saxParser = factory.newSAXParser();

    // Obtenemos el XMLReader desde el SAXParser
    XMLReader Analizador = saxParser.getXMLReader();

    // Añadimos nuestro manejador al analizador
    Analizador.setContentHandler(new Manejador());
    //añadimos el manejador de errores
    Analizador.setErrorHandler( new ErrorHandler() {
        @Override
        public void warning(SAXParseException e) throws SAXException {
            System.out.println("WARNING : " + e.getMessage())
        }
        @Override
        public void error(SAXParseException e) throws SAXException {
            System.out.println("ERROR : " + e.getMessage());
            //si no se quiere que se siga procesando se lanza esta excepción
            throw e;
        }
        @Override
        public void fatalError(SAXParseException e) throws SAXException {
            System.out.println("FATAL : " + e.getMessage());
            throw e;
        }
    });
    // Procesamos el xml de ejemplo
    Analizador.parse(new InputSource(new FileInputStream("Actores1.xml")));
} catch (SAXException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}
```

Ejemplo de XML

```
<?xml version="1.0" encoding="UTF-8"?>
<personas>
    <persona id="1">
        <nombre>Ana</nombre>
        <edad>30</edad>
    </persona>
    <persona id="2">
        <nombre>Luis</nombre>
        <edad>25</edad>
    </persona>
</personas>
```

Manejador SAX: ManejadorPersonas.java

```
public class ManejadorPersonas extends DefaultHandler {
    private String contenido = "";
    private String idActual = "";

    @Override
    public void startDocument() throws SAXException {
        System.out.println("Inicio del documento XML");
    }

    @Override
    public void endDocument() throws SAXException {
        System.out.println("Fin del documento XML");
    }

    @Override
    public void startElement(String uri, String localName, String qName,
    Attributes attributes) throws SAXException {
        contenido = "";

        if (qName.equals("persona")) {
            idActual = attributes.getValue("id");
            System.out.println("\nPersona ID: " + idActual);
        }
    }

    @Override
    public void characters(char[] ch, int start, int length) throws SAXException {
        contenido += new String(ch, start, length).trim();
    }

    @Override
    public void endElement(String uri, String localName, String qName) throws SAXException {
        switch (qName) {
            case "nombre" -> System.out.println("Nombre: " + contenido);
            case "edad" -> System.out.println("Edad: " + contenido);
        }
    }
}
```