Ciclo Formativo DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Módulo 6 Acceso a Datos

Unidad Formativa 1

INICIATIVA Y COORDINACIÓN
IFP Innovación en Formación Profesional

Supervisión editorial y metodológica:
Departamento de Producto de Planeta Formación

Supervisión técnica y pedagógica:
Departamento de Enseñanza de **IFP** Innovación en Formación Profesional

Módulo: Acceso a Datos
UF 1 / Desarrollo de Aplicaciones Multiplataforma

MÓDULO 6

Unidad Formativa 1

Acceso a Datos

Esquema de contenido

OBJETIVOS

- Utilizar clases para la gestión de ficheros y directorios.
- Valorar las ventajas y los inconvenientes de las distintas formas de acceso.
- Utilizar clases para recuperar información almacenada en un fichero XML.
- Utilizar clases para convertir a otro formato información contenida en un fichero XML.
- Gestionar excepciones en el acceso a ficheros.

En este capítulo se muestran distintas alternativas al trabajar con ficheros, se tiene que tener en cuenta que las soluciones dadas no son únicas, peros si pueden considerarse un referente dentro de las soluciones actuales al trabajar con ficheros. Aunque las diferentes soluciones que se muestran, son bajo la perspectiva de Java, la mayoría de entornos de programación dan soporte a estas mismas alternativas, pero con una sintaxis distinta.

INTRODUCCIÓN

Antes de empezar con el temario del curso tiene que quedar claro que se entiende por acceso a datos.

Según el IEEE (*Institute of Electrical and Electronics Engineers*) software es:

"El conjunto de los programas de cómputo, procedimientos, reglas, documentación y **datos** asociados que forman parte de las operaciones de un sistema de computación".

Una definición más conocida asumida por el área de la ingeniería del software es:

"El software es programas + **datos**"

Como se puede ver, en las dos definiciones se ha resaltado la palabra *datos*. Los datos son lo que necesitan los programas para realizar el objetivo para el que han sido desarrollados.

Para dar una explicación de los datos, antes debe mencionarse que existen dos tipos de datos, *persistentes* y *no persistentes*.

- Los datos persistentes son los que necesitan ser guardados en un sitio "seguro" para que en posteriores ejecuciones el programa pueda recuperar su estado anterior. Por ejemplo, en un teléfono móvil la agenda con los números de teléfono de los contactos representa un conjunto de datos persistentes, ya que de poco serviría una agenda que pierde sus contactos cada vez que se apaga el teléfono.
- Los datos no persistentes por otro lado son aquellos que no es necesario que el programa guarde entre ejecución y ejecución ya que solo son necesarios durante la ejecución del programa. Por ejemplo, los teléfonos móviles llevan un registro de las aplicaciones que se están ejecutando en un momento dado: agenda, navegador, apps de todo tipo etc. En este caso cuando el dispositivo se apaga estos datos no son necesarios ya que cuando el teléfono se vuelva a iniciar están aplicaciones no estarán cargadas.

Explicados los dos tipos de datos distintos que pueden haber dentro de una aplicación se puede entender que cuando se quiera mantener los datos en distintas ejecuciones del mismo programa, esos datos tendrán que ser persistentes, en el caso contrario no será necesario.

Para conseguir la persistencia de los datos en una aplicación independientemente del sistema operativo que se esté utilizando, se crean bases de datos y sistemas de ficheros (llamados *sistemas de almacenamiento*). Las bases de datos como Oracle y MySQL son los sistemas más comunes para guardar la información, pero no son los únicos. Existen tecnologías distintas a las bases de datos relacionales que permiten la persistencia y se basan en distintos modelos.

El objetivo de este temario es dar a conocer conocimientos sobre los distintos sistemas de almacenamiento destinados a la persistencia de datos y enseñar de forma práctica (utilizando Java) como las aplicaciones informáticas acceden a esos datos, los recuperan y los integran. Ficheros XML, base s de datos orientadas a objetos, bases de datos objeto-relacionales, bases de datos XML nativas, acceso a datos con conectores JDBC, frameworks de mapeo objeto-relacional etc.

Para seguir este temario sin problemas se debe tener en cuenta que los siguientes puntos son los conocimientos mínimos para poder seguirlo:

1. Se debe saber generar y manejar ficheros XML y esquemas XML (lenguajes de marcas y sistemas de gestión de información).
2. Se deben manejar sistemas gestores de bases de datos relacionales y SQL como lenguaje de consulta y modificación.
3. Se debe manejar Java, programación básica y entornos de desarrollo.

1. FORMAS DE ACCESO A UN FICHERO. CLASES ASOCIADAS

En Java en el paquete java.io, existen varias clases que facilitan trabajar con ficheros desde distintas perspectivas, los tipos de fichero se pueden clasificar en dos grandes tipos, según el tipo de contenido y según el modo de acceso:

- Según el tipo de contenido:
  - Ficheros de caracteres (o de texto): Son los creados exclusivamente con caracteres, por lo tanto, pueden ser editados con cualquier editor de texto.
  - Ficheros binarios (o de bytes): son ficheros, los cuáles los bytes que contienen representan otra información: imágenes, audio, video etc. Este tipo de ficheros solo puede ser abierto por aplicaciones concretas para tratar con ellos, que saben cómo están estructurados los bytes dentro del fichero, y así poder reproducir el contenido.
  - Según el modo de acceso:
  - Ficheros secuenciales: en estos ficheros la información se almacena de forma secuencial, por lo tanto, para acceder al byte n previamente se tiene que haber pasado por el byte n-1
  - Ficheros aleatorios: de forma contraria a los secuenciales se puede acceder a una posición concretamente del fichero sin tener que pasar por los anteriores.

En los siguientes apartados se muestran diferentes formas para el acceso a ficheros de Java, sin embargo, independientemente del modo en el que se accede al fichero, Java utiliza la clase que se encuentra dentro del paquete java.io, esta clase representa una clase o un directorio del sistema de ficheros.

Dentro de la clase File Java dispone de una serie de métodos para obtener información sobre el fichero o directorio. A continuación, se muestra un ejemplo de cómo se instancia un objeto de la clase:

Ejemplo

```
File f = new File (“ejemplo\\fichero.xml”);
```

Si aplicamos los siguientes métodos sobre el objeto File:

```
System.out.println(“Nombre “: + f.getName());
System.out.println(“Directorio padre “: + f.getParent());
System.out.println(“Ruta alternativa “: + f.getPath());
System.out.println(“Ruta absoluta “: + f.getAbsolutePath());
```

El resultado será:

```
Nombre: fichero.xml
Directorio padre: ejemplo
Ruta alternativa: ejemplo.fichero.xml
Ruta absoluta: c:\manejoficheros\ejemplo\fichero.xml
```

2. GESTIÓN DE FLUJO DE DATOS

Para comunicar un programa con un origen o destino se usan flujos de información entre el programa y el fichero. Un flujo es el objeto que hace de intermediario entre el programa y el origen o destino de la información. La abstracción aplicada por los flujos permite que cuando el programador accede a la información solo se preocupe de trabajar con el objeto que proporciona el flujo de información sin importar el origen o destino.

Por ejemplo, Java ofrece la clase FileReader, donde se implementa un flujo de caracteres que lee de un fichero de texto.

Otro ejemplo es la clase FileWriter, esta clase implementa un flujo de caracteres que se imprimen en un fichero de texto.

Por otro lado, si lo que se desea es acceder a ficheros que contienen información binaria en vez de texto, Java proporciona otras clases para interactuar con ellos. FileInputStream y FileOutputStream.

Las clases anteriores son para un acceso secuencial, si lo que se desea es un acceso aleatorio Java contiene la clase RandomAccessFile, que permite acceder directamente a cualquier posición del fichero.

Todos los métodos de interactuar con ficheros se hacen prácticamente de la misma manera:

1. Para leer: se abre un flujo de desde un fichero. Mientras haya información se lee la información. Una vez terminado se cierra el flujo.
2. Para escribir: se abre el flujo de desde un fichero. Mientras haya información se escribe en el flujo. Una vez terminado se cierra el flujo.

Los problemas de gestión del archivo, por ejemplo, como se escribe en binario o como se hace cuando el acceso es secuencial o aleatorio, Java se encarga de manipularlo. El programador solo tiene que trabajar con los datos que lee y escribe.

2.1. CLASE FILEWRITER

Esta clase permite escribir caracteres en un fichero de modo secuencial. Esta clase hereda los métodos necesarios para ello de la clase Writer. Los constructores principales son:

FileWriter (String ruta, boolean añadir)

FileWriter (File fichero)

El parámetro ruta indica la localización del archivo en el sistema operativo. El parámetro añadir igual a true indica que el fichero se usa para añadir datos a un fichero ya existente.

El método más popular de FileWriter, heredado de Writer, es el método write() que puede aceptar un array de caracteres (buffer), pero también puede aceptar un string:

Public void write(String str) throws IOException

2.2. CLASE FILEREADER

Esta clase permite leer caracteres de un fichero de modo secuencial. Esta clase hereda los métodos de la clase Reader. Los constructores principales son:

FileReader (String ruta)

FileReader (File fichero)

Para abrir un fichero se puede utilizar un objeto de tipo File o una ruta de directorios, la forma más común es utilizar el método read() el cual solo acepta un array de caracteres.

Public int read(char[] cbuf) throws IOException

2.3. CLASE FILEOUTPUTSTREAM

Esta clase permite escribir bytes en un fichero de forma secuencial, el fichero puede ser abierto vacío o preparado para añadirle datos a los que ya contiene. Todas las escrituras con esta clase se hacen a través de un buffer (array de bytes).

Public void write (byte[] b) throws IOException

2.4. CLASE FILEINPUTSTREAM

Esta clase permite leer bytes de un fichero de forma secuencial, sus constructores son idénticos a los de la clase FileReader. El método más utilizado en esta clase es el método read()

Public int read(byte[] cbuf) throws IOException

2.5. RANDOMACCESSFILE

Este flujo permite acceder directamente a cualquier posición dentro del fichero. Proporciona dos constructores básicos.

RandomAccessFile (String ruta, String modo)

RandomAccessFile (File fichero, String modo)

El parámetro modo específica para qué se abre el archivo, por ejemplo, si el modo es "r" el fichero se abrirá en modo lectura, sin embargo, si el parámetro es "rw" el fichero se abrirá en lectura y escritura.

2.6. EJEMPLO DE USO DE FLUJOS

Para resumir, el uso de flujos comparte los siguientes pasos:

Se abre el fichero: se crea un objeto de la clase correspondiente que al tipo de fichero que se quiere manejar y el modo de acceso. Por norma general la sintaxis es:

TipoDeFichero obj = new TipoDeFichero(ruta);

El parámetro ruta puede ser tanto una cadena de texto que contiene la ruta de acceso o un objeto de tipo File.

La sintaxis anterior es válida para todos los tipos de objetos vistos anteriormente excepto para los objetos del tipo RandomAccessFile, para este último es necesario indicar el modo en el que se abre el fichero: "r","rw":

RandomAccessFile obj = new RandomAccessFile(ruta,modo);

Se utiliza el fichero: se utilizan los métodos específicos de cada clase para leer o escribir.

Gestión de excepciones: todos los métodos que utilicen java.io deben tener en su definición una cláusula throws IOException.

Se cierra el fichero y se destruye el objeto: una vez se ha terminado de trabajar con el fichero lo recomendable es cerrarlo usando el método close() de cada clase.

obj.close();

El siguiente código muestra un ejemplo de acceso a un fichero siguiendo los pasos anteriores.

Ejemplo

El código escribe en un fichero de texto llamado libros.xml una cadena de texto con un fragmento de XML:

```
<Libros> <Libro>>Titulo> El Capote</Titulo></Libro></Libros>
String texto = "<Libros><Libro><Titulo> El Capote </Titulo></Libro></Libros>";
//Guarda en nombre el nombre del archivo que se creará.
String nombre = "libros.xml";

try{
//Se crea un nuevo objeto FileWriter
FileWriter fichero = new FileWriter(nombre);
//Se escribe el fichero
fichero.write(texto + "\r\n");
//Se cierra el fichero
fichero.close();
}catch(IOException ex){
    System.out.println("error al acceder al fichero");
    }
}
```

3. TRABAJO CON FICHEROS XML (EXTENDED MARKUP LANGUAGE)

El lenguaje de marcas extendido (eXtended Markup Language [XML]) ofrece la posibilidad de representar la información de forma neutra, independiente del lenguaje de programación y del sistema operativo empleado. Gracias a las posibilidades ofrecidas por XML se han podido diseñar muchas aplicaciones, un ejemplo son los servicios web.

Desde un punto de vista a "bajo nivel", un documento XML no es otra cosa que un fichero de texto. Realmente nada impide utilizar librerías de acceso a ficheros, para acceder y manipular ficheros XML.

Sin embargo, desde un punto de vista a "alto nivel", un documento XML no es un fichero de texto común. Su uso intensivo en el desarrollo de aplicaciones hace necesarias herramientas específicas para acceder y manipular este tipo de archivos de manera eficiente. En resumen, estas herramientas permiten manejar los documentos XML de forma simple y sin cargar innecesariamente al sistema.

Las herramientas que leen el lenguaje XML y comprueban si el documento es válido sintácticamente se denominan analizadores sintéticos o parsers. Un parser es un módulo, biblioteca o programa encargado de transformar el fichero de texto en un modelo interno que optimiza su acceso. Para XML existen un gran número de parsers disponibles para dos de los modelos más conocidos: DOM y SAX. Estos parsers tienen implementaciones para la gran mayoría de lenguajes.

Existen dos tipos de herramientas:

- Herramientas que validan los documentos XML. Estas comprueban que el documento XML al que se quiere acceder está bien formado, según la definición de XML y, además que es válido con respecto a un esquema XML.
- Herramientas que no validan los documentos XML. Estas comprueban que el documento está bien formado según la definición XML, pero no necesitan de un esquema asociado para comprobar si es válido con respecto a ese esquema.

4. ACCESO A DATOS CON DOM (DOCUMENT OBJECT MODEL)

DOM es una interfaz de programación que permite analizar y manipular dinámicamente y de manera global el contenido, el estilo y la estructura de un documento.

DOM está definido en tres niveles:

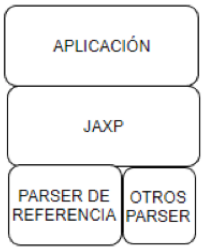
1. Describe la funcionalidad básica de la interfaz DOM, así como el modo de navegar por el modelo en un documento general.
2. Estudia tipos de documentos específicos.
3. Amplía las funcionalidades de la interfaz para trabajar con tipos de documentos específicos.

Para trabajar con un documento XML primero se almacena en memoria en forma de árbol con nodos padre, nodos hijos y nodos finales que son aquellos que no tienen descendientes. En este modelo todas las estructuras de datos del documento XML se transforman en algún tipo de nodo, y luego esos nodos se organizan jerárquicamente en forma de árbol para representar la estructura descrita por el documento XML. Una vez creada en memoria esta estructura, los métodos de DOM permiten recorrer los diferentes nodos del árbol y analizar a qué tipo particular pertenecen. En función del tipo de nodo, la interfaz ofrece una serie de funcionalidades u otras para poder trabajar con la información que contienen.

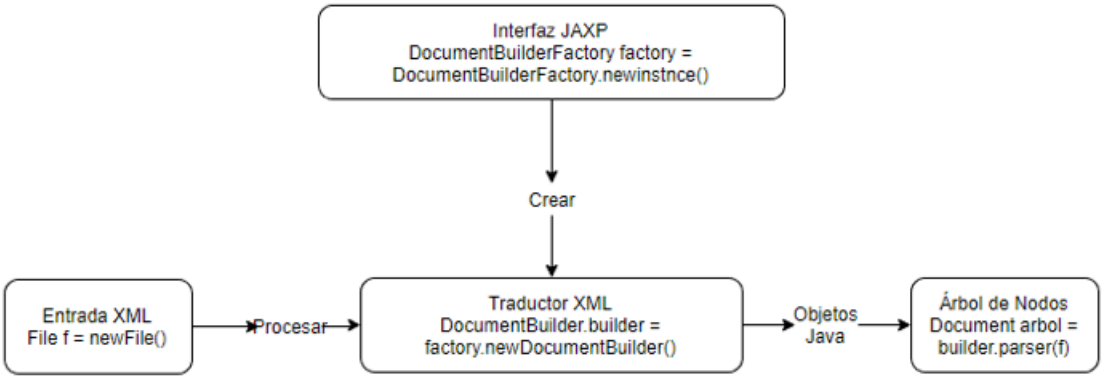
4.1. EJEMPLO DE USO DE FLUJOS

DOM ofrece una manera de acceder a documentos XML tanto para ser leído como para ser modificado. Su único inconveniente es que el árbol DOM se crea todo en memoria principal, por lo que, si el documento XML es muy grande, la creación y manipulación de DOM sería intratable.

Actualmente la propuesta principal para trabajar con Java y DOM se basa en el uso de JAXP (Java API for XML Processing).



La siguiente figura muestra un esquema que relaciona JAXP con el acceso a DOM. Desde la interfaz se crea un DocumentBuilderFactory. A partir de esa factoria se crea un DocumentBuilder que permitirá cargar en él la estructura del árbol DOM.



4.2. ABRIR DOM DESDE JAVA

Para abrir un documento XML desde Java y crear un árbol DOM con él se utilizan las clases DocumentBuilderFactory y DocumentBuilder, que pertenecen al paquete javax.xml.parsers, y Document, que representa un documento en DOM y pertenece al paquete org.w3c.dom. Aunque existen otras posibilidades, en el ejemplo mostrado seguidamente se usa un objeto de la clase File para indicar la localización del archivo XML.

4.3. RECORRER UN ÁRBOL DOM

En el código anterior lo más destacable es: Que el programador sabe que el elemento «Libro> solo tiene un atributo, por lo que accede directamente a su valor (n.getAttributes(). item(0).getNode Value()). Una vez detectado que se está en el nodo de tipo Elemento (que puede ser el título o el autor) entonces se obtiene el hijo de este (tipo #text) y se consulta su valor (ntemp.get Child Nodes().item (0).getNode Value().

4.4. MODIFICAR Y SERIALIZAR

Además de recorrer en modo "solo lectura" un árbol DOM, este también puede ser modificado y guardado en un fichero para hacerlo persistente. En esta sección se muestra un código para añadir un nuevo elemento a un árbol DOM y luego guardar todo el árbol en un documento XML. Es importante destacar lo fácil que es realizar modificaciones con la librería DOM. Si esto mismo se quisiera hacer accediendo a XML como si fuera un fichero de texto "normal" (usando File Writer, por ejemplo), el código necesario sería mucho mayor y el rendimiento en ejecución bastante más bajo.

Ejemplo

```
public int annadir DOM (Document doc, String titulo, String autor, String anno) {
 try {
// Se crea un nodo tipo Element con nombre titulo (KTitulo>)
No de ntitulo=doc.createElement (YTitulo");
// Se crea un nodo tipo texto con el título del libro
Node intitulo_text=doc.createTextNode (titulo);
// Se añade el nodo de texto con el título como hijo del elemento Titulo ntitulo . appendChild (ntitulo_text) ;
// Se hace lo mismo que con título a autor (KAutor») Node nautor=doc. createElement (YAutor") ;
Node nautor text=doc.createTextNode(autor);
nautor. appendChild (nautor text) ;
// Se crea un nodo de tipo elemento (< libro>) Node nlibro=doc. createElement (Libro")
 //Al nuevo nodo libro se le añade un atributo publicado en ((Element) nlibro). setAttribute (Ypublicado_en", anno ) ;
// Se añade a libro el nodo autor y título creados antes nlibro. appendChild (ntitulo); nilibro . appendChild (nautor);
//Finalmente, se obtiene el primer nodo del documento y a él se le añade como hijo el nodo libro que ya tiene colgando todos sus
//hijos y atributos creados antes.
Node raiz=doc. getChild Nodes (). item (0);

raiz.appendChild (n.libro) ;

return 0; } catch (Exception e) {
    e - print StackTrace () ; return -1;
}
```

Revisando el código anterior se puede apreciar que para añadir nuevos nodos a un árbol DOM hay que conocer bien cómo de diferente es el modelo DOM con respecto al documento XML original. La prueba es la necesidad de crear nodos de texto que sean descendientes de los nodos de tipo Elemento para almacenar los valores XML.

Una vez se ha modificado en memoria un árbol DOM, éste puede ser llevado a fichero para lograr la persistencia de los datos. Esto se puede hacer de muchas maneras. Una alternativa concreta se recoge en el siguiente código.

En el ejemplo se usa las clases XMLSerializer y Out Format que están definidas en los paquetes com.sun. org. apache.xml. internal.serialize.Output Format y com.sun.org.apache.xml. internal.serialize.XMLSerializer. Estas clases realizan la labor de serializar un documento XML.

Serializar (en inglés marshalling) es el proceso de convertir el estado de un objeto (DOM en nuestro caso) en un formato que se pueda almacenar. Serializar es, por ejemplo, llevar el estado de un objeto en Java a un fichero o, como en nuestro caso, llevar los objetos que componen un árbol DOM a un fichero XML. El proceso contrario, es decir, pasar el contenido de un fichero a una estructura de objetos, se conoce por unmarshalling.

La clase XMLSerializer se encarga de serializar un árbol DOM y llevarlo a fichero en formato XML bien formado. En este proceso se utiliza la clase Output Format porque permite asignar diferentes propiedades de formato al fichero resultado, por ejemplo que el fichero esté indentado (anglicismo, indent) es decir, que los elementos hijos de otro elemento aparezcan con más tabulación a la derecha para así mejorar la legibilidad del fichero XML.

Ejemplo

```
public int guardar DOMcomo FILE ()
{
try {
//Crea un fichero llamado salida.
Xml File archivo Xml = new File (salida. xml");
//Especifica el formato de salida Output
Format format = new Output Format (doc);
//Especifica que la salida esté indentada. format.
Set Indenting (true);
XMLSeria lizer serialio Zer = new XML Serializier (new FileOutputStream (archivo_xml) , format);
 serializer. serialize (doc) ;
 return 0;
} catch (Exception e) {
return -1;
}
```

Según el código anterior, para serializar un árbol DOM se necesita: Un objeto File que representa al fichero resultado (en el ejemplo será salida.xml). Un objeto Output Format que permite indicar pautas de formato para la salida.

5. ACCESO A DATOS CON SAX (SIMPLE API FOR XML)

SAX (Simple API for XML) es otra tecnología para poder acceder a XML desde lenguajes de programació Aunque SAX tiene el mismo objetivo que DOM esta aborda el problema desde una óptica diferente. Por lo general, usa SAX cuando la información almacenada en los documentos XML es clara, está bien estructurada y no se necesita hacer modificaciones.

Las principales características de SAX son:

SAX ofrece una alternativa para leer documentos XML de manera secuencial. El documento solo se lee una vez. A diferencia de DOM, el programador no se puede mover por el documento a su antojo. Una vez que se abre el documento, se recorre secuencialmente el fichero hasta el final. Cuando llega al final se termina el proceso (parser).

SAX, a diferencia de DOM, no carga el documento en memoria, sino que lo lee directamente desde el fichero.

Esto lo hace especialmente útil cuando el fichero XML es muy grande. SAX sigue los siguientes pasos básicos:

1. Se le dice al parser SAX qué fichero quiere que sea leído de manera secuencial.
2. El documento XML es traducido a una serie de eventos.
3. Los eventos generados pueden controlarse con métodos de control llamados callbacks.
4. Para implementar los callbacks basta con implementar la interfaz ContentHandler (su implementación por defecto es DafaultHandler).

El proceso se puede resumir de la siguiente manera:

- SAX abre un archivo XML y coloca un puntero en al comienzo del mismo.
- Cuando comienza a leer el fichero, el puntero va avanzando secuencialmente.
- Cuando SAX detecta un elemento propio de XML entonces lanza un evento. Un evento puede deberse a:
  - Que SAX haya detectado el comienzo del documento XML.
  - Que se haya detectado el final del documento XML.
  - Que se haya detectado una etiqueta de comienzo de un elemento, por ejemplo «libro».
  - Que se haya detectado una etiqueta de final de un elemento, por ejemplo, K/libro».
  - Que se haya detectado un atributo.
  - Que se haya detectado una cadena de caracteres que puede ser un texto.
  - Que se haya detectado un error (en el documento, de 1/0, etc.).
- Cuando SAX devuelve que ha detectado un evento, entonces este evento puede ser manejado con la clase DefaultHandler (callbacks). Esta clase puede ser extendida y los métodos de esta

clase pueden ser redefinidos (sobrecargados) por el programador para conseguir el efecto deseado cuando SAX detecta los eventos. Por ejemplo, se puede redefinir el método public void startElement(), que es el que se invoca cuando SAX detecta un evento de comienzo de un elemento. Como ejemplo, la redefinición de este método puede consistir en comprobar el nombre del nuevo elemento detectado, y si es uno en concreto entonces sacar por pantalla un mensaje con su contenido. Cuando SAX detecta un evento de error o un final de documento entonces se termina el recorrido.

En las siguientes secciones se muestra el acceso a SAX desde Java.

En el material adicional incluido en este libro se puede encontrar la carpeta AccesoDOM que contiene un proyecto hecho en NetBeans 7.1.2. Este proyecto es una aplicación que muestra el acceso a documentos XML con SAX (DOM y JAXB).

5.1. ABRIR CON SAX DEDE JAVA

Para abrir un documento XML desde Java con SAX se utilizan las clases: SAXParserFactory y SAXParser que pertenecen al paquete javax.xmlparsers. También es necesario extender la clase Default Handler que se encuentra en el paquete orgaml.sax.helpers. Default Handler. Además, en el ejemplo mostrado a continuación se usa la clase File para indicar la localización del archivo XML. Las clases SAXParserFactoryo y SAXParserlo proporcionan el acceso desde JAXP. La clase Default Handlerlo es la clase base que atiende los eventos devueltos por el parser. Esta clase se extiende en las aplicaciones para personalizar el comportamiento del parser cuando se encuentra un elemento XML

Ejemplo

```
public int abrir XML SAX (Manejador:SAX sh, SAXParser parser ) {
try {
    SAXParserFactory factory=SAXParserFactory. newInstance () ;
// Se crea un objeto SAXParser para interpretar el documento XML. par Ser=factory. newSAXParser
() ; // Se crea una instancia del manejador que será el que recorra el documento //XML secuencialmente sh=new ManejadorSAX(); return 0; } catch (Exception e) { e ... print StackTrace () ; return -1;
```

Como se puede entender siguiendo los comentarios del código, primeramente se crean los objetos factory y parser. Esta parte es similar a como se hace con DOM. Una diferencia con DOM es que en SAX se crea una instancia de la clase ManejadorSAX. Esta clase extiende Default Handler y redefine los métodos (callbacks) que atienden a los eventos. En resumen, la preparación de SAX requiere inicializar las siguientes variables, que serán usadas cuando se inicie el proceso de recorrido del fichero XML:

Un objeto parser: en el código la variable se llama parser.

Una instancia de una clase que extienda Default Handler, que en el ejemplo es ManejadorSAX. La variable se llama sh.

5.2. Recorrer XML CON SAX

Para recorrer un documento XML una vez inicializado el parser lo único que se necesita es lanzar el parser. Evidentemente, antes es necesario haber definido la clase que extiende Default Handler (en el ejemplo anterior era ManejadorSAX). Esta clase tiene la lógica de cómo actuar cuando se encuentra algún elemento XML durante el recorrido con SAX (callbacks).

Un ejemplo de clase ManejadorSAX es el siguiente:

Ejemplo

```
class Manejador SAX extends DefaultHandler {

int ultimoelement ;

String cadena resultado= “;

public Manejador SAX () (

ultimoelement=0;

} // Se sobrecarga (redefine) el método startElement

(d Over ride

public void startElement (String uri, String local Name, String gName, Attributes atts) throws SAXException {

if (qName. equals (“Libro”) ) (

cadena resultado=cadena resultado + \Publicado en: « +atts.getValue (atts.getQName (0)) +        w
\n

ultimoelement=1;

} else if (dName. equals ( ‘Titulo”) ) { ultimoelement=2; cadena resultado= cadena resultado + \n “
+”El título es: “; else if (cName. equals (Autor’)) { ultimoelement=3; cadena resultado= cadena
resultado + \n “ +”El autor es:      } //Cuando en este ejemplo se detecta el final de un elemento
Klibro», se pone una línea // discontinua en la salida. (Override

public void end Element (String uri, String localName, String dname) throws SAXException { if
(qName. equals (“Libro”) ) (

cadena resultado = cadena resultado + \n ----------

(Override public void characters (char) ch, int start, int length) throws SAXException { if (ultimoe-
lement == 2) { for (int i=start; i3length +start; i----) cadena_resultado=cadena_resultado+ch [i] ;

} else if (ultimoelement == 3) { for (int i=start; iClength-start; i----) ; ]cadena_resultado= cadena_
resultado+ch [i   –
```

Esta clase extiende el método startElement, endElement y characters. Estos métodos (callbacks) se invocan cuando durante el recorrido del documento XML, se detecta un evento de comienzo de elemento, final de elemento o cadena de caracteres. En el ejemplo, cada método realiza lo siguiente:

- startElement(): cuando se detecta con SAX un evento de comienzo de un elemento, entonces SAX invoca a este método. Lo que hace es comprobar de qué tipo de elemento se trata.

- Si es <Libro> entonces saca el valor de su atributo y lo concatena con una cadena (cadena resultado) que tendrá toda la salida después de recorrer todo el documento.
- Si es <Titulo> entonces a cadena resultado se le concatena el texto "El título es:".
- Si es <Autor> entonces a cadena resultado se le concatena el texto "El autor es:".
- Si es otro tipo de elemento no hará nada.
- endElement(): cuando se detecta con SAX un evento de final de un elemento, entonces SAX invoca a este le concatena el texto "\n ----------";
- characters(): cuando se detecta con SAX un evento de detección de cadena de texto, entonces SAX invoca este método. El método lo que hace es concatenar a cadena resultado cada uno de los caracteres de la cadena detectada.

6. ACCESO A DATOS CON JAXB (BINDING)

De las alternativas vistas en las secciones anteriores para acceder a documentos XML desde Java, JAXB (Java Architecture forXML Binding) es la más potentes y actual de las tres. JAXB (no confundir con la interfaz de acceso JAXP) es una librería de (un)-marshalling. El concepto de serialización o marshalling, que ha sido introducido en la Sección 1.44, es el proceso de almacenar un conjunto de objetos en un fichero. Unmarshalling es justo el proceso contrario: convertir en objetos el contenido de un fichero. Para el caso concreto de XML y Java, unmarshalling es convertir el contenido de un archivo XML en una estructura de objetos Java.

De manera resumida, JAXB convierte el contenido de un documento XML en una estructura de clases Java:

. El documento XML debe tener un esquema XML asociado (fichero.xsd), por tanto, el contenido del XML debe ser válido con respecto a ese esquema.

JAXB crea la estructura de clases que albergará el contenido del XML en base a su esquema. El esquema es la referencia de JAXB para saber la estructura de las clases que contendrán el documento XML.

Una vez JAXB crea en tiempo de diseño (no durante la ejecución) la estructura de clases, el proceso de unmarshalling (creación de objetos de las clases creadas con el contenido del XML) y marshalling (almacenaje de los objetos como un documento XML) es sencillo y rápido, y se puede hacer en tiempo de ejecución.

JAXB es capaz de obtener de este esquema una estructura de clases que le dé soporte en Java. De manera simplificada, JAXB obtendría las siguientes clases Java:

```
public class Libros {
protected List<Libros . Libro > libro ; public List<Libros . Libro> getLibro ( ) { if (libro == null) { libro = new ArrayList<Libros. Libro> () ;
} return this . libro;
}
public static class Libro {
protected String titulo;
protected String autor;
protected String publicadoEn;
public String getTitulo () { return titulo;
public void setTitulo (String value) { this. titulo = value;
}
public String getAutor () { return aut Or ;
public void setAutor (String Value) { this . autor = Value;
}
public String getPublicadoEn() { return publicado En ;
public void setPublicadoEn (String value) { this . publicado En = value;
}
}
```

6.1. ¿COMO CREAR CLASES JAVA DE ESQUEMAS XML?

Partiendo de un esquema XML como el mostrado en la Figura 1.5, el proceso para crear la estructura de clases Java que le de soporte es muy sencillo: Directamente con un JDK de Java

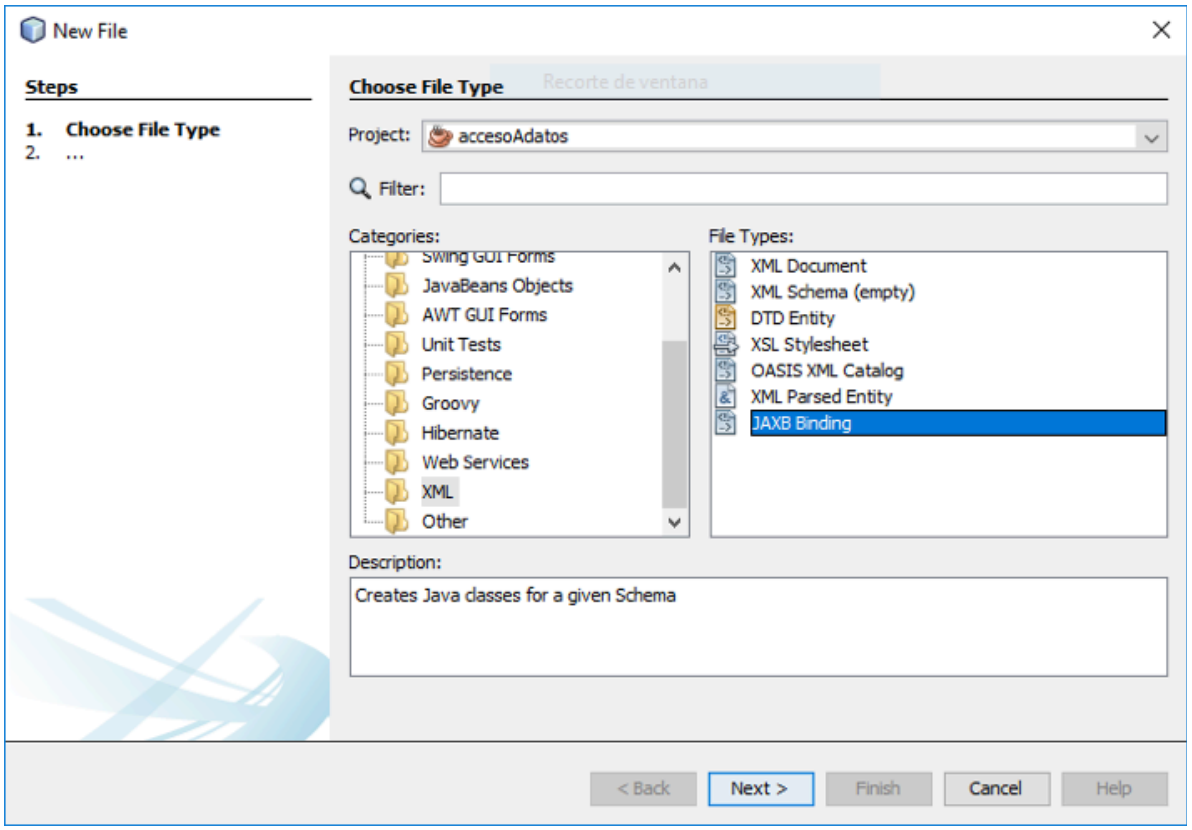
Con JDK 1.7.0 se pueden obtener las clases asociadas a un esquema XML con la aplicación xjc. Para ello, solo es necesario pasar al programa el esquema XML que se quiere emplear en la ejecución. Por ejemplo, suponiendo que el ejemplo de la Figura 1.5 es un fichero llamado Libros Esquema.xsd, la creación de las clases que le den soporte en JAXB sería con el comando:

Xjc LibrosFsquema.xsd

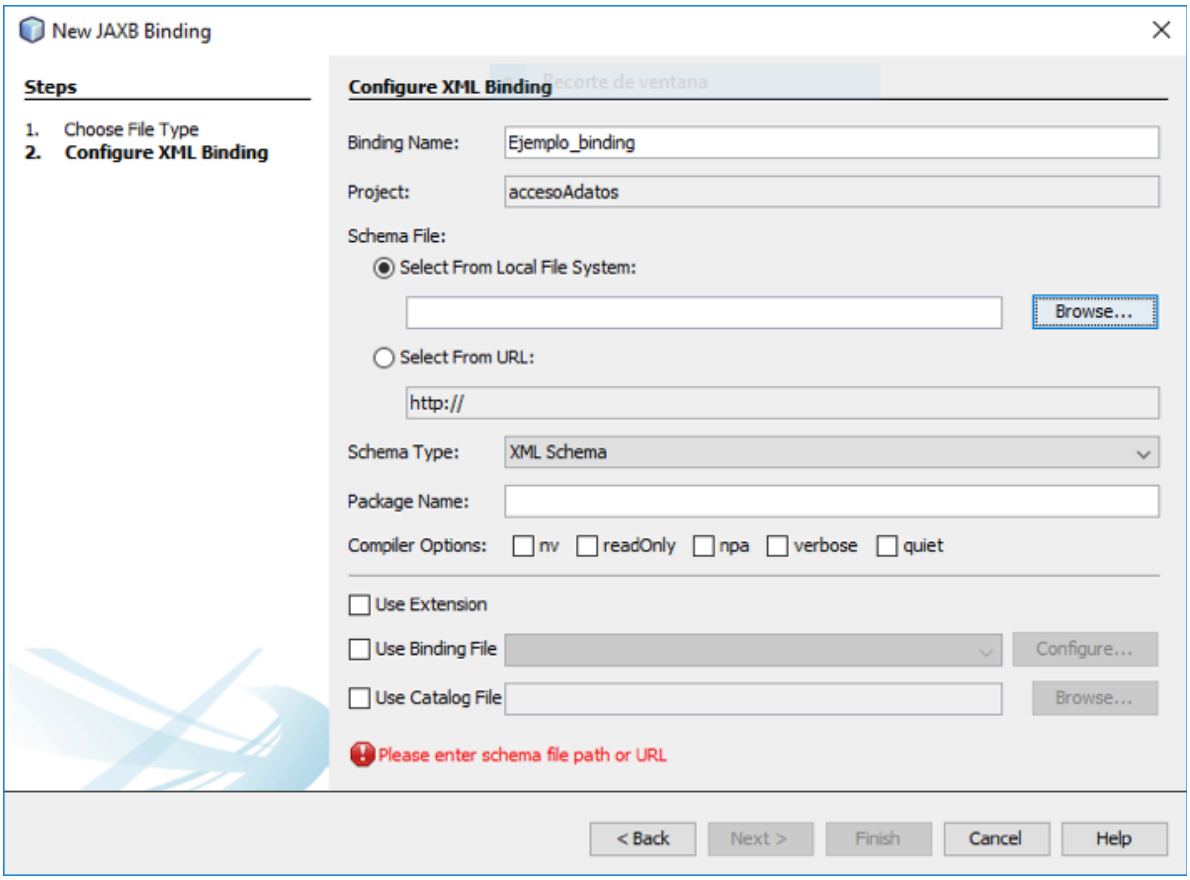
Usando el IDE NetBeans

Con el IDE NetBeans 7.1.2 se pueden obtener las clases asociadas a un esquema XML. Para ello, solo hay que seguir los siguientes pasos:

1. Añadir el fichero con el esquema XML al proyecto en el que se quiere usar JAXB.
2. Seleccionar Archivo->Nuevo Fichero para añadir un nuevo elemento al proyecto. En la ventana que aparece para indicar el tipo de fichero que se quiere añadir seleccionar XML->JAXB Binding. La Figura 1.6 muestra la ventana con esas opciones seleccionadas.



3. Los campos más importantes son la localización del esquema XML sobre el que se crearán las clases Java y el paquete en el que se guardarán las clases nuevas creadas.



4. Una vez rellenados esos datos se crea una nueva carpeta en el árbol del proyecto con el paquete y dentro de las clases que enlazan con el esquema XML. Esas clases ya creadas se pueden utilizar en el proyecto abriendo el enlace JAXB y cargando el documento XML seleccionado en esas estructuras de objetos. Las siguientes secciones muestran el proceso con código.

6.2. ABRIR XML CON JAXB

Un documento XML con JAXB se abre utilizando las clases: JAXBContext y Unmarshaller que pertenencen al paquete javax.xml.bind.*.

- La clase JAXBContext crea un contexto con una nueva iinstancia de JAXB para la clase principal obtenida del esquema XML.
- La clase Unmarshaler se utiliza para pbtener del documento XML los datos que son almacenados en la estructura de objetos de Java.

Para poder abrir un documento XML con estas clases es necesario que previamente esté creada la estructura de clases a partir de un esquema XML.

5.

7. PROCESAMINETO DE XML: XPATH(XML PATH LANGUAGE)

La alternativa más sencilla para consultar información dentro de un documento XML es mediante el uso de XPath (XML Path Language), una especificación de la W3C para la consulta de XML. Con XPath se puede seleccionar y hacer referencia a texto, elementos, atributos y cualquier otra información contenida dentro de un fichero XML. En 1999, W3C publicó la primera recomendación de XPath (XPath 1.0), y en 2010 se publicó la segunda recomendación XPath 2.0.19

En la Sección 5.8.1 se muestra XQuery como el lenguaje más destacado y potente actualmente para la consulta de XML en bases de datos XML nativas. XQuery está basado en XPath 2.0, por tanto, es necesario conocer las nociones básicas de XPath para entender el funcionamiento de XQuery.

7.1. LO BÁSICO DE XPATH

XPath comienza con la noción contexto actual. El contexto actual define el conjunto de nodos sobre los cuales se consultará con expresiones XPath. En general, existen cuatro alternativas para determinar el contexto actual para una consulta XPath. Estas alternativas son las siguientes:

- usa el nodo en el que se encuentra actualmente como contexto actual.
- usa la raíz del documento XML como contexto actual.
- usa la jerarquía completa del documento XML desde el nodo actual como contexto actual.
- usa el documento completo como contexto actual.

Para seleccionar elementos de un XML se realizan avances hacia abajo dentro de la jerarquía del documento. Por ejemplo, la siguiente expresión XPath selecciona todos los elementos Autor del documento XML Libros.

/Libros/Libro/Autor

Si se desea obtener todos los elementos Autor del documento se puede usar la siguiente expresión:

//Autor

De esta forma no es necesario dar la trayectoria completa.

También se pueden usar comodines en cualquier nivel del árbol. Así, por ejemplo, la siguiente expresión selecciona todos los nodos Autor que son nietos de Libros:

/Libros/ * / Autor Las expresiones XPath seleccionan un conjunto de elementos, no un elemento simple. Por supuesto, el conjunto puede tener un único miembro, o no tener miembros.

Para identificar un conjunto de atributos se usa el carácter G) delante del nombre del atributo. Por ejemplo, la siguiente expresión selecciona todos los atributos publicado en de un elemento Libro:

/Libros/Libro / @publicado_en

También se pueden seleccionar múltiples atributos con el operador @*. Para seleccionar todos los atributos del elemento Libro en cualquier lugar del documento se usa la siguiente expresión:

//Libro/@*

Además, XPath ofrece la posibilidad de hacer predicados para concretar los nodos deseados dentro del árbol XML.

7.2. XPATH DESDE JAVA

En Java existen librerías que permiten la ejecución de consultas XPath sobre documentos XML. En esta sección se muestra un ejemplo de cómo se puede abrir un documento XML y ejecutar consultas XPath sobre él usando DOM. Las clases necesarias para ejecutar consultas XPath son:

1. XPathFactory, disponible en el paquete javax.xml.xpath.°: esta clase contiene un método compile(), que comprueba si la sintaxis de una consulta XPath es correcta y crea una expresión XPath (XPath Expression).
2. XPath Expressiono, disponible también en el paquete javax.xml.xpath.°: esta clase contiene un método evaluate() que ejecuta un XPath.
3. DocumentBuilderFactory, disponible en el paquete javax.xmlparsers.o, y Document del paquete orguv3c.xml.*. Ambas clases han sido trabajadas con DOM en la Sección 1.4.2.
4.

8. CONCLUSIONES Y PROPUESTAS PARA AMPLIAR

En este capítulo se han mostrado diferentes formas de acceso a ficheros. Lejos de pretender profundizar en todas las posibilidades de cada acceso, lo que se ha buscado ha sido dar una visión global sobre el tratamiento de ficheros con Java.

El lector interesado en profundizar en las tecnologías expuestas en el capítulo puede hacerlo en las siguientes líneas de trabajo.

1. Trabajar más en profundidad los flujos para el tratamiento de archivos en Java. Para ello, el título Java 2. Curso de programación de Fco. Javier Ceballos, de la editorial RA-MA, es una buena referencia.

2. Conocer otros modos de acceso a documentos XML, como jDOM que ofrece un modelo más natural para trabajar con XML desde Java que el ofrecido por DOM. Es una alternativa diferente al DOM de W3C visto en el capítulo, y muy aceptada en el terreno profesional.

En cualquier caso, un amplio conocimiento en el manejo de ficheros y en el acceso a XML, junto con todo lo que XML ofrece (esquemas XML, herramientas de validación de documentos, XPath, etc.) es necesario para desarrollar aplicaciones de acceso a datos solventes, así como para entender parte de los capítulos siguientes.

Actividad

1.1. Modifica el código del punto 1.2.6 para escribir un fichero XML bien formado y leerlo posteriormente, mostrando la salida por pantalla

1.2. Utiliza el código del punto 1.4.4 para hacer un método que permita modificar los valores de un libro:

a) A la función se le pasa el título de un libro y se debe cambiar por un nuevo título que también se le pasa como parámetro.

b) El resultado debe ser guardado en un nuevo documento XML llamado "modificación.xml"

1.3. Modifica el código del punto 1.5.2 para que:

Cuando SAX encuentre un elemento <Libros> aparezca un mensaje que diga "Se van a mostrar los libros de este documento"

1.4. Sobre el código disponible en la carpeta AccesoDOM que contiene un proyecto hecho en NetBeans:

a) Modifica el esquema XML llamado LibrosEsquema.xsd para que permita un nuevo elemento <editorial>. Comprueba que un documento XML con un nuevo elemento editorial es válido para ese nuevo esquema creado.

b) Crea las clases JAXB asociadas con ese esquema nuevo creado.

c) Modifica la aplicación y comprueba que funciona mostrando todos los elementos de un libro (incluido la editorial)

1.5. Sobre el código disponible en la carpeta Acceso_XPath, que contiene un proyecto hecho en NetBeans se propone:

Modificar el código para que se puedan ejecutar consultas que devuelvan objetos de tipo Libro, como por ejemplo: /Libro/Libro.

Índice