Universidad Autónoma de Querétaro
Facultad de Informatica
Maestría en Ingeniería de Software Distribuido

PROCESAMIENTO DE XML A TRAVÉS DE JDOM DE JAVA

TESIS

Que como parte de los requisitos para obtener el grado de

Maestro en "Ingeniería de Software Distribuido"

Presenta:
LUZ MARÍA DORANTES HERNÁNDEZ

Dirigido por:
MCC. MA. TERESA GARCÍA RAMÍREZ

SINODALES

M.C.C. Ma. TERESA GARCÍA RAMÍREZ
Presidente
Firma

M.C. CLAUDIA MORALES CASTRO
Secretaria
Firma

DRA. ARMIDA GÓNZALEZ LORENCE
Vocal
Firma

M.C.C. ROSA MARÍA ROMERO GONZÁLEZ
Suplente
Firma

M.S.I. GERARDO RODRÍGUEZ ROJANO
Suplente
Firma

MC. ALEJANDRO SANTOYO RODRÍGUEZ
Director de la Facultad

DR. LUIS GERARDO HERNÁNDEZ SANDOVAL
Director de Investigación y Posgrado

Centro Universitario
Santiago de Querétaro, Qro., Febrero 2007
México

RESUMEN.

En el ámbito de la comunicación de datos, la tecnología computacional ofrece un avance importante, en la implementación del uso del lenguaje de marcas XML (Lenguaje Extensible de Marcado cuyas siglas en inglés son Extensible Markup Language), como alternativa mejorada para generar la misma información en diferentes formatos, esta flexibilidad es necesaria en aplicaciones en Internet, en dispositivos móviles, entre otros. Una posibilidad, que rompe esquemas, es la libertad del desarrollador para elaborar su propio lenguaje de marcas y con ello disminuir las posibilidades de error o inconsistencia en la transferencia de datos. Para acceder a la información de un archivo XML, es necesario el uso de parsers (procesadores) como SAX (Simple API for XML), DOM (Document Object Model) y JDOM (Java Document Object Model) que se pueden implementar en java, siendo este un lenguaje de programación que permite ejecutar las aplicaciones en cualquier sistema operativo, independientemente de la arquitectura del equipo computacional, también es un lenguaje orientado a objetos lo cual facilita el desarrollo de sistemas complejos. DOM y JDOM ofrecen mayores beneficios en el procesamiento de XML, ya que ambos crean en memoria una estructura jerárquica de tipo árbol, permitiendo acceder a cualquier dato almacenado, en el momento en que se requiera, permiten hacer actividades de lectura y escritura hacia el documento XML, a diferencia del parser SAX, que por el hecho de ser de solo lectura no permite crear ni actualizar los datos hacia el archivo XML [13]. Por lo anterior se analizan las capacidades y recursos necesarios que requiere cada parser, para el procesamiento de archivos XML. JDOM realiza un procesamiento de datos más rápido, requiere menos recursos de librerías importadas por las aplicaciones, fortalece el reuso de código, por medio de la implementación de herencia y polimorfismo, las cuales no se pueden usar con el parser DOM, que es más dependiente de la herramienta utilizada [5].

(Palabras clave: aplicaciones de Internet, XML, parsers SAX, DOM y JDOM orientado a objetos).

ABSTRACT.

In the communication of information environment the computational technology offers an important advance in the implementation of the use of the mark XML language, as enhanced alternative gives birth generate the same information in different formats, this flexibility is necessary in applications in Internet, in mobile devices between others. A possibility, that breaks schemes, is the freedom of the developer to elaborate his own mark language and with it reducing the possibilities of error or inconsistency in the datum transfer. To gain access to the information of the file XML, is necessary the use of parsers (processors), those of common use and more complete are SAX, DOM and JDOM, since they can be implemented in java, being this a programming language that permits to execute the applications in any operating system, independently of the architecture of the computational equipment, also its a programming language oriented to objects, which facilitates the development of complex systems. DOM and JDOM offer major benefits in the prosecution of XML, since both use memory in a hierarchic structure of type tree, allowing to gain access to any stored fact, at the moment when it is needed, allow to do activities of reading and writing towards the document XML, in contrast to the parser SAX, which for the fact of being of only reading, it allows neither to create or to update the information towards the file XML [13]. For the previous thing, there are analyzed the capacities and necessary resources that the parser needs, for the prosecution of files XML, JDOM realizes a more rapid prosecution of information, needs less resources of bookstores imported by the applications, DOM strengthens the reuse of code, by means of the implementation of heredity and polymorphism, which cannot be used by the parser DOM that is more dependent of the tool utilized [5].

(Key words: applications of Internet, mobile applications, language of marks, XML, parsers SAX, DOM and JDOM oriented to objects).

Dedicatoria

A:

Dios: Siempre esta conmigo.

Mis padres: Luz de mi vida, y mi inspiración.

Mi marido Salvador, por su paciencia y amor.

Mis hermanos por su amistad y ejemplo de vida, a ¡Mis sobrinos mi alegría!

Agradecimientos

A:

M.C. Ma. Teresa García Ramírez por su amistad, paciencia, ayuda y valiosas observaciones para la elaboración de la tesis.

Mis amigos, que son mis compañeros de estudio y compañeros de trabajo.

Director Ing. Rafael Rodríguez Gallegos y Autoridades del Instituto Tecnológico de San Juan del Río, por su apoyo para la realización de mi formación profesional.

ÍNDICE

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

1. INTRODUCIÓN.

1.1. Descripción del Problema.

Desde la aparición de la WWW (World Wide Web), se han desarrollado diversas tecnologías para la implementación y mejora de las aplicaciones Web, buscando agilizar la funcionalidad y resolver los problemas que plantean las diferentes tecnologías usadas. Inicialmente, las aplicaciones estaban estructuradas usando documentos HTML (Lenguaje de Marcado de Hipertexto, siglas en inglés: Hypertext Marckup Language) con una estructura de hipertexto, que permitía recorrer los diferentes documentos que componían la aplicación. La estructura basada en documentos HTML se ha mejorado añadiendo códigos de lenguajes de programación mediante diferentes tecnologías, para aumentar la funcionalidad de la aplicación [13].

Como parte de las tecnologías que dan soporte a las necesidades reales y actuales de los sistemas de aplicación Web, se crea el lenguaje XML el cual permite resolver ciertas limitaciones de HTML especialmente en el intercambio de documentos entre las empresas y organizaciones, por lo cual se requieren parsers para tener un medio de desarrollo adecuado para implementar el uso de XML.

Los nuevos entornos de programación enfatizan en la construcción de programas, aplicaciones o sistemas de información complejos, que están implementados y se ejecutan en servidores Web, los cuales permiten interactuar con los diversos usuarios de la red [7]. Un lenguaje adecuado para la implementación de estas aplicaciones es Java, ya que es un lenguaje de programación independiente de la plataforma, que permite el desarrollo tanto de aplicaciones locales como aplicaciones en Internet. Además ha tenido un crecimiento acelerado en el desarrollo de diferentes tipos de aplicaciones.

La compañía Sun ha creado otras tecnologías, basadas en Java, que permiten el desarrollo de aplicaciones Web dinámicas, donde se incluye código HTML, XML y Java entre otros. Para el procesamiento de documentos XML se requiere de parsers [15] que son módulos de software que permiten leer documentos XML y proporcionar programas de aplicación con

acceso a su contenido y estructura. Muchos de los parsers para XML están escritos en lenguaje Java. El estudio del parser JDOM para el procesamiento de XML permitirá conocer la estructura y funcionamiento de este parser y como interacciona con XML, determinando las ventajas de su uso, documentando los resultados en comparación de los parser SAX y DOM mediante la implementación de una aplicación para procesar archivos XML.

1.2. Objetivos de la investigación.

- Recabar información de las características de los componentes de XML.
- Analizar las características de SAX y DOM para el procesamiento de XML.
- Analizar las características de JDOM de JAVA para el procesamiento de XML.
- Documentar la técnica de búsqueda de JDOM.
- Realizar pruebas comparativas de los algoritmos de aplicación usando SAX, DOM, así como JDOM de JAVA.
- Identificar las ventajas y eficiencia del uso de JDOM, en casos específicos.

1.3. Justificación.

Debido a los nuevos entornos de programación, se requieren construir aplicaciones y sistemas de información complejos ejecutados en servidores, los cuales permitan interactuar con los diversos usuarios de la red. Java es un lenguaje de programación independiente de la plataforma, permite desarrollar aplicaciones locales y de Internet. La compañía Sun ha creado otras tecnologías, basadas en Java, que permiten el desarrollo de aplicaciones Web, en las cuales se incluye código HTML, XML, JSP y Java.

Por lo anterior, mediante el análisis y uso del parser JDOM para procesar la información contenida en documentos XML, se realizará la comparación de uso de recursos necesarios para los parser SAX y DOM en comparación de JDOM, así como la velocidad de ejecución

de cada uno de ellos, lo cual permitirá comprobar que JDOM es más eficiente para el procesamiento de XML.

1.4. Método de investigación.

Documental Cualitativa.

Se lleva a cabo una investigación documental, para realizar el análisis de los diferentes componentes de XML.

Cualitativo.

Se realizan prácticas de SAX, DOM y JDOM de Java para el procesamiento de XML, con el objetivo de comprobar su eficiencia para el desarrollo de sus aplicaciones.

Documental.

Se documentan los resultados y conclusiones de las tecnologías y métodos para el procesamiento de XML a través de los parser SAX, DOM y JDOM de JAVA, lo cual se llevará a cabo a través de las siguientes actividades.

- Análisis del modelo de aplicaciones XML.
- Análisis los componentes de XML.
- Análisis las características de los modelos SAX, DOM y JDOM de JAVA para el procesamiento de XML.
- Comprobación del modelo de JDOM como herramienta eficiente para el procesamiento de XML.

1.5. Estructura de la tesis.

Esta tesis está estructurada en dos partes. La parte uno, comprende del capítulo uno al tres, ofrece una introducción a XML y Java, así como un análisis de cada una de las tecnologías

para procesar documentos XML. El capítulo uno, la introducción a este trabajo, el capítulo dos, “antecedentes” presenta las características de XML y Java. El capítulo 3, “procesamiento”, hace una descripción de los parsers utilizados en esta tesis para procesamiento de documentos XML.

La parte dos, comprende los capítulos cuatro a siete, en el capítulo 4 se analizan las herramientas utilizadas para el desarrollo de este trabajo, el capítulo 5 describe la aplicación desarrollada para probar cada uno de los parser, los resultados obtenidos en la aplicación desarrollada se muestran en el capítulo 6 y finalmente en el capítulo 7 se muestran las conclusiones del presente trabajo.

En los apéndices, se incluye la configuración del sistema para procesar documentos XML y los códigos completos de la aplicación. En el apéndice A, se realiza una guía para la configuración del sistema para el procesamiento de XML. En el Apéndice B se muestran los códigos completos de la aplicación.

2. ANTECEDENTES

Actualmente el manejo de la información se ha diversificado debido a la rápida evolución de Internet y de las comunicaciones Web, lo que ha hecho necesario el surgimiento de nuevas tecnologías o metodologías para el acceso de información.

2.1 Lenguajes de marcación.

La creación de lenguajes de marcación va más allá del aspecto de la información, estos lenguajes permiten asignar un significado a los datos. El uso de lenguajes de marcación, lenguajes de programación (C, Perl, Java entre otros) y las nuevas tecnologías aumentan la funcionalidad de la WWW al poder crear aplicaciones interactivas en el entorno Web, los lenguajes y las tecnologías creadas en su entorno están evolucionando continuamente [10].

En el ámbito de la utilización de estos lenguajes han aparecido nuevas aplicaciones informáticas que operan en la red de Internet como el comercio electrónico (e-Commerce) o el negocio electrónico (e-Business) entre los lenguajes de marcación más populares podemos mencionar los siguientes:

GML (General Markup Language) fue uno de los primeros lenguajes de marcación diseñado para escribir estructuras de datos descriptivas esto es, estructuras de datos describiendo otras estructuras de datos [7].

SGML (Standard Generalized Markup Language) surge de GML como un estándar internacional para el intercambio y almacenamiento de información a partir de 1986. Actualmente SGML aún es utilizado en varios proyectos por tratarse de un lenguaje de marcación muy eficaz, pero difícil de desarrollar. Este lenguaje de marcación se usa en proyectos gubernamentales, industrias manufactureras y publicistas, inclusive por el navegador ("Explorer" o "Netscape") para asegurar que la información sea reutilizable independientemente del sistema operativo o aplicación [11].

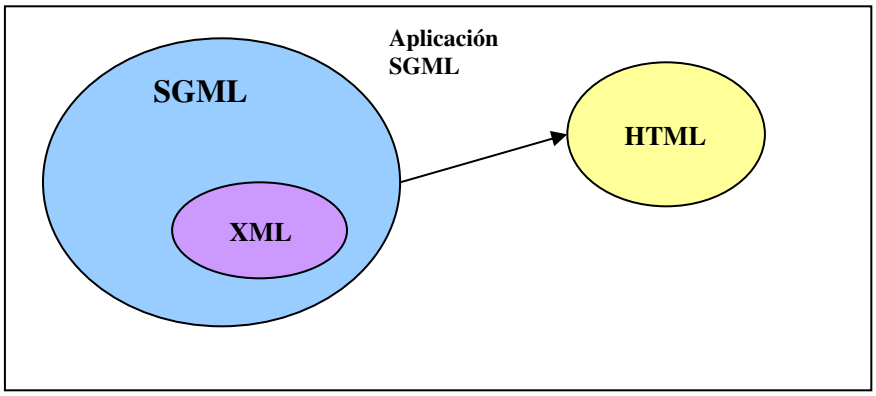

Figura 2.1. Lenguajes de Marcas

HTML (Hypertext Marckup Language) es parte de SGML. Los navegadores (como Netscape y Explorer) contienen al menos tres DTD (Definición de Tipo de Datos) para HTML, estos DTD indican como debe desplegarse la información en el navegador, forman parte primordial de SGML y son estos DTD los que realmente van conformando el estándar HTML. Diversos navegadores están equipados con DTD propietarios, lo cual hace que ciertos elementos de HTML sean incompatibles con otros navegadores.

2.2. XML.

El intercambio de Información siempre ha sido un grave problema cuando se utilizan lenguajes y sistemas operativos diferentes, lo cual se presenta desde los niveles de procesadores de palabras hasta la utilización de bases de datos que utilizan diferentes métodos para acceso a la información [11].

Previa aparición de Internet fueron creados mecanismos para lograr el intercambio fluido de información entre diferentes sistemas, el primer método fue GML, posteriormente SGML y actualmente XML un estándar internacional para la definición de lenguajes de marcado que permite realizar una gestión de la información eficiente y facilita el manejo de la misma.

XML fue diseñado para trabajar en la Web y algunos autores como Abraham Gutiérrez considera que posiblemente sea la innovación más importante desde la aparición de Java [7]. Entre las ventajas que ofrece son la independencia de los datos respecto a las

aplicaciones, elementos para describir la estructura de un documento, herramientas para organizar la información, métodos para visualizar la información, entre otras.

El éxito de XML se debe a que facilita el intercambio de información, ya que es posible utilizar este lenguaje de marcación para distribuir información que sea utilizada por bases de datos, aplicaciones de servidor, aparatos inalámbricos, impresoras, entre otros. La principal ventaja que presenta este lenguaje es su independencia del sistema operativo y de la aplicación, esto es, se puede tener un documento escrito en XML y este puede manipularse en los diferentes sistemas operativos: Sun Solaris, Windows, o en un ambiente Java, VBScript, entre otros. XML no es una solución para todo sistema de información, por lo cual es conveniente ubicar su utilización [15].

En la figura 2.2 se muestra un documento XML que a través de varias herramientas desarrolladas para el lenguaje, se puede utilizar para diferentes aplicaciones (aparatos inalámbricos, bases de datos, servidores Web entre otros), las flechas bidireccionales indican que el proceso se puede realizar en cualquier sentido y en cualquier medio que soporte XML. Debido a esta facilidad de intercambio de información, XML se ha utilizado en desarrollos B2B (Business to Business) [9]. En la figura 2.2 se presenta la independencia del medio de una aplicación XML.

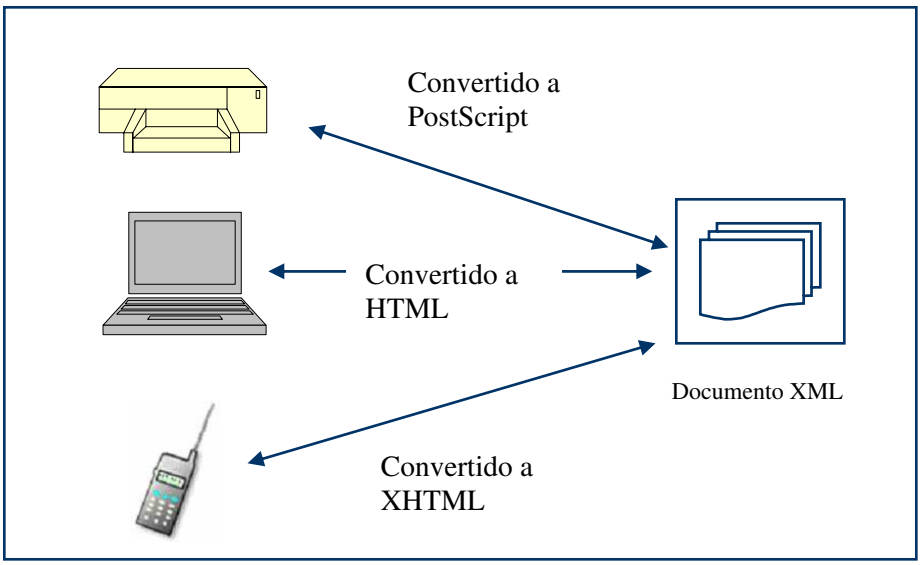

Figura 2.2. XML es independiente del medio

2.2.1. Componentes de XML

La comunicación de la información es un tema amplio, por lo cual existen varias especificaciones de comunicación interrelacionadas (algunas se encuentran en una fase de desarrollo inicial) que funcionan de manera conjunta para formar la familia de la tecnología XML, de tal manera que cada especificación cubre un aspecto diferente para la comunicación de la información, en la tabla 2.1 se listan los componentes de XML más importantes:

Tabla 2.1. Componentes de XML

Componente	Descripción
XML	Es un Lenguaje extensible de marcado publicado por el consorcio W3C (Consorcio de World Wide Web) para resolver las limitaciones de HTML.
DTD (Definición de Tipos de Documento, sus siglas en inglés Document Type Definition)	Las DTD brindan diversos métodos para crear plantillas para diferentes tipos de documento. Se puede lograr que los documentos producidos por otros desarrolladores, utilicen estas plantillas y obtener así documentos compatibles con una definición particular.
Namespaces (Espacio de Nombre)	Es un medio para distinguir entre un vocabulario XML y otro, lo que permite crear documentos más consistentes mediante el uso de múltiples vocabularios dentro de un mismo documento.
XPath (Lenguaje de ruta XML, sus siglas en inglés XML Path Language)	Describe un lenguaje de consulta para el direccionamiento de partes de un documento XML. Esto permite que las aplicaciones puedan requerir una parte específica de un documento XML, en lugar de tener que tratar siempre con todo el documento completo.
CSS (Hojas de Estilo en Cascada sus siglas en inglés Cascading Style Sheets)	Para visualizar los documentos XML, en los casos más simples se puede usar CSS, para definir la presentación de los documentos.
XSL (Lenguaje de Hojas de Estilo Extensible sus siglas en inglés Extensible Style Sheet Language)	Es un lenguaje extensible de hojas de estilo permite describir cómo la información contenida en un documento XML debe ser transformada o formateada para su presentación en un medio específico.
XLink (enlaces entre documentos XML) y XPointer (Lenguaje de punteros XML)	Son lenguajes que se utilizan para vincular documentos XML entre sí, de una manera similar a los hipervínculos de HTML.
SAX	Es otro medio alternativo para tener una interfaz con documentos XML desde el código de un programa.
XSLT (Lenguaje de hojas de estilo para XML)	Lenguaje para transformar documentos de un formato XML a otro formato (HTML, otro vocabulario XML, texto plano, PDF, entre otros.)
XHTML (hypertext markup language)	Reformulación del HTML como formato de datos XML.
SOAP (Simple Object Access Protocol).	Lenguaje que especifica la forma de enviar contenido XML a través de Internet.
DOM	Es un medio para que las aplicaciones más tradicionales puedan tener un interfaz con los documentos XML.

2.2.2. Gestión Eficiente con XML.

Uno de los principales requisitos para que dos sistemas intercambien datos es que se pongan de acuerdo en la representación y en el formato de los documentos el concepto "Gestión de la información abierta", significa administrar la información de manera que esté disponible para el procesamiento por cualquier programa y no únicamente para el que la creó se fundamenta en el principio de independencia de los datos: los datos se deben almacenar en formatos en representaciones normalizadas, que distingan los datos reales de los códigos de los programas de procesamiento.

2.2.3. Beneficios de Utilizar XML.

- Por sus características, XML permite el manejo de datos estructurados y el intercambio de datos, pudiendo extraer únicamente los datos que se requieran así como intercambiar información con base de datos u otra información a través de Internet; a continuación se listan las ventajas de XML para la comunicación de datos:

- **XML** complementa **HTML**: Se pueden utilizar datos de XML en una página de HTML.

- **Buscadores**: Aumenta la rapidez de las búsquedas debido a la información contextual que se encuentra en los documentos XML.

- **Actualizaciones**: No hay necesidad de actualizar el sitio página por página, ya que DOM y JDOM permiten el acceso y actualización de elementos individuales.

- **Visualización de datos elegidos por el usuario**: Distintos datos se pueden acceder por diferentes usuarios, esto es presentar la misma información de diversas maneras.

- **Independencia de los datos** respecto de las aplicaciones.

- **Información sobre la información**, un contexto, muy útil en el proceso informático [9].

- **Visualización de la información**. Métodos para variar la visualización de la información y capacidad para mostrarla en función del contenido y de la semántica del texto según la aplicación.

- **Capacidad** para enviar información a las aplicaciones.

- **Enlaces** para los datos relacionados.

2.2.4. Aplicaciones de XML.

XML puede utilizarse en cualquier aplicación en la que se pretenda guardar, recuperar o actualizar información estructurada y validar su estructura y contenido, lo que incluye prácticamente cualquier información informática [13] algunas aplicaciones son las siguientes:

- **XML en publicaciones**: permite separar contenido y presentación pudiendo utilizar XML para la representación de los datos de la base de datos, la sustitución de HTML y para formato de documentos científicos y de oficina, así mismo la representación de los datos se puede publicar de diferentes formas.

- **XML en intercambio de información entre aplicaciones**: en aplicaciones distribuidas, y en general, en el intercambio de información entre aplicaciones se pude considerar a XML como protocolo nativo de comunicación. Sobre una capa XML, se monta una capa de procesadores XML que ofrecen a las aplicaciones servicios básicos para la gestión de los datos almacenados en formato XML. Los datos pueden validarse en ambos extremos de la comunicación, en la figura 2.3 se muestra como se puede realizar el intercambio de datos usando XML.

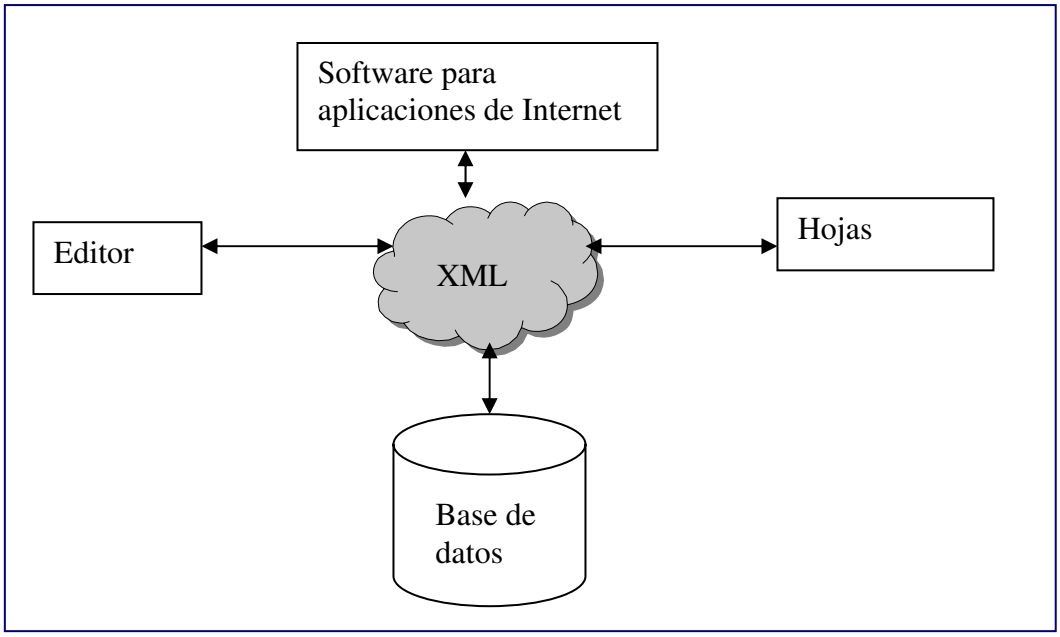

Figura 2.3. Intercambio de datos usando XML

XML y persistencia: los datos de configuración de aplicaciones pueden guardarse con formato XML. Esto facilita el intercambio de información entre aplicaciones. La serialización de objetos permite guardar el estado de un objeto para facilitar el intercambio de objetos por valor entre aplicaciones distribuidas en distintas plataformas, por ejemplo entre objetos CORBA (Common Object Request Broker Architecture) y objetos RMI (Java Remote Method Invocation).

No es fácil modelar las relaciones estructuradas de una base de datos mediante un diagrama entidad/relación que luego se traduzca en un modelo físico para un cierto RDBMS (Relational Data Base Management System), se puede utilizar XML para modelar datos fuertemente estructurados como por ejemplo productos compuestos, incluyendo las relaciones de integridad referencial.

2.2.5. XML en Sistemas de Información.

Actualmente existen varias metodologías de desarrollo de software, los métodos complejos, según la aplicación puede ajustarse al proyecto y a las condiciones de desarrollo. Desde un punto de vista arquitectónico tales metodologías ayudan a disminuir los riesgos más

importantes y sobre todo auxilian a representar la funcionalidad básica del sistema a construir.

Para el desarrollo de aplicaciones complejas se puede aplicar la metodología orientada a objetos usando el lenguaje Java, los parser que procesan datos en formato XML se pueden desarrollar en Java, codificando datos de acuerdo a un esquema semántico. Para codificar información en XML se genera un archivo de texto estructurado donde se combina la información fuente con metadatos (datos altamente estructurados que describen información) lo cual se ilustra en la figura 2.4.

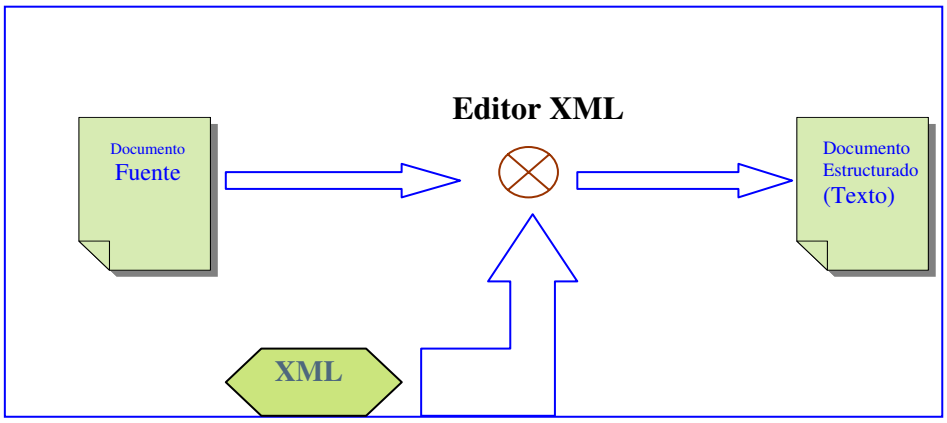

Figura 2.4. Editor XML

Una aplicación XML puede analizar y extraer información de un documento XML, obtener información de otras fuentes (como base de datos, documentos, software entre otros) y combinarla con el documento XML para un proceso posterior.

Modelos de Procesamiento XML

Una aplicación basada en XML puede recuperar información y generar documentos XML entre otros formatos resultantes, las actividades que se realizan para el procesamiento de XML se listan a continuación:

- Analizar y validar el documento fuente.
- Reconocer/buscar datos basándose en el etiquetado del documento fuente.
- Extraer la información importante una vez que se ha localizado.

Los modelos de procesamiento que se pueden aplicar para procesar documentos de XML son SAX, DOM y JDOM. Si se usa SAX para procesar un documento XML, se tienen que codificar métodos para manejar eventos generados por el analizador, según encuentre diferentes identificadores del lenguaje de marcas.

Un analizador SAX genera un flujo temporal de eventos, por lo cual se debe hacer en un sólo ciclo el proceso de entrada de datos en formato XML (nombrado, análisis, reconocimiento, extracción y mapeo) cada evento se procesa inmediatamente y la información requerida se envía con el evento [15].

Para que una aplicación procese datos XML cuando se van analizando, se deben declarar manejadores con el analizador SAX, un manejador es un conjunto de llamadas de retorno que SAX define para agregar el código de la aplicación en eventos importantes durante el análisis de un documento. Estos eventos se producirán cuando el documento se analice, esta es una de las razones por las que SAX es una interfaz tan poderosa ya que permite manejar un documento secuencialmente sin tener que cargar primero el documento entero en memoria.

Una característica importante es que al escribir código SAX no se sigue un estilo orientado a objetos, sino que se realiza con un estilo jerárquico, como los eventos de SAX ocurren secuencialmente no es posible manejar los hijos de los elementos directamente, sino que el almacenamiento debe asignarse para almacenar el nombre del elemento que está siendo procesado. El análisis SAX incluye por lo general leer un documento y almacenar los datos que se pasan a la llamada de vuelta con el nombre del último elemento procesado (mediante la llamada de startElement()) al final del procesamiento del elemento (endElement()) se carga la información y se puede utilizar.

Mientras SAX y este acercamiento secuencial son algo más rápidos que DOM el código resultante no es tan claro ni tan fácil.

En el uso del parser DOM se necesita el código que permita pasar a través de una estructura de datos tipo árbol creada por el analizador desde el documento fuente. Cuando se usa DOM, el procesamiento de la entrada XML se hace al menos en dos ciclos: primero, el analizador DOM crea una estructura de datos tipo árbol que modela el documento fuente XML (árbol DOM), luego se recorre el árbol DOM buscando la información para extraerla y procesarla posteriormente este último ciclo se puede repetir tantas veces como sea necesario mientras el árbol DOM exista en memoria.

DOM suministra una visualización completa de un documento XML de hecho el documento debe ser leído completamente en la memoria antes de poder acceder a el aunque esto no es un gran problema con archivos pequeños puede ser una desventaja con documentos grandes de XML en la Tabla 2.2 se listan las características de los parser SAX y DOM.

Tabla 2.2. Características de SAX y DOM

SAX	DOM
Modelo basado en eventos	Estructura de datos tipo árbol
Acceso serie (flujo de eventos)	Acceso aleatorio (estructura de datos en memoria)
Bajo uso de memoria (sólo se generan eventos)	Alto uso de memoria (todo el documento se carga en memoria)
Para procesar partes del documento (capturar eventos importantes)	Para editar el documento (procesar la estructura de datos en memoria)
Para procesar el documento sólo una vez (flujo de eventos temporal).	Para procesar el documento múltiples veces (documento cargado en memoria).

2.3 Java.

Java es un lenguaje de programación independiente de la plataforma, que permite el desarrollo tanto de aplicaciones locales como aplicaciones en Internet. Además ha tenido un crecimiento acelerado en el desarrollo de aplicaciones de diferentes tipos. La compañía Sun ha creado otras tecnologías, basadas en Java, que permiten el desarrollo de aplicaciones Web dinámicas, donde se incluye código HTML, XML y Java entre otros.

2.3.1. Programación en Red.

Una de las mayores fortalezas de Java es su buen funcionamiento en red [3]. Los diseñadores de bibliotecas de red de Java han logrado que trabajar en red sea bastante similar a la escritura y lectura de un archivo, excepto que el archivo existe en una computadora remota y esta pueda decidir exactamente que desea hacer con la información que se le envía o solicita. Siempre que es posible se han abstraído los detalles de la red, dejándose para la JVM (Java Virtual Machine) y la instalación de Java que haya en la computadora local.

La conexión de red se realiza por medio de la comunicación de objetos, el multihilo inherente a Java es útil ya que permite a la red la manipulación simultánea de múltiples conexiones. A continuación se muestra un comparativo de las características que tiene Java con otros lenguajes orientados a objetos.

Tabla 2.3. Comparativo de Java contra otros Lenguajes Orientados a Objetos [3]

Característica	***Java***	***SmallTalk***	***Perl***	***C++***
Simple	Sí	Sí	Regular	No
OO	Sí	Sí	Sí	Regular
Robusto	Sí	Sí	Sí	Sí
Seguro	Sí	Regular	Sí	Sí
Interpretado	Sí	Sí	Sí	No
Dinámico	Sí	Sí	Sí	No
Portable	Sí	Regular	No	Regular
Neutral	Sí	Regular	Sí	No
Multihilo	Sí	No	Sí	No
Rec. Basura	Sí	Sí	No	No
Excepciones	Sí	Sí	Sí	Sí
Rendimiento	Alto	Medio	Medio	Muy Alto

2.3.5. Computación cliente/servidor.

En el concepto básico de la computación cliente/servidor, se presentan problemas por que se tiene un único servidor que trata de dar servicio a múltiples clientes simultáneamente. Generalmente, está involucrado algún sistema gestor de base de datos de manera que el diseñador "reparte" la capa de datos entre distintas tablas para lograr un uso óptimo de los datos. Además estos sistemas suelen admitir que un cliente inserte nueva información en el servidor.

Esto significa que es necesario asegurarse de que el nuevo dato de un cliente no sobreponga los nuevos datos de otro cliente, o que no se pierda este dato en el proceso de añadir hacia la base de datos (proceso de transacción) al cambiar el software el cliente debe ser construido, depurado e instalado en las máquinas cliente lo cual se vuelve más complicado.

Es especialmente problemático dar soporte a varios tipos de computadoras y sistemas operativos. Existe un aspecto de rendimiento importante: es posible tener cientos de clientes haciendo peticiones simultáneas a un mismo servidor, de forma que es importante un retraso mínimo. Los programadores deben empeñarse a fondo para disminuir las cargas de las tareas en proceso, generalmente repartiéndolas con las máquinas cliente, pero en ocasiones, se dirige la carga hacia otras máquinas ubicadas junto con el servidor, denominadas intermediarios "middleware" (que también se utiliza para mejorar el mantenimiento del sistema global).

El procesamiento cliente/servidor consume prácticamente la mitad de todas las actividades de programación. Por ejemplo en una aplicación bancaria es responsable de recibir las órdenes y transacciones de tarjetas de crédito hasta la distribución de cualquier tipo de datos; en el pasado, se han intentado y desarrollado soluciones individuales para problemas específicos, inventando una solución nueva cada vez [3]. Estas soluciones eran difíciles de crear y utilizar; y el usuario debía aprenderse nuevas interfaces para cada una de ellas, por lo tanto el problema cliente/servidor debe resolverse con un enfoque global.

2.3.2. Java y los sistemas cliente/servidor.

Java es muy útil para la solución de problemas de programación tradicionales autónomos, también es importante resolver problemas de programación para aplicaciones Web. Para entender el beneficio del uso de estas aplicaciones, es necesario analizar los sistemas cliente/servidor.

La idea principal de un sistema cliente/servidor es que se dispone de un depósito central de información, generalmente en una base de datos que se desea distribuir de acuerdo a una demanda en un determinado conjunto de computadoras o personas. Para entender el concepto de cliente/servidor se considera que el depósito de información está ubicado centralmente, de tal forma que puede ser modificado de manera que los cambios se distribuyan a los consumidores de la información.

A la computadora en la que se ubica el depósito de información y el software que la distribuye se le denomina el servidor [3]. El software que reside en la computadora remota se comunica con el servidor; toma la información, la procesa y después la muestra en la computadora denominada cliente.

A continuación se listan las ventajas de Java como lenguaje de programación de propósito general para las aplicaciones del lado del servidor:

a) Independencia de plataformas

b) Alta productividad

c) Soporte incorporado a Internet

d) Soporte de caracteres internacionales

Los programas escritos en Java corren en plataformas distintas y gracias a la JVM se puede obtener información en diferentes formatos de salida tales como imágenes, sonido, información vía fax, datos que se pueden desplegar en dispositivos móviles entre otros.

2.3.3. Alta Productividad.

Las características que contribuyen a aumentar la productividad del lenguaje Java son:

- Clara separación entre la interfaz y la implementación.
- Ausencia de construcciones susceptibles de generar errores, como la herencia múltiple y la sobrecarga del operador.
- Soporte de nivel de lenguaje para hilos y monitores (objetos de datos para sincronización).

2.3.7. Programación en el lado del cliente.

El diseño original servidor-navegador de la Web proporcionaba contenidos interactivos, pero la capacidad de interacción se obtenía completamente del servidor produciendo páginas estáticas para el navegador del cliente, que solo las interpretaba y visualizaba. El HTML básico contiene mecanismos simples para la recopilación de datos: cajas de entrada de textos, cajas de prueba, cajas de radio y listas desplegables, además de un botón que solo podía programarse para borrar los datos del formulario o "enviar" los datos del formulario de vuelta al servidor.

Este envío de datos se lleva a cabo a través del CGI (Common Gateway Interface), proporcionado por todos los servidores Web. El texto del envío transmite a CGI lo que debe hacer con él [3].

Conectables (plug-ins).

Uno de los mayores avances en la programación en la parte del cliente es el desarrollo de programas para reproducir determinados formatos de gráficos, reproducir datos multimedia, entre otros (conectables), éstos son modos en los que un programador puede añadir nueva funcionalidad al navegador descargando fragmentos de código que se conecta en el punto adecuado del navegador. A través de los conectables, se añade comportamiento rápido y potente al navegador.

El valor del conectable para la programación en el lado cliente es tal que permite a un programador experto desarrollar un nuevo lenguaje y añadirlo a un navegador sin permiso de la parte que desarrolló el propio navegador por consiguiente los navegadores proporcionan una posibilidad que permite la creación de nuevos lenguajes de programación en el lado cliente.

2.3.8. Programación en el lado del servidor.

Si un usuario quiere registrar su nombre en una base de datos al incorporarse a un grupo o presentar una orden, implica cambios a la base de datos las peticiones que requieren de modificación a la base de datos deben procesarse vía algún código en el lado servidor generalmente, esta programación se ha desempeñado mediante el lenguaje de programación Perl y guiones CGI, pero han ido apareciendo sistemas más sofisticados.

Entre éstos se encuentran los servidores Web basados en Java que permiten llevar a cabo toda la programación del lado servidor en Java escribiendo lo que se denominan *servlets* (Interface local). Estos y sus descendientes los JSP (Java Server Pages), son las dos razones principales por las que las compañías que desarrollan sitios Web están desarrollando sus aplicaciones usando el lenguaje de programación Java, especialmente por que eliminan los problemas de tener que tratar con navegadores de distintas características.

2.3.9. Beneficios del Lenguaje Java.

Universalidad. Java es el lenguaje idóneo para desarrollar aplicaciones para Internet, de hecho la mayor parte de los navegadores (Netscape Navigator, Internet Explorer, HotJava) integran máquinas virtuales, y por tanto intérpretes de Java lo cual hace posible acceder automáticamente a los applets en las páginas Web la sencillez de Java hace que esta integración no reduzca las ventajas de los navegadores, permitiendo además la ejecución rápida y simultánea de muchos applets.

También Java es universal debido a su transportabilidad, o independencia de plataforma, ya que para ejecutar un programa se compila una sola vez a partir de entonces se puede hacer correr en cualquier máquina que tenga implementado un intérprete de Java.

Las bibliotecas estándar de funciones y métodos de Java (definidas en su API, *Application Programming Interface*) facilitan la programación de multitud de acciones complejas (desarrollo de interfaces gráficas, multimedia, multitarea, interacción con bases de datos entre otros) ningún otro lenguaje (ni compilado ni interpretado) dispone como Java de una cantidad tan grande de funciones accesibles en cualquier plataforma sin necesidad de cambiar el código fuente.

Sencillez. Java es un lenguaje de gran facilidad de aprendizaje, pues en su concepción se eliminaron todos aquellos elementos que no se consideraron absolutamente necesarios. Por ejemplo, en comparación con otros lenguajes como C ó C++, es notable la ausencia de punteros, de tal manera que no se hace referencia de forma explícita a una posición de memoria; ahorrando tiempo a los programadores, ya que el uso de punteros es una de las principales fuentes de errores en la ejecución de un programa. Además el código escrito en Java es más legible que el escrito en C ó C++.

Java tiene un mecanismo de "recolección de basura" el cual usando la capacidad multitarea de Java hace que, durante la ejecución de un programa, los objetos que ya no se utilizan se eliminen automáticamente de la memoria dicho mecanismo facilita el diseño de un programa y optimiza los recursos de la máquina para la ejecución del mismo (con los lenguajes tradicionales, la eliminación de objetos y la optimización de recursos se debe hacer al diseñar el programa).

Orientación a objetos. C++ al igual que Java es un lenguaje orientado a objetos una diferencia importante es que Java lo es desde su origen, mientras que C++ se diseñó como un lenguaje compatible con C (que no es orientado a objetos) de tal manera que un programa escrito en C++ puede no usar las características del modelo orientado a objetos

de C++ y compilarse sin problemas en un compilador de C++, en comparación un programador usa la orientación a objetos cuando escribe un programa en Java de manera natural.

En los lenguajes orientados a objetos, un objeto es un elemento de programación, reutilizable, y se puede definir como la representación formada por un conjunto de variables (los datos) y un conjunto de métodos (o instrucciones para manejar los datos) por ejemplo en una aplicación de Bibliotecas un objeto puede ser un código de barras, que contiene datos (por ejemplo el número mismo del código de barras) e instrucciones para manejarlos (por ejemplo el método para calcular el dígito de control) los objetos además poseen la capacidad de enviarse mensajes entre sí durante la ejecución de un programa.

Un objeto puede contener métodos y variables que no son accesibles desde fuera del objeto, lo que evita posibilidades de error al diseñar un programa. Esta característica se llama "ocultamiento de la información".

Es posible reutilizar código, en el ejemplo anterior, un programador puede reutilizar el objeto "código de barras" para escribir software para diversos tipos de transacciones, como préstamo, devolución, reserva, entre otros, sin necesidad de reescribir cada vez las instrucciones para calcular el dígito de control dado que el programa se encargará de hacerlo cada vez que se use el objeto.

Seguridad. Se considera que un lenguaje es más seguro cuando es menor la posibilidad de que errores en la programación, o diseños malintencionados de programas (virus), causen daños en el sistema. La seguridad de Java se establece en los siguientes niveles:

Nivel de seguridad dado por las características del lenguaje, tales como la ausencia de punteros (que evita cualquier error de asignación de memoria) o el "ocultamiento de la información" propio de la programación orientada a objetos.

Nivel de seguridad dado por el diseño de la Máquina Virtual de Java que posee un verificador de los códigos intermedios de java (byte codes), de tal manera que antes de

ejecutarlos analiza su formato para comprobar que no existan punteros en ellos, que se accede a los recursos del sistema a través de objetos de Java.

Adaptación a redes (y a Internet). Java entró en el mercado para potenciar la interactividad en Internet y en los applets que son programas en Java que se cargan junto con una página Web desde un servidor ejecutados (por la Máquina Virtual del navegador del cliente) como una parte de la página Web.

Java no solo es un lenguaje de programación potente seguro y de plataforma cruzada (multiplataforma) ya que se está extendiendo continuamente para proporcionar aspectos de lenguaje y bibliotecas para desarrollar sistemas complicados java brinda el soporte para ejecutar varios procesos al mismo tiempo, el acceso a base de datos, la programación en red y la computación distribuida.

2.4 XML y JAVA.

XML es un estándar industrial para representar datos, de manera independiente del sistema al igual que HTML, XML especifica los datos en etiquetas, pero hay importantes diferencias entre los dos lenguajes de marcas, las etiquetas XML tienen relación con el significado del texto que contienen, mientras que las etiquetas HTML especifican cómo mostrar el texto contenido.

Otra diferencia importante entre XML y HTML es que las etiquetas XML son extensibles, permitiendo escribir etiquetas propias de XML para describir el contenido. Con HTML, se tiene la limitante de usar sólo aquellas etiquetas predefinidas en la especificación HTML como un documento XML no incluye instrucciones de formateo, puede mostrarse de varias formas mantener los datos separados de las instrucciones de formateo significa que los mismos datos pueden publicarse en diferentes medios.

Los parsers de Java para XML permiten escribir aplicaciones Web completamente en el lenguaje Java la plataforma Java hace posible el manejo de datos portables los parsers de Java para XML hacen fácil el uso del XML si se consideran estas ventajas de manera

conjunta se tendrá la una buena combinación que es portabilidad de datos, portabilidad de código y facilidad de uso. El procesador de XML para Java es una biblioteca de clases Java y por tanto requiere el uso de algunas tareas básicas de programación para Java puesto que XML para Java está escrito en Java, se puede ejecutar en cualquier plataforma de sistema operativo o en cualquier hardware que soporte Java.

3. PROCESAMIENTO

3.1. Parsers de XML.

Los parsers de XML son tecnologías disponibles para procesar documentos XML, se ejecutan en diferentes niveles de abstracción y proporcionan diferentes niveles de aplicación para el programador entre los parsers más comunes están SAX y DOM los cuales no fueron diseñados para un ambiente de desarrollo en Java, por lo que se requirió proporcionar una metodología para procesar XML más acorde con las aplicaciones Java para tal objetivo surgió JDOM.

A continuación se describe el modelo y los elementos de programación de estos parsers para el procesamiento de documentos XML en diferentes aplicaciones usando el lenguaje Java.

3.1.1. Parser SAX.

El parser SAX usa un modelo basado en eventos y permite el procesamiento de un documento fuente como un flujo de eventos que se ejecutan mientras se analiza el flujo continuo de retro llamadas e invocaciones a métodos. Los eventos están anidados de la misma forma que los elementos en el documento, por lo tanto, no se crea ningún modelo de documento intermedio [9].

El parser SAX para XML proporciona eventos para analizar datos XML, es decir realizar el proceso de leer el documento desglosando los datos en partes utilizables; en cada paso SAX define los eventos que pueden ocurrir por ejemplo SAX define una interfaz *org.xml.sax.ContentHandler* que define métodos como *StarDocument() y endElement().* La implantación de esta interfaz permite un control completo sobre las partes del proceso de análisis de XML.

Hay una interfaz parecida para el manejo de errores y construcciones léxicas. Se define un conjunto de errores y avisos, permitiendo el manejo de varias situaciones que existan

durante el análisis de XML, como un documento inválido o un documento con mal formato, se puede añadir más funcionalidad para hacer a medida el proceso del análisis.

SAX define cuatro interfaces de manejadores principales: *org.xml.sax.ContentHandler, org.xml.sax.ErrorHandler, org.xml.sax.DTDHandler y org.xml.sax.EntityResolver*.

ContentHandler permite manejar eventos estándar relacionados con los datos en un documento XML, es un interfaz que recibe eventos del documento y realiza las notificaciones:

- Inicio o final de un documento.
- Inicio y final de un elemento.
- Datos de carácter.
- Espacios en blanco ignorables en el contenido de elementos.
- Instrucciones de procesamiento.

ErrorHandler recibe notificaciones del analizador cuando se encuentran errores en los datos XML, notifica:

- Un error recuperable.
- Un error fatal/no recuperable.
- Un aviso.

DTDHandler es un interfaz para recibir notificación de eventos básicos relacionados con DTD, como:

- Un evento de declaración de notación.
- Un evento de declaración de entidad sin analizar.

EntityResolver es un interfaz para resolver referencias a entidades externas que se especifiquen en un documento XML.

Cada una de estas interfaces se pueden implementar por medio de clases de aplicaciones que realicen acciones específicas en el proceso de análisis, como se muestra en la Figura 3.1 estas clases se pueden declarar en el analizador con los métodos *setContentHandler()*, *setErrorHandler(), setDTDHandler()* y *setEntityResolver()* el analizador invoca los métodos de llamada de retorno durante el análisis. Una aplicación debe proporcionar al menos un manejador de documento (o contenido) para poder capturar los eventos y procesarlos.

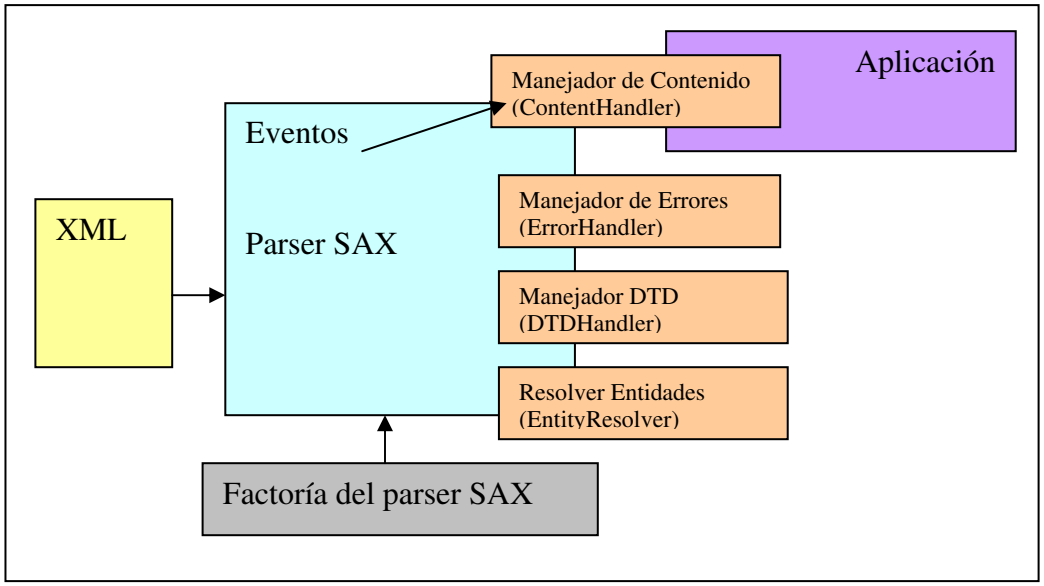

Figura 3.1. Modelo del parser SAX

El desarrollador tiene que implementar un manejador de contenido (*ContentHandler*) cuando se usa el parser SAX o subclasificar un manejador por defecto (*DefaultHandler*) para que se puedan capturar los eventos y los objetos de la aplicación mapeados o directamente manejados por la lógica de la aplicación.

Se usa una factoría del procesador SAX (*SAXParserFactory*) para crear un objeto para el procesamiento de documento XML la implementación personalizada del interfaz manejador base (*HandlerBase*) y ruta informática (path) del documento fuente XML a procesar son pasados como parámetros al analizador durante el análisis, se llama al método *startElement()* por cada etiqueta de inicio del documento fuente. El programa procesa un documento XML basándose en el API SAX.

En la Figura 3.2 se muestra el uso del método *startElement()* que imprime información basada en las etiquetas de inicio, una aplicación real requiere que se capturen eventos como *endElement()* (que notifica un contenido textual) y mantener un contexto para mapear el flujo de eventos de los objetos de la aplicación.

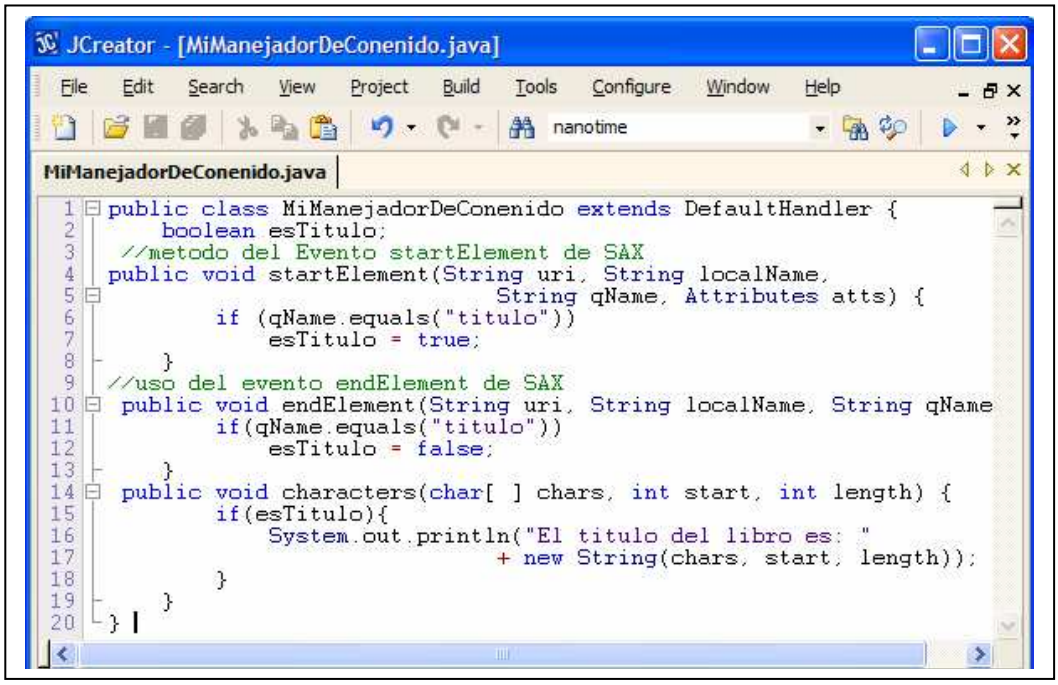

Figura 3.2. Ejemplo de manejadores de eventos de SAX

3.1.2. Parser DOM.

DOM es una interfaz neutral del lenguaje y la plataforma que permite a los programas acceder dinámicamente, actualizar el contenido, la estructura y el estilo del documento XML, esencialmente es una estructura de datos en forma de árbol y un conjunto de métodos para acceder y editar esa estructura. El modelo de documento puede accederse aleatoriamente y puede ser procesado múltiples veces a través del árbol.

DOM define interfaces para cada una de las entidades de un documento XML:

- Interfaz Nodo, en este interfaz un sólo nodo en el árbol del documento define:
  - Métodos para acceder, insertar, eliminar y reemplazar nodos hijo.
  - Métodos para acceder al nodo padre.

- Métodos para acceder al documento.

- Interfaz Documento es un nodo que representa un documento completo XML.

- Interfaz Texto es un nodo que representa el contenido textual de un elemento XML.

En la Figura 3.3 se muestra como la lógica de aplicación puede usar directamente al árbol DOM y la relación de los elementos que interactúan en el modelo del parser DOM:

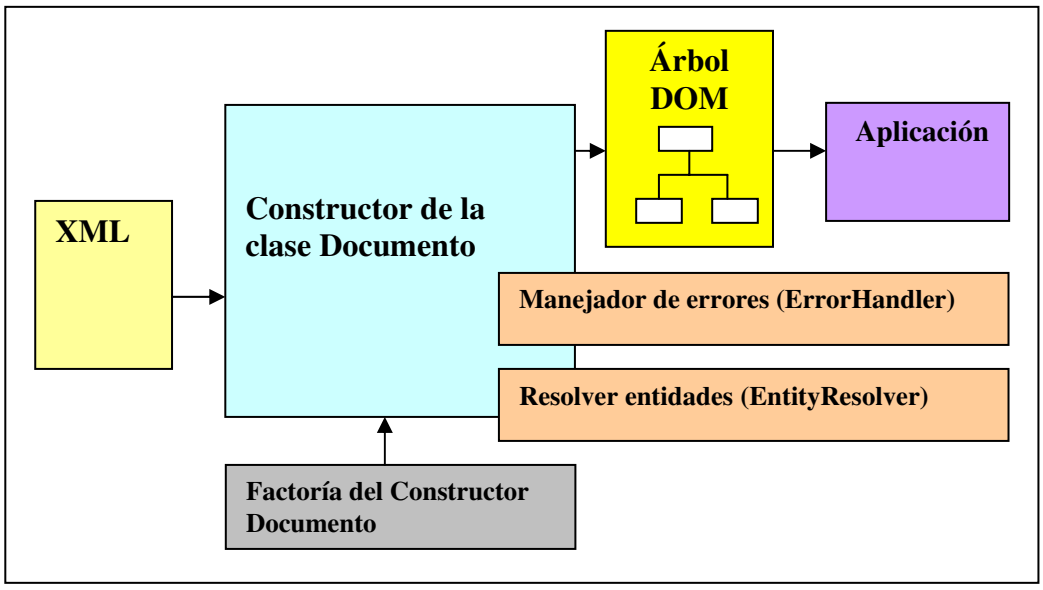

Figura 3.3. Modelo del parser DOM

Cuando se usa el parser DOM la aplicación tiene que acceder o editar una representación en la memoria del documento fuente. El parser DOM analiza y carga en memoria un documento XML generando una estructura de datos de tipo árbol y mediante el recorrido de sus nodos se listan los datos en formato de texto.

Una vez que el documento se ha procesado y se ha creado un objeto de tipo Documento, se pueden olvidar las diferencias entre los diversos procesadores (parsers) y trabajar con las interfaces estándar que proporciona el parser de DOM.

Todos los nodos de un árbol DOM están representados por la interfaz *Nodo*. Los métodos más importantes de esta interfaz son lo que permiten navegar por el árbol de nodos DOM. Como no todos los nodos tienen hijos, es importante comprobar la presencia de hijos con *hasChildren()* antes de llamar a *getFirstChild()* y *getLastChild()*. Lo mismo se aplica para los atributos, mediante los métodos correspondientes. A continuación se muestra un ejemplo de las instrucciones para acceder a los nodos del árbol que crea DOM, para el procesamiento de documentos XML.

En la Figura 3.4 se muestra como se declara una variable de tipo factoría mediante la cual se puede hacer referencia al documento XML usando el parser DOM.

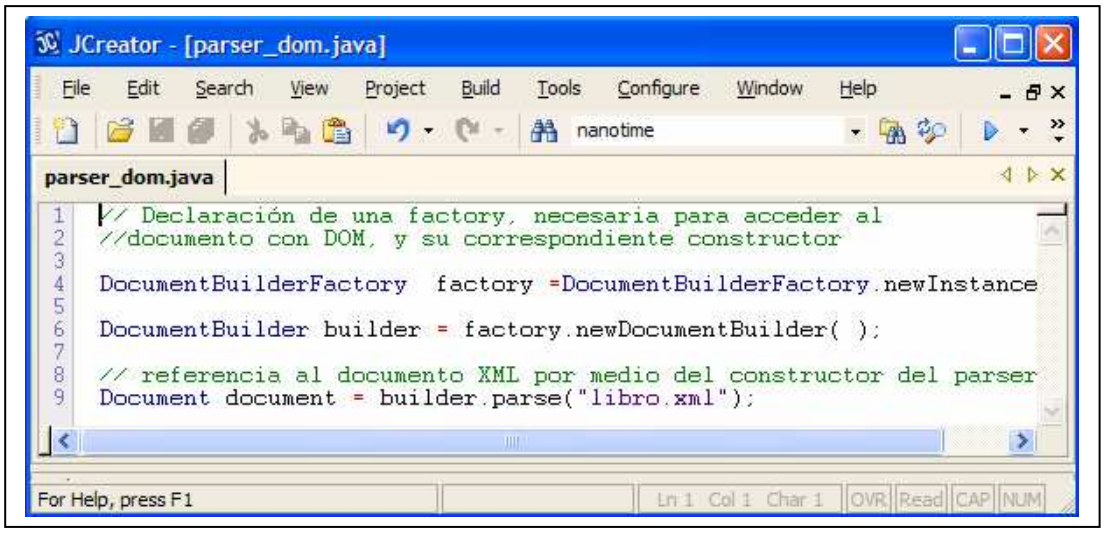

Figura 3.4. Declaración de la factoría para DOM

Una vez que se determina la localización del documento a analizar por medio del uso de la clases *DocumentBuilderFactory* y *DocumentBuilder* a través de su método *newDocumentBuilder(),* se puede llamar al método *parse()* el cual retorna el objeto documento (Document) resultante del análisis.

El parser DOM almacena la lista de hijos de un nodo en objetos de tipo lista de nodos (*NodeList)*, de clase similar al almacenamiento de los datos tipo arreglo de Java, la lista de nodos son objetos que, si se añade o elimina un nodo de la lista, dicho cambio se ve reflejado en el documento y viceversa. Esto puede dificultar el recorrido de la lista de nodos ya que su longitud puede variar mientras se está recorriendo, y los nodos pueden moverse

de un lugar a otro de la lista. La lista de nodos no son objetos seguros para la concurrencia, como casi todo lo demás en DOM, por lo que no es seguro leer y escribir simultáneamente en una lista de nodos.

En la Figura 3.5 se muestra el fragmento de código en el cual se recorre el árbol DOM para acceder a los nodos por medio del uso del método *getFirstChild()* para referenciar a los datos contenidos en cada nodo del árbol.

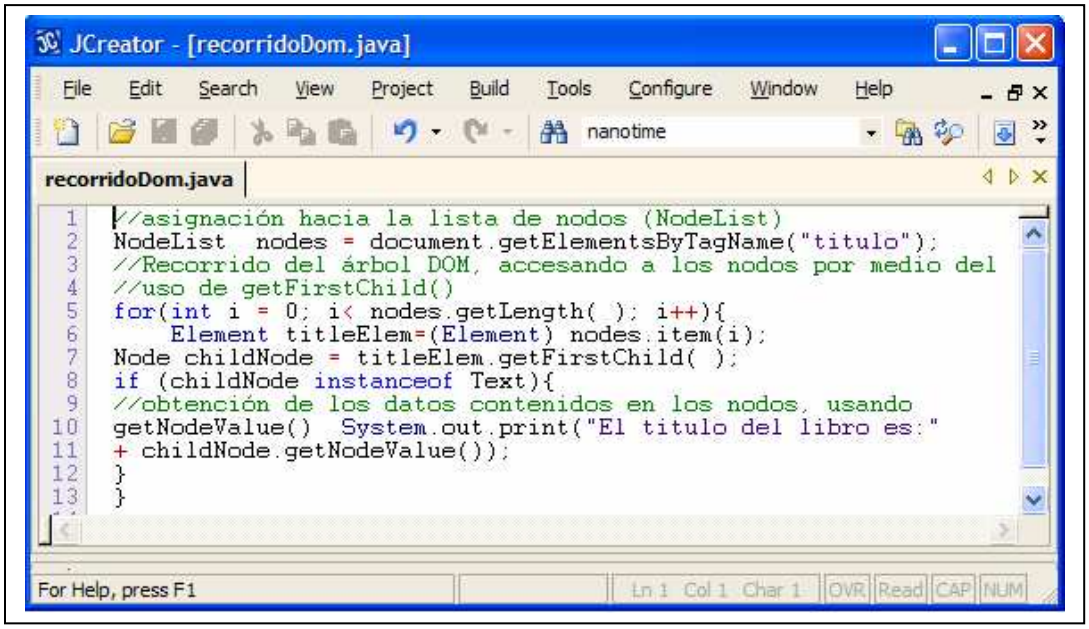

Figura 3.5. Recorrido de un árbol DOM

3.1.3. Parser JDOM.

El parser JDOM sirve para procesar documentos XML desde Java y simplifica dicha tarea en comparación a cómo se haría usando DOM, JDOM representa un documento XML como un árbol de objetos en memoria compuesto de elementos, atributos, comentarios, instrucciones de procesamiento, nodos de texto, entre otros. Como se muestra en la Figura 3.6 a diferencia de DOM los distintos tipos de nodos de un árbol están representados con clases concretas y no mediante el uso de interfaces, también se muestra la interacción de los elementos que forman el modelo JDOM para el desarrollo de una aplicación.

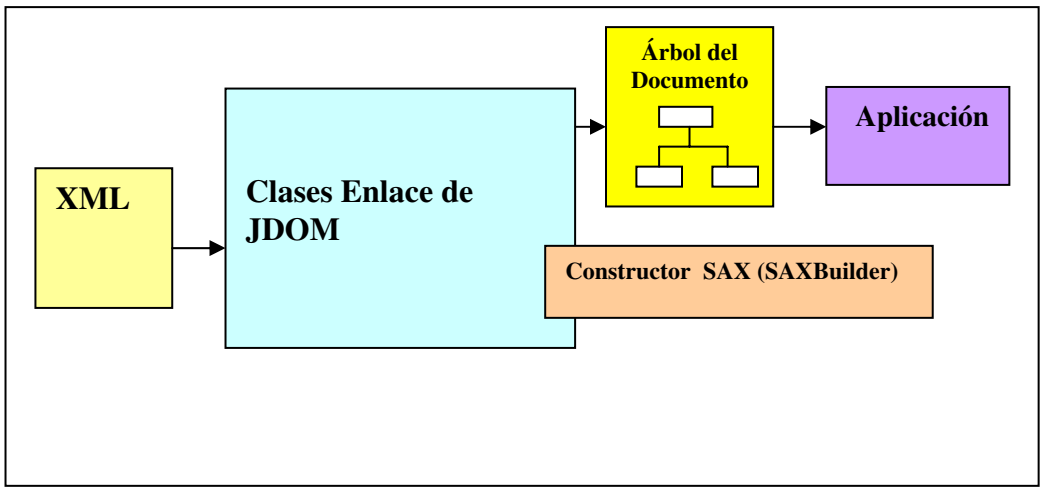

Figura 3.6. Modelo del parser JDOM

En JDOM no existe una interfaz Nodo o similar que todos los tipos de nodo implementen o extiendan, como JDOM está especialmente pensado para Java, sigue las convenciones de nombrado del lenguaje y utiliza las clases de biblioteca del JDK, por ejemplo, todas las clases principales de JDOM implementan los métodos Java *equals*(), *toString*() y *hashCode*(), además de implementar las interfaces *Cloneable* y *Serializable*.

Los hijos de un objeto de tipo documento (*Document)* se almacenan en un dato tipo lista para su uso es necesario referenciar al recurso *java.util.List* en vez de usar un dato de tipo lista de nodos referenciado en *org.w3c.dom.NodeList* como hacía DOM. JDOM no incluye un parser XML por sí mismo, viene empaquetado junto al parser *xerces* sin embargo es posible usar JDOM en conjunto con cualquier otro parser compatible.

Para el procesamiento del contenido de un documento XML en JDOM, existen tres clases que se pueden asociar a los objetos de un documento XML, por medio del uso de los paquetes o clases que se importan hacia el programa se pueden realizar diferentes actividades como se muestra en la Tabla 3.1.

Tabla 3.1. Uso de clases de JDOM

Clases	Uso
org.w3c.dom.Document	Se puede hacer referencia al documento completo XML.
org.w3c.dom.Node	Se puede hacer referencia a cada uno de los nodos DOM.
org.w3c.dom.Att	Se puede acceder a los atributos de los elementos contenidos en un documento XML.

Por medio del uso de los paquetes que se muestran en la Tabla 3.1 se puede acceder a las siguientes clases:

Clase *Element*:

Element (String nombre) es un constructor para crear etiquetas.

etiqueta.setText (String texto) para asignarle un contenido a la etiqueta.

etiqueta.addContent (Element hijo) el parámetro es el hijo que se desea que pertenezca de un determinado elemento.

elemento.addAttribute(String atributo, String valor) sirve para añadir un atributo.

removeAttribute (String atributo) se usa para borrar un atributo.

Clase *Document*

El constructor Document (Element raiz) se usa para crear un documento pasándole por medio del parámetro el elemento que se asigna como la raíz, mediante el uso de la instrucción root.setRootElement (Element nuevaRaiz) se puede cambiar el elemento raíz. Para generar una salida de datos del Documento ya sea en pantalla o archivo se puede usar la siguiente clase:

Clase *XMLOutputter*:

XMLOutputter (String espacio, Boolean nuevaLinea) en el primer parámetro se indica la cadena de caracteres deseado de espacios, y en el segundo parámetro se indica si se desea o no introducir nueva línea, la instrucción *linea.output(Document doc, OutputStream salida)* se utiliza para escribir el documento en un determinado archivo.

La clase *org.jdom.output.XMLOutputter* es la manera más común de serializar un documento XML en forma de archivo, o cualquier otro tipo de flujo de datos, en la Figura 3.7 se muestra el uso de la clase *XMLOutputter* para escribir un documento XML en la consola.

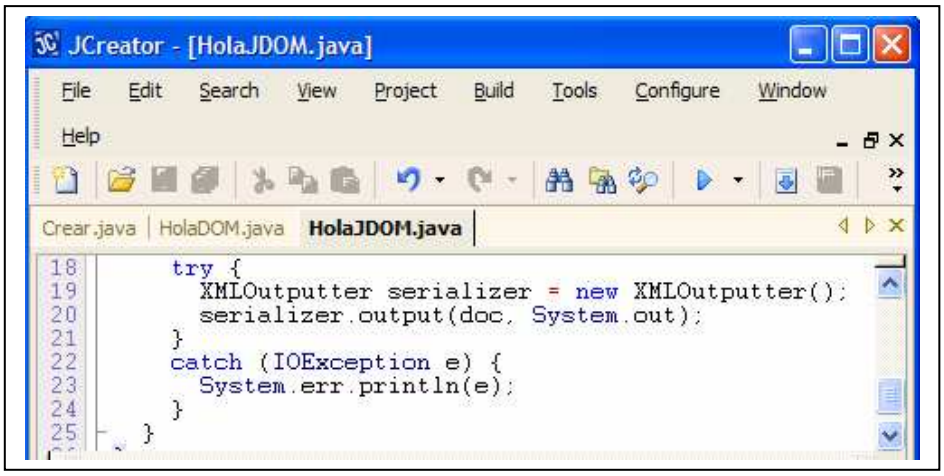

Figura 3.7. Uso de la instrucción *XMLOutputter*

No sólo se pueden escribir documentos completos, sino también partes de un documento, por ejemplo, en la Figura 3.8 se muestra el fragmento de código que escribe solamente el elemento "saludo":

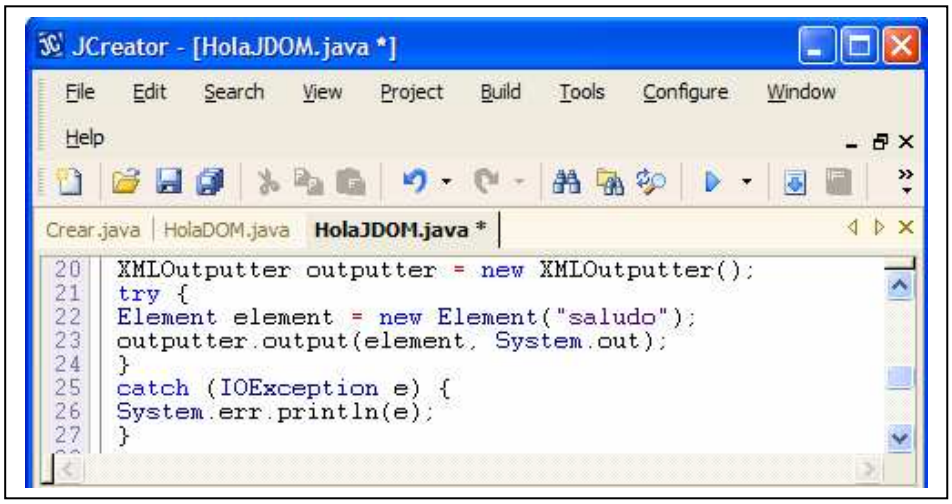

Figura 3.8. Salida a pantalla de un solo elemento

JDOM puede leer documentos XML de una amplia variedad de fuentes, como archivos *(java.io.File), URLs (java.net.URL)* o cualquier otro objeto al que se le pueda asociar un flujo de datos *(java.io.InputStream)* o un lector *(java.io.Reader)* en Java, además también pueden emplearse árboles SAX y/o DOM como origen de datos a través de *SAXBuilder* y *DOMBuilder* respectivamente también pueden emplearse orígenes de datos de SAX *(org.xml.sax.InputSource).*

La navegación por el árbol de objetos JDOM suele hacerse mediante el uso de objetos de tipo *Element*. La lista de hijos de un elemento determinado está disponible en forma de una lista, para su definición se requiere importar la clase *java.util.List,* se pueden implementar los métodos *getChilden()* y *getContent()*. El método *getChildren()* sólo devuelve los hijos directos de tipo *Element*, ignorando el resto de nodos, como los comentarios, las instrucciones de procesado o los nodos de texto para llegar hasta estos nodos hace falta emplear *getContent()* que devuelve todos los tipos de nodo.

Con el análisis de los componentes de los modelos de SAX, DOM y JDOM se puede observar que los parsers hacen uso de clases y librerías que se pueden importar hacia el programa mediante las cuales se realiza el procesamiento de documentos XML y estas

clases y librerías se implementan en Java, permitiendo una mayor flexibilidad para ejecutarse en diferentes plataformas.

3.2. Herramientas para construir documentos XML en Java.

El desarrollo y manipulación de documentos en XML no es tarea fácil, se necesitan aplicaciones que realicen las funciones de análisis, creación, manipulación entrada y salida, entre otras actividades, para su implementación se requieren tecnologías de Java y *Xerces* que se han desarrollado para facilitar la creación de aplicaciones XML, para las aplicaciones que se ejecutan en Java se requiere instalar la versión 1.4 o superior, se puede consultar el proceso de instalación en el anexo de instalación del JDK.

Existen varios analizadores XML de buena calidad y la mayor parte de ellos gratuitos. Uno de los más conocidos y usados es el *Xerces* cual hay versiones en Java, *Perl* y en C++, es adecuadamente rápido y además incorpora todos los últimos estándares del Consorcio *World Wide Web* (W3C) para el proceso de instalación ver el anexo instalación de *Xerces* otras herramientas que se pueden utilizar por ser XML un subconjunto de SGML son las siguientes:

Adept Editor: es una herramienta completa de alto costo, para construir documentos en XML se puede utilizar este software para definir las estructuras de datos y mostrar como deben aparecer en pantalla se puede encontrar información de *Adept Editor* para *windows* en la referencia [5].

FrameMaker + SGML 5.5: Provee control y flexibilidad, permitiendo al usuario escoger elementos, atributos y estructura de documentos de XML de una lista, además de publicar documentos en XML se pueden crear documentos de HTML, PDF, *Postscript* y SGML 5.5. Está disponible para *windows* y *macintosh* en la referencia [14].

XMetal: Es una herramienta avanzada de XML, su ambiente es parecido a un procesador de palabras, es fácil de usar. Está disponible en Windows en la referencia [16].

XML Pro: Es un editor de XML básico, permite crear y editar documentos utilizando menús y pantallas. Disponible para Windows en la referencia [16].

XML Spy: Permite editar archivos de XML, XHTML, Lenguaje de Hojas de Estilo Extensible (XSL) y definición de tipo de documento (DTD) adicionalmente provee tres vistas avanzadas de los documentos, una vista mejorada para edición estructurada, una vista de fuente con coloración de sintaxis para trabajo de bajo nivel y una vista integrada del navegador que soporta hojas de estilo de CSS y XSL se puede encontrar XML Spy para windows en la referencia [16].

4. APLICACIÓN.

En este capítulo se desarrollan aplicaciones en las cuales se pueden observar los recursos de software que requiere cada parser así como el desempeño de cada uno, también se especifica el código para la implementación y uso de las clases que hacen posible acceder y procesar documentos XML para cada parser en particular, para ello se hará el desarrollo de la misma aplicación en SAX, DOM y JDOM. El capítulo inicia con aplicaciones sencillas y al final se desarrolla una aplicación de búsqueda hacia datos almacenados en un documento XML conteniendo información de diferentes trabajadores.

4.1. Creación de documentos XML.

En la Figura 4.1 se muestra un documento XML sencillo, la primera línea es la declaración *<?xml version="1.0" encoding="UTF-8"?>* se indica:

- Información sobre la versión de XML que se esta utilizando (versión 1.0).

- Información sobre el tipo de codificación de caracteres que se está utilizando, (código ASCII de 7 bits) que es un subconjunto del código (Unicode) denominado UTF-8 (No es necesario declararlo ya que es el que los parsers manejan por defecto).

En la segunda línea esta el contenido “Hola” entre las etiquetas <saludo>.

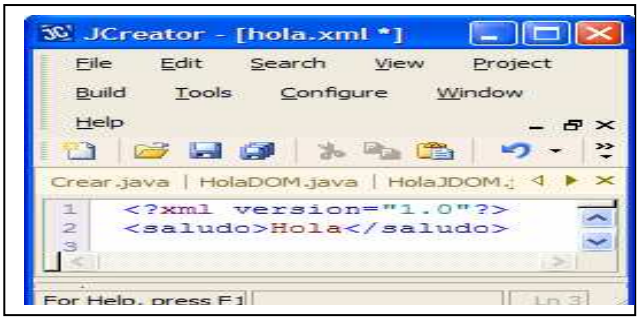

Figura 4.1. Documento XML con una etiqueta

Con el objetivo de mostrar un ejemplo sencillo que comúnmente es generar un mensaje de salida a pantalla, para ejemplificar la simplicidad del código que se usa con el parser JDOM en comparación al código que se requiere cuando se usa el parser DOM dado que no se pueden crear documentos XML con el parser SAX, en la Figura 4.2 se muestra el código necesario para crear nodos hacia un documento XML usando JDOM.

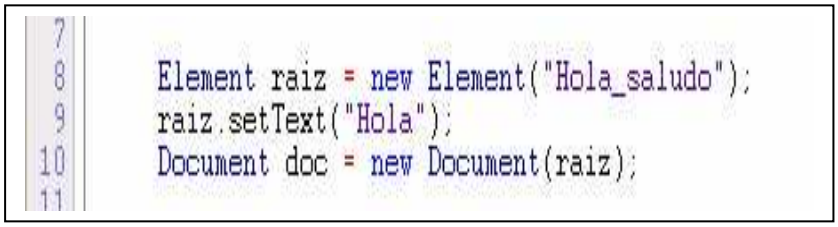

Figura 4.2. Creación de XML con un elemento con JDOM

El fragmento de código usando el parser JDOM sirve para generar el archivo XML con una etiqueta cuyo contenido es "Hola", se puede ver el anexo de aplicaciones para revisar el programa completo.

Versión DOM.

En la Figura 4.3 se muestra el uso del parser DOM para definir el elemento raíz que inicialmente no está asociado por lo cual se necesita ejecutar al constructor del documento para asociarlo como elemento raíz, se puede consultar el anexo de aplicaciones para revisar el programa completo.

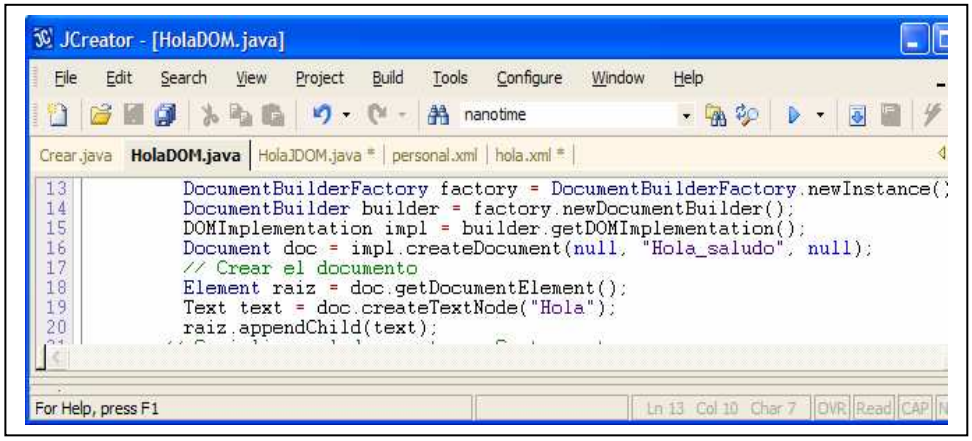

Figura 4.3. Creación de XML con un elemento usando DOM

Un segundo ejemplo para la creación de documentos XML con DOM será para la generación de la serie de fibonacci para diez valores, en la Figura 4.4 se muestra el ciclo que contiene las instrucciones para añadir cada nodo hijo al elemento raíz, de tal manera que cuando finaliza el ciclo se tienen los diez valores de la serie en el documento XML

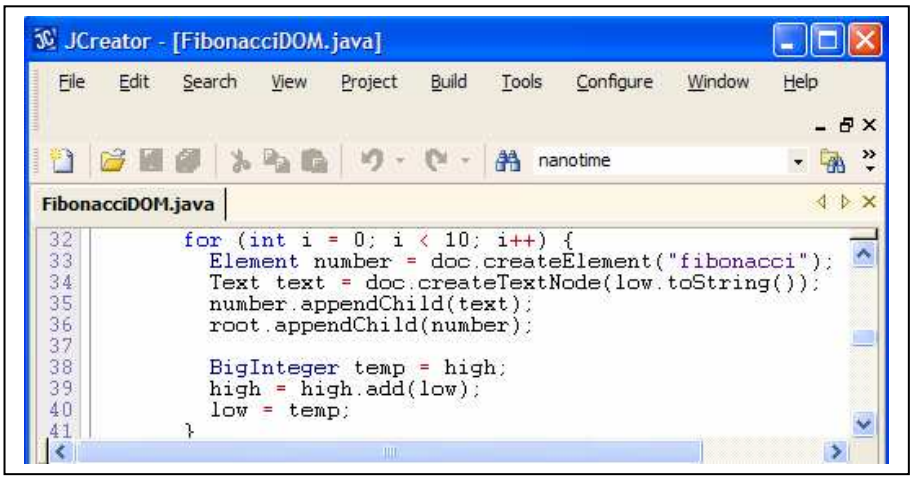

Figura 4.4. Creación de diez valores de la serie de fibonacci con DOM

En el fragmento de código, se usan la instrucción *number.appendChild(text)* para insertar un elemento hijo de tipo numérico a la estructura de árbol, la instrucción *root.appendChild(number)* sirve para establecer el elemento raíz de la estructura, se puede analizar el código completo en el anexo de aplicaciones.

4.2. Creación de documentos XML con JDOM.

En el proceso de obtener un documento XML el primer objetivo es contar con un objeto documento (Document) JDOM ya que es la clase básica que representa un documento XML, cuando no existen datos XML como punto de partida crear un documento JDOM es simplemente cuestión de invocar el construtor *Document doc=new Document (new Element (raiz)).*

JDOM es un conjunto de clases concretas y no un conjunto de interfaces esto significa que todos los códigos tan complicados que utilizan las factorías para construir los objetos necesarios para crear un elemento en DOM no se requieren en JDOM, simplemente se

desarrolla la operación nueva (new) en el objeto Document y se obtiene un tipo de dato Document que se puede utilizar.

En la Figura 4.5 se muestra el fragmento de código que genera un documento XML que contenga cinco números de la serie de fibonacci empleando el parser de JDOM:

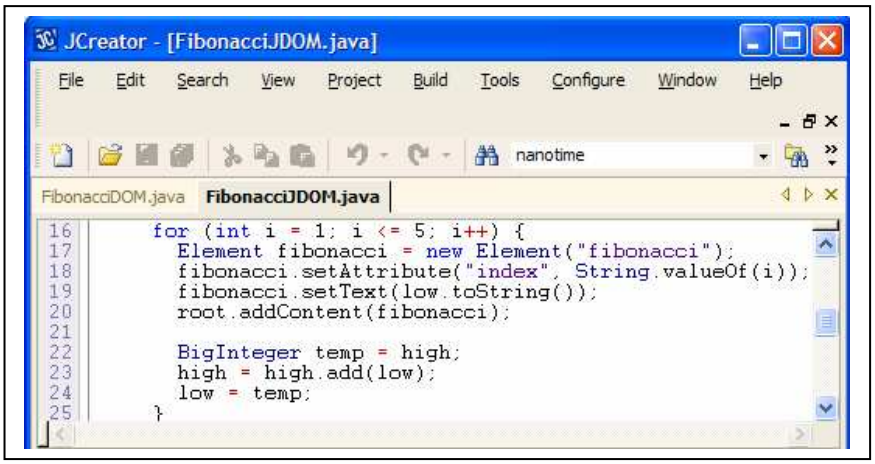

Figura 4.5. Creación de diez valores de la serie de fibonacci con JDOM

Mediante la sentencia *root.addContent (fibonacci)* contenida en el ciclo se asigna hacia la etiqueta raíz del documento XML los valores numéricos que forman la serie de fibonacci, con la sentencia *fibonacci.setAttribute("index", String.valueOf(i))* se asignan valores como atributos en las etiquetas del documento, a continuación se muestra una versión para obtener una salida formateada (cada etiqueta del archivo XML separadas por un salto de línea), para lo cual se usa la clase *XMLOutputter* permitiendo especificar un formato determinado, en cuyo parámetro se indica al constructor una instancia de la clase *Format.getPrettyFormat( )* el ejemplo de uso para obtener una salida formateada se muestra en las siguientes sentencias.

```
XMLOutputter serializer = new XMLOutputter(Format.getPrettyFormat());
serializer.output(doc, System.out);
```

4.3. Lectura de documentos XML con SAX.

Dado que con el parser SAX no se permite la creación de Documentos XML por ser de solo lectura, se utilizará el archivo XML como la fuente de datos a procesar, para la generar el documento XML se deben crear etiquetas cuyo contenido sean los valores de la serie de fibonacci este documento se puede realizar en el editor de java cuya extensión de archivo es xml como se puede apreciar en la Figura 4.6.

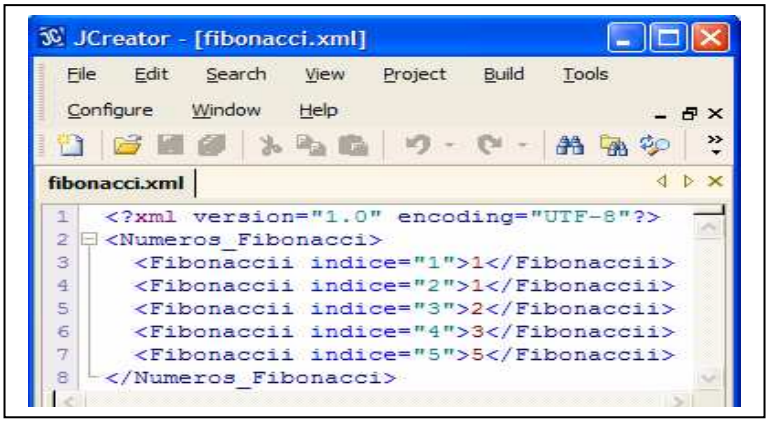

Figura 4.6. Archivo fibonacci.xml

4.4. Lectura de archivos XML.

Cuando se usa el parser SAX se deben usar las clases *SAXParser* y *SAXParserFactory,* la clase SAXParser contiene la instancia de la implementación del analizador utilizado y recupera esa instancia de la clase *SAXParserFactory*, para desplegar los valores numéricos de la serie de fibonacci a consola mediante la lectura del archivo XML con el uso de datos de tipo factoría del parser SAX, se muestra en la Figura 4.7 el ciclo que recorre el documento XML para listar su contenido (para ver el programa completo consultar el anexo de aplicaciones).

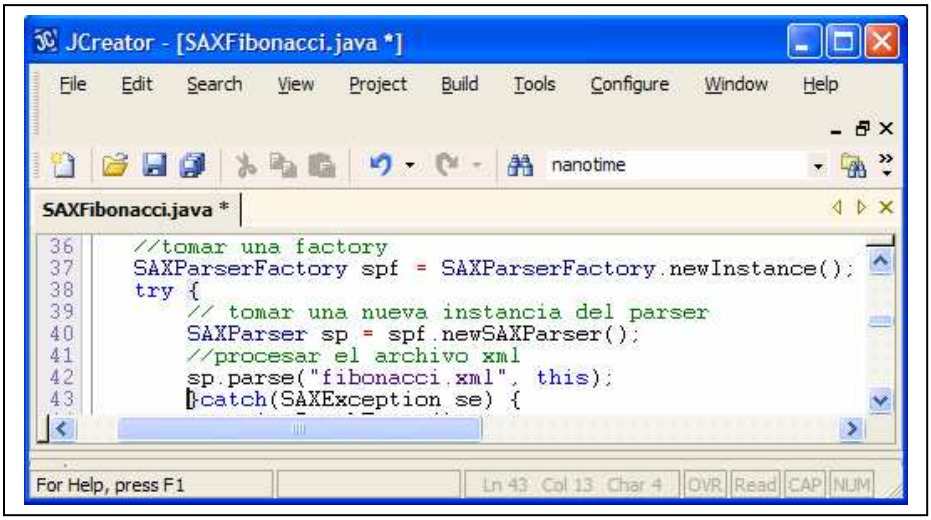

Figura 4.7. Uso de *SAXParserFactory*.

En el fragmento de código de la Figura 4.7 se muestra el uso de la clase *SAXParserFactory* para definir el dato mediante el cual se toma una nueva instancia del parser almacenada en el objeto y de esta manera acceder al archivo XML con la sentencia *sp.parse("fibonacci.xml", this).*

4.5. Lectura del Documento XML.

Para realizar la lectura del documento XML se desarrolla una aplicación para procesar empleados de una compañía cuyos datos son: nombre, tipo de empleado, seguro social y edad y están contenidos en el archivo *empledoos.xml*, para procesar el documento XML se implementan los métodos de lectura y búsqueda usando cada uno de los parsers, los programas de aplicación que se presentan usan el mismo documento XML y proporcionan la misma salida, permitiendo el análisis comparativo de los resultados obtenidos.

En la Figura 4.8 se muestra el archivo XML que contiene los datos de los empleados en sus correspondientes etiquetas: <nombre>, <clave>, <edad> y <tipo> este documento se usará para realizar el acceso a la información y desplegar su contenido a pantalla implementando el uso de los parsers SAX, DOM y JDOM.

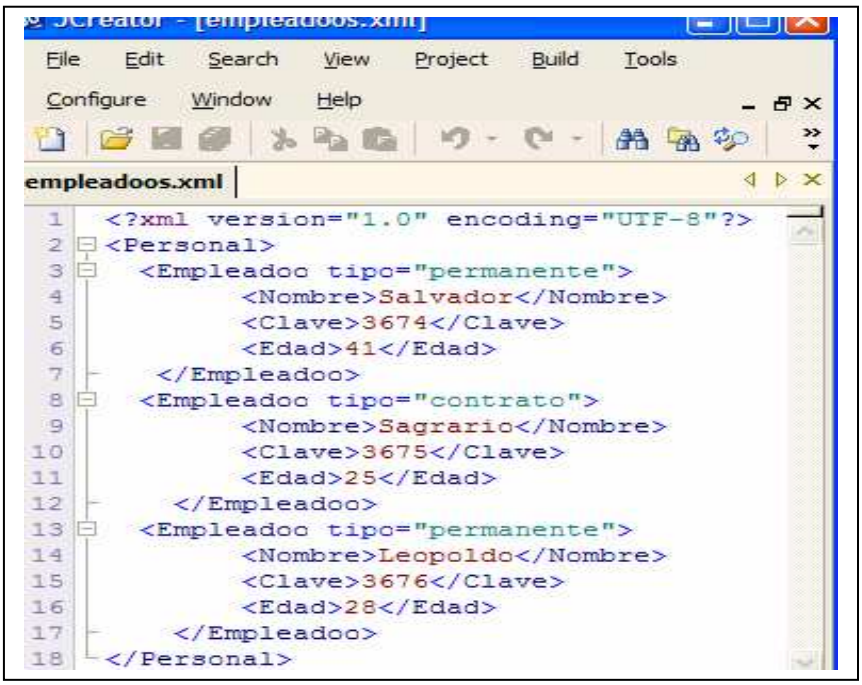

Figura 4.8. Archivo *empleadoos.xml*

Se implementa también el uso de la clase *Empleadoo* mostrada en la Figura 4.9 escrita en Java esta contiene los métodos *ponEdad ()*, *ponNombre()*, *ponClave(), ponTipo()* y su correspondiente constructor *Empleadoo*, en el cual por medio del objeto *this* se puede hacer referencia al contenido de cada etiqueta del documento XML. Esta clase se ejecuta en el programa *SAXParserLista* (ver anexo de Aplicaciones) el cual lista el contenido del documento XML a pantalla.

Empleadoo.java *

```
1  public class Empleadoo {
2      private String nombre;
3      private int edad;
4      private int clave;
5      private String tipo;
6      public Empleadoo(){
7      }
8      public Empleadoo(String nombre, int clave, int edad, String tipo) {
9          this.nombre = nombre;
10         this.edad = edad;
11         this.clave  = clave;
12         this.tipo = tipo;
13     }
14     public int tomarEdad() {
15         return edad;
16     }
17     public void ponerEdad (int edad) {
18         this.edad = edad;
19     }
20     public int tomarClave() {
21         return clave;
22     }
23     public void ponerClave(int clave) {
24         this.clave = clave;
25     }
26     public String tomarNombre() {
27         return nombre;
28     }
29     public void ponerNombre(String nombre) {
30         this.nombre = nombre;
31     }
32     public String tomarTipo() {
33         return tipo;
34     }
35     public void ponerTipo(String tipo) {
36         this.tipo = tipo;
37     }
38     public String toString() {
39         StringBuffer sb = new StringBuffer();
40         sb.append("Datos de Empleados -"+" \n");
41         sb.append("Nombre:" + tomarNombre());
42         sb.append(", ");
43         sb.append("Tipo:" + tomarTipo());
44         sb.append(", ");
45         sb.append("Clave:" + tomarClave());
46         sb.append(", ");
47         sb.append("Edad:" + tomarEdad());
48         sb.append(".");
49         return sb.toString();
50     }
51 }
```

Figura 4.9. Archivo *Empleadoo.java*

En la clase *Empleadoo* se usan los métodos *tomarNombre(), tomarTipo()*, *tomarClave()* y *tomarEdad()* que permiten interactuar con las aplicaciones de acceso y búsqueda y con los datos contenidos en las etiquetas del archivo XML, de tal manera que sirve como interfaz para dar servicio a las aplicaciones cliente, en cuyo código se usan los diferentes parsers.

4.6. Lectura de documentos XML con SAX.

La secuencia de la lógica del funcionamiento el parser SAX se realiza mediante el uso de los manejadores de evento *startElement* y *endElement,* los cuales se muestran en la Figura 4.10, ya que SAX no carga el documento completo sino que para su procesamiento se ejecutan eventos en el tiempo en que se va accesando al documento XML.

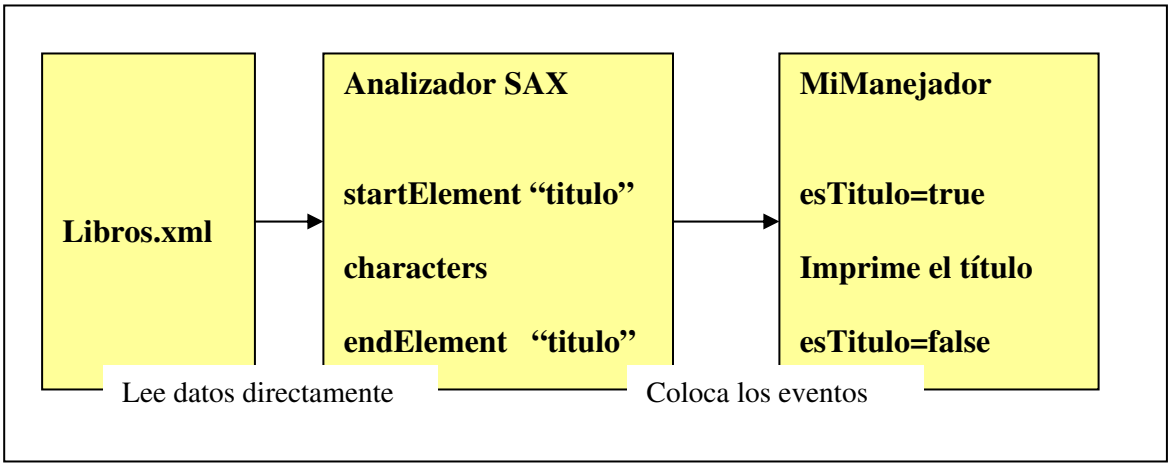

Figura 4.10. Analizadores de SAX

Mediante el uso de la factoría del parser SAX como se muestra en la figura 4.11 se declara el objeto que permite instanciar al *SAXParser (SAXParser sp = spf.newSAXParser()),* lo cual permite procesar los datos contenidos en el documento *empleadoos.xml*, con la instrucción *sp.parse("empleadoos.xml", this).*

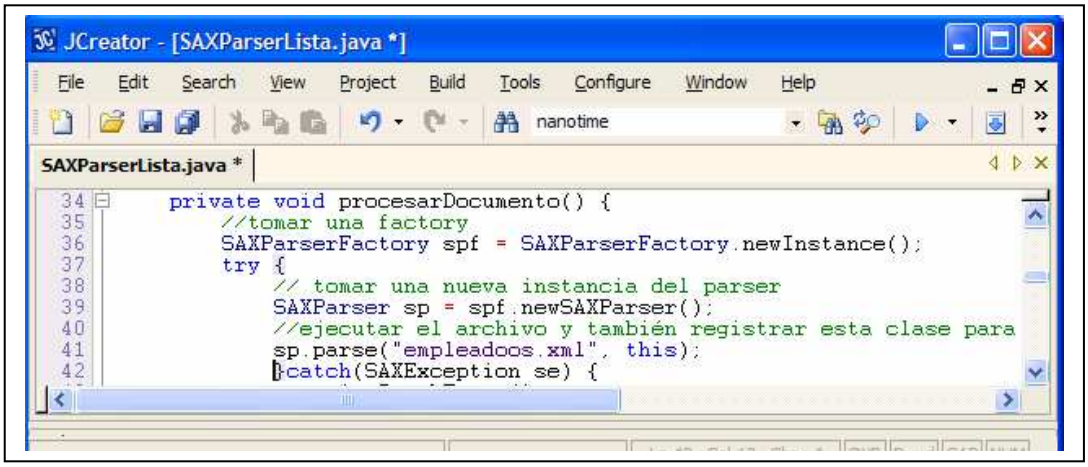

Figura 4.11. Método *procesarDocumento(),* aplicación despliega de SAX

En la Figura 4.12 se muestra la implementación de los manejadores de evento *startElement* y *endElement* de la aplicación mediante los cuales se acceda a los datos del archivo *empleadoos.xml*.

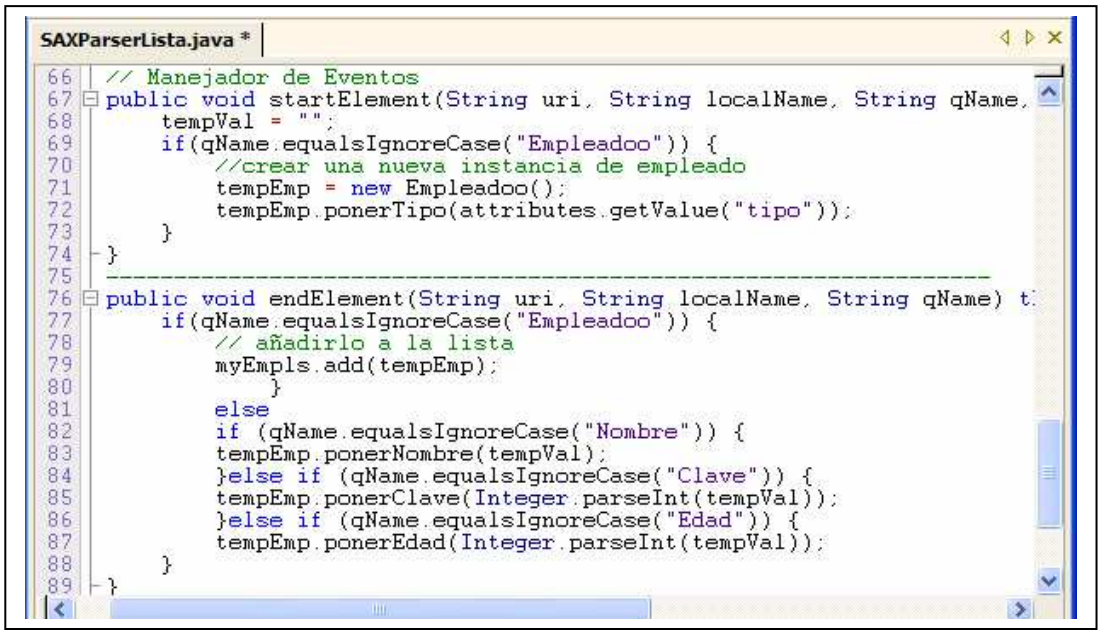

Figura 4.12. Manejadores de evento*: startElement( ) y endElement( )*

4.7. Lectura de documentos XML usando el parser DOM.

En la Figura 4.13 se muestra el método *procesarDocumento()* en el cual en primera instancia se referencia hacia el elemento raíz del documento XML con la instrucción *Element docEle = dom.getDocumentElement()*, con el objetivo de iniciar el recorrido y posteriormente desplegar a pantalla su contenido.

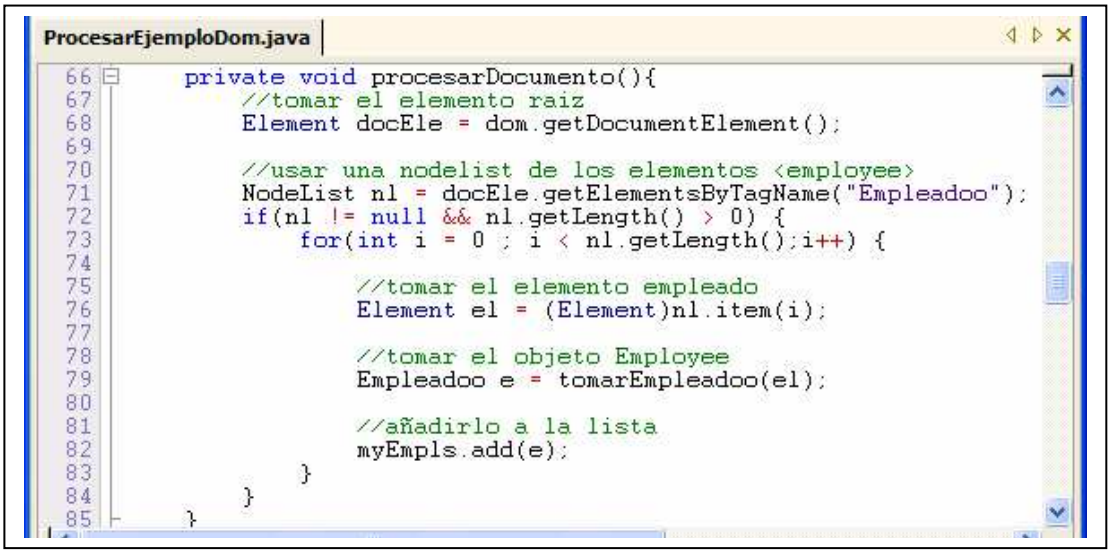

Figura 4.13. Método *procesarDocumento()*, aplicación despliega de DOM

Como se puede ver antes de entrar al ciclo que recorre el árbol, es necesario definir un tipo de dato *NodeList* para asignar a los elementos a partir de la etiqueta raíz "*Empleadoo*", mediante *docEle.getElementsByTagName("Empleadoo")* después se compara la longitud de la lista, de tal manera que si es diferente de cero entra al recorrido usando para ello un ciclo en este caso para acceder a la información de cada etiqueta del documento XML.

4.8. Lectura de documentos XML con JDOM.

Una de las ventajas principales de JDOM es que no se requiere usar factorías u otros modelos de programación para crear datos tipo *Document* o constructores del Documento, se puede comparar con DOM pero no se puede comparar con SAX ya que este último es un parser de solo lectura. JDOM se apoya en los parsers SAX y DOM de *Xerces* incluye adaptadores para los parsers más comunes en el paquete *org.jdom.adapters*.

Para leer los datos contenidos en un documento XML, JDOM se apoya en SAX a través de *SAXBuilder* lo común es usar JDOM en conjunción con un parser SAX pues son más eficientes que DOM al ocupar menos memoria (Los parsers DOM suelen cargar el documento entero en memoria, en forma de un árbol de objetos DOM para poder realizar el proceso).

4.9. Navegación por árboles JDOM.

En el uso del parser JDOM para el procesamiento de documentos XML no es necesario el uso de una factoría para obtener un constructor mediante se hace referencia al archivo de XML en su lugar se define un constructor de tipo *SAXBuilder*, cabe mencionar que es importante implementar un manejador de errores para evaluar situaciones en los que no exista el archivo que se instancia (*try - catch(Exception ioe)*).

En la Figura 4.14 se muestra el método *procesarDocumento()* para leer un documento usando un dato tipo lista (*List*) y almacenar los nodos hijos de la etiqueta raíz (E*mpleadoo*) del archivo *empleadoos.xml* mediante el uso de un ciclo se lista el contenido de los elementos, para acceder a los datos instanciando la clase *Empleadoo.java* mediante el uso del método *tomarEmpleadoo(el)*.

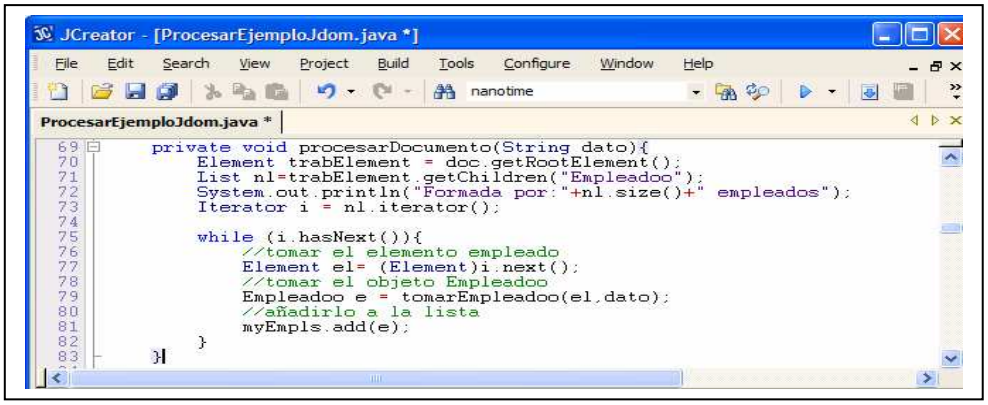

Figura 4.14. Método *procesarDocumento()* aplicación despliega de JDOM

En el código del método se hace referencia al nodo raíz, con la instrucción *Element trabElement = doc.getRootElement()* en el objeto de tipo lista se obtienen los nodos hijos, de tal manera que estos se cuentan con la función *size()* haciendo uso de otra función asociada a los datos tipo lista (*iterator())* para poder controlar la cantidad de iteraciones necesarias para recorrer el árbol JDOM.

4.10. Búsqueda en Documentos XML.

Con el objetivo de realizar la búsqueda de datos requeridos hacia un archivo XML, se usan nuevamente los archivos *empleadoos.xml* y *Empleadoo.java* en las aplicaciones de los parsers SAX, DOM y JDOM, de tal manera que comparten una estructura estandarizada con el fin de que permita un análisis comparativo y evaluación de código usando un mismo formato.

4.11. Búsqueda en Documentos XML usando el parser SAX.

En la Figura 4.15 se muestran los manejadores de Eventos*: public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException, y public void endElement(String uri, String localName, String qName)* para acceder a la información mediante el parser SAX y realizar la búsqueda de tal manera que despliegue los datos a buscar.

Figura 4.15. Métodos principales de la aplicación búsqueda con SAX

A diferencia de los manejadores de eventos que se usan para desplegar los datos contenidos en el archivo XML en este caso la complejidad es mayor para implementar la aplicación de búsqueda ya que el manejador de eventos *endElement ( )* realiza una evaluación para detectar el dato que se requiere en la búsqueda, así mismo se instancian los métodos *ponerNombre(), ponerClave(), ponerEdad(),* cuyo código esta implementado en la clase *Empleadoo.java*.

4.12. Búsqueda en Documentos XML usando DOM.

DOM es una especificación W3C presentada como "un interfase neutral del lenguaje y la plataforma que permitirá a los programas acceder dinámicamente y actualizar el contenido, la estructura y el estilo de documento" esencialmente es una estructura de datos en forma de árbol y un conjunto de métodos para acceder y editar esa estructura, el modelo de documento puede accederse aleatoriamente y ser procesado múltiples veces, en la Figura 4.16 se ilustran los elementos del árbol DOM que se genera a partir de los datos contenidos en el archivo *empleadoos.xml*.

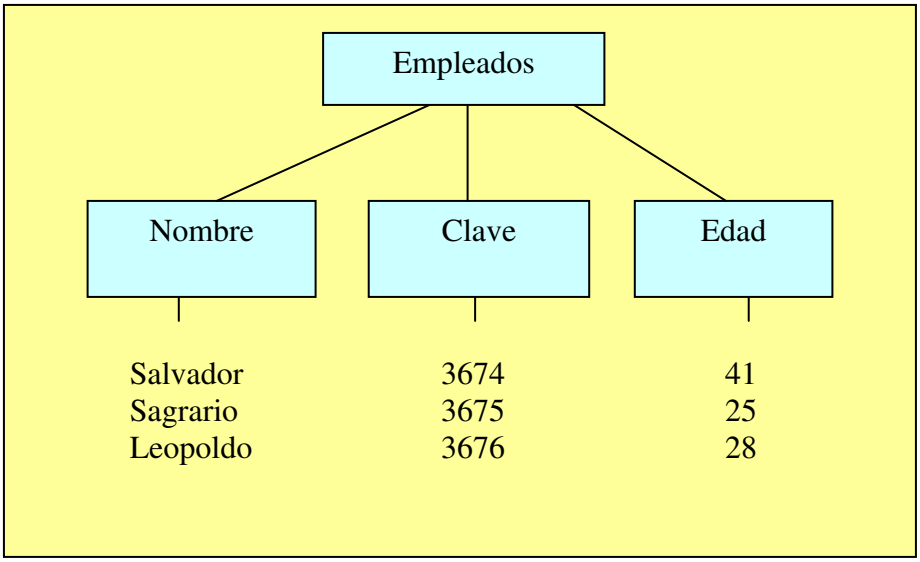

Figura 4.16. Árbol DOM

Cuando se usa el parser DOM la aplicación tiene que acceder o editar la representación en memoria del documento fuente, para realizar la búsqueda mediante el parser DOM para analizar y cargar en memoria el documento *empleadoos.xml*, se pasa a través del árbol DOM y se listan los datos requeridos en formato de texto, esta aplicación hace referencia al árbol DOM mediante el uso del constructor del documento desde el archivo de entrada XML y se obtienen todos los elementos del nodo raíz *Empleadoo* (<Nombre>, <Clave>, <Edad> y <Tipo> de empleado).

En la Figura 4.18 se despliega el contenido del método *procesarArchXml( )* la implementación hace referencia al nombre de etiqueta de los datos contenidos en el documento XML usando una factoría para definir un objeto con la instrucción (*DocumentBuilderFactory* dbf = *DocumentBuilderFactory.newInstance())* con el cual se obtiene una instancia del constructor *document* con la instrucción *DocumentBuilder* db = dbf.*newDocumentBuilder()*, para instanciar el método parser con el que se genera el árbol, pasando como argumento el archivo fuente *empleadoos.xml*.

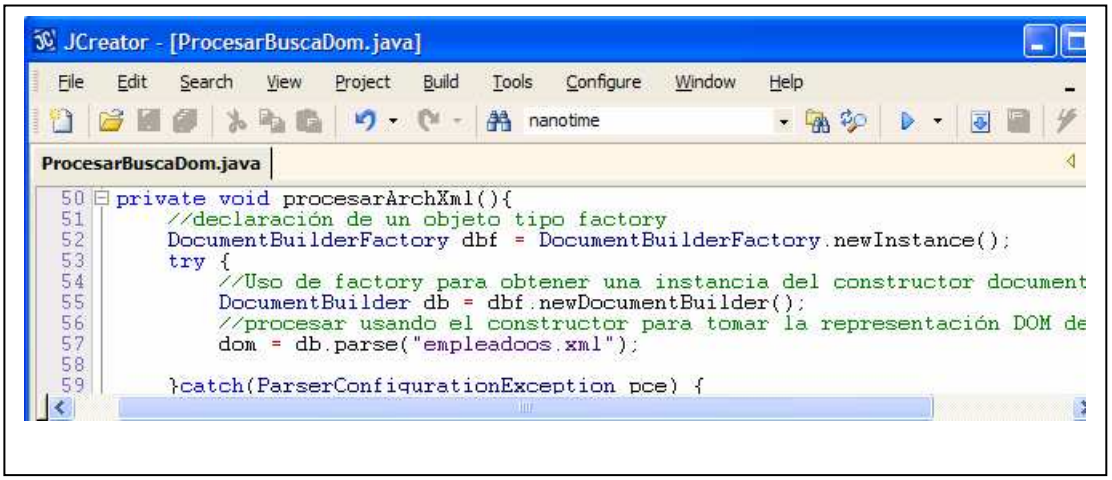

Figura 4.17. Método *procesarArchXml( )*, aplicación búsqueda DOM

En el método *procesarDocumento( )* que se lista en la Figura 4.19 se toma el elemento raíz mediante la instrucción *Element docEle = dom.getDocumentElement(),* de tal manera que se requiere usar el objeto de tipo elemento para asignar los nodos hacia un tipo de dato lista de nodos (*NodeList*), una vez que el objeto contiene esta lista se controla la cantidad de iteraciones que realiza el ciclo con el fin de tomar los datos mediante el uso de la clase *Empleadoo* se puede analizar el código completo en el anexo de aplicaciones.

ProcesarBuscaDom.java

```
74  private void procesarDocumento(String dato){
75      //tomar el elemento raiz
76      Element docEle = dom.getDocumentElement();
77
78      //usar una nodelist (lista de nodos) de los elementos <Empleadoo>
79      NodeList nl = docEle.getElementsByTagName("Empleadoo");
80      if(nl != null && nl.getLength() > 0) {
81          for(int i = 0 ; i < nl.getLength();i++) {
82
83              //tomar el elemento empleado
84              Element el = (Element)nl.item(i);
85
86              //tomar el objeto Empleadoo
87              Empleadoo e = tomarEmpleadoo(el, dato);
88
89              //añadirlo a la lista
90              myEmpls.add(e);
91          }
```

Figura 4.18. Método procesarDocumento( ) aplicación búsqueda DOM

4.13. Búsqueda en Documentos XML usando JDOM.

JDOM proporciona una forma de representar un documento para lectura, manipulación y escritura de manera fácil y eficiente es un parser alternativo a DOM y SAX, aunque se integra bien con ambos el siguiente programa usa el parser JDOM para analizar y cargar en memoria el documento *empleadoos.xml* que describe un conjunto de datos de empleados, luego pasa a través del árbol JDOM y los lista en formato texto usando la clase *SAXBuilder* en esta aplicación se realiza un procedimiento similar al usado por el parser DOM ya que para el procesamiento, se usa un constructor de documentos desde el archivo de entrada XML.

El método *procesarArchXml ( )* que se lista en la Figura 4.20 se observa que antes de ejecutar el método *procesarDocumento( )* se genera el árbol JDOM mediante el constructor (*build*) del documento, el cual se puede instanciar con la clase *SAXBuilder* pasando por argumento el archivo fuente a ser procesado *empleadoos.xml*.

ProcesarBuscaJdom.java

```
67      private void procesarArchXml(){
68
69          try {
70
71              SAXBuilder constructor=new SAXBuilder(false);
72              //usar el parser Xerces
73              doc=constructor.build("empleadoos.xml");
74
75          }catch(Exception ioe) {
```

Figura 4.19. Método *procesarArchXml ( )*, aplicación de búsqueda con JDOM

En el método *procesarDocumento( )* que se muestra en la Figura 4.21 se llevan a cabo las iteraciones para acceder a los nodos del árbol JDOM implementando el uso de un ciclo, este método es parecido al usado por el parser DOM ya que como se mencionó JDOM y DOM usan una estructura de árbol en memoria que contiene los datos del archivo fuente.

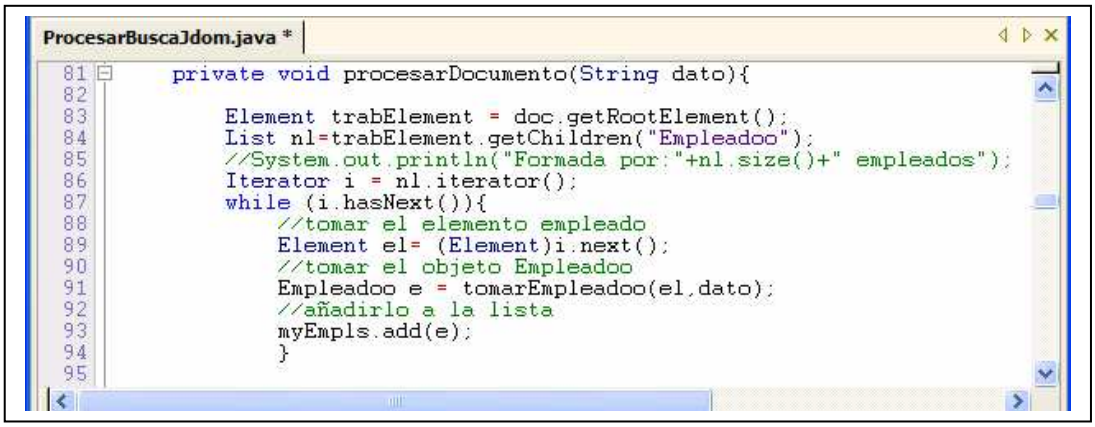

Figura 4.20. Método *procesarDocumento( )* aplicación de búsqueda con JDOM

5. RESULTADOS.

El análisis del rendimiento de los parser SAX, DOM y JDOM se realizó evaluando el código para la aplicación de búsqueda se obtuvieron valores que se graficaron para observar la velocidad de procesamiento mayor de JDOM en comparación a los parsers SAX y DOM, mediante la implementación de librerías que se importan desde el Xerces, permitiendo el uso del modelo de la programación orientada a objetos simplificando el acceso a los datos contenidos en el documento XML.

En la realización de la evaluación se comparó el rendimiento de SAX, DOM y JDOM aplicados en el proceso búsqueda hacia el documento *empleadoos.xml* se midió el tiempo de procesamiento cuyos resultados se muestran en la Tabla 5.1.

Los parser SAX, DOM y JDOM son adaptables a las características de la computadora y a la estructura de los datos lo cual también permitió evaluar la cantidad de recursos de código que cada parser requiere para ejecutar las aplicaciones estandarizando las estructuras y métodos utilizados.

5.1. Búsqueda en Documentos XML usando SAX.

Para realizar el procesamiento del archivo XML una vez instalado el API *Xerces* se usan las clases del parser y una vez implementada la práctica realizar pruebas y documentar los resultados, en la Figura 5.1 se muestra el fragmento de código para ejecutar el método principal *procesarDocumento( )*en el cual se recorre el árbol *JDOM* para realizar la búsqueda de datos.

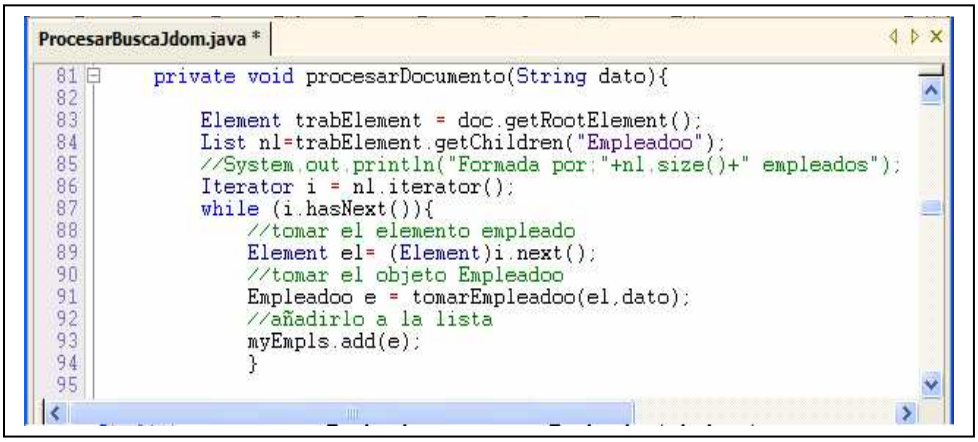

Figura 5.1. Método *procesarDocumento( )* aplicación de búsqueda con SAX

Ejecución:

En la programación de los eventos de SAX para procesar XML el analizador reportó los eventos de inicio y fin de los elementos de la aplicación mientras pasa a través del documento sin construir una estructura interna, debido a estas características se reducen los recursos de código por lo que se puede usar este analizador para el procesamiento de documentos grandes, en la Figura 5.2 se muestra la salida generada de la búsqueda del elemento nombre que fue la opción del usuario.

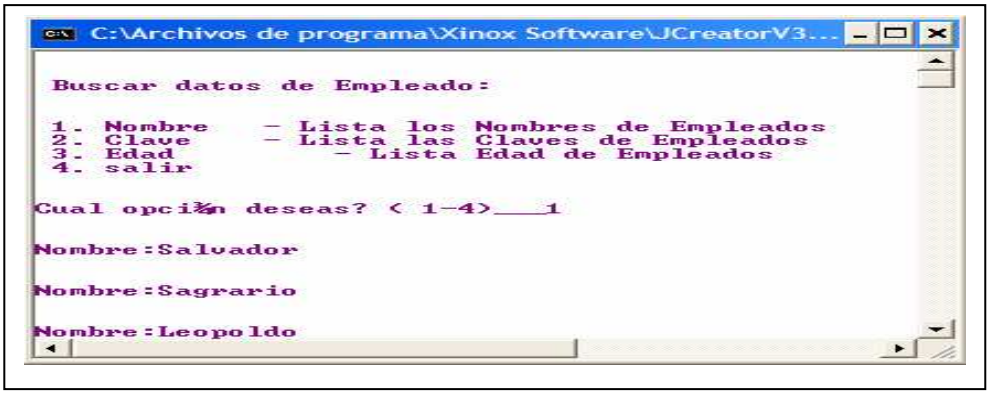

Figura 5.2. Búsqueda usando SAX

5.2. Búsqueda en Documentos XML usando DOM.

El procesamiento de documentos XML basados en marcas (etiquetas) facilita el intercambio de información de manera clara y ordenada se almacenan los datos en estructuras bien definidas nombrando cada etiqueta de tal manera que los datos de los empleados sean fáciles de gestionar, en la Figura 5.3 se muestra el método *runEjemplo(String dato)* que contiene la ejecución de los métodos principales *procesarArchXML()el cual crea el árbol DOM* y *procesarDocumento(dato)* que recorre el árbol DOM.

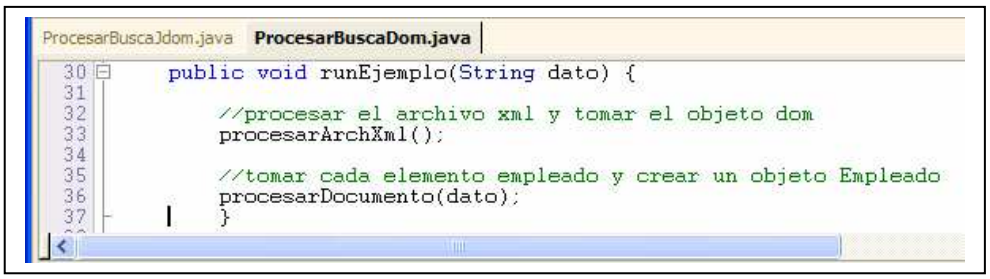

Figura 5.3. Método *runEjemplo(String dato)* aplicación búsqueda con DOM

Ejecución.

El parser DOM procesa los datos del documento XML usando una estructura de árbol por lo cual se desarrolla un ciclo para acceder a cada nodo (padre e hijo) DOM permite que una aplicación tenga acceso aleatorio al documento con estructura jerárquica, si el archivo XML es más grande también se incrementará la memoria utilizada por lo cual para aplicaciones grandes y complejas no es recomendable su uso en la Figura 5.4 se muestra el resultado de la ejecución de la búsqueda al elegir la opción dos para buscar la clave en el documento *empleadoos.xml*.

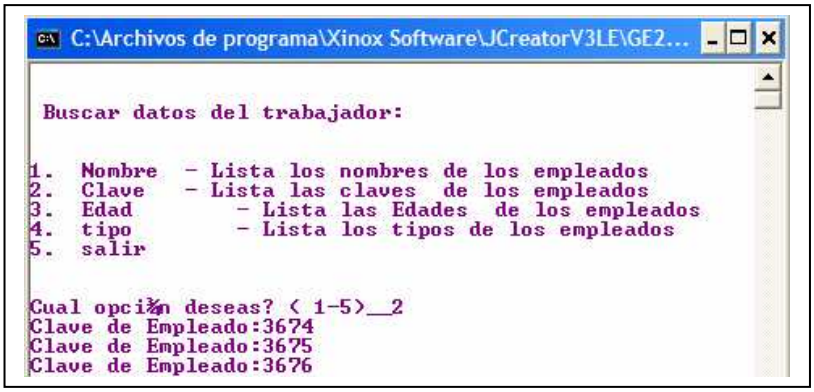

Figura 5.4. Búsqueda usando DOM.

5.3. Búsqueda en Documentos XML usando JDOM.

La característica importante de JDOM es que no usa factorías ni otros modelos avanzados de programación para crear un tipo de dato *Document*, lo cual si se requiere en el uso del parser DOM para ambos parser se genera una estructura de árbol para el acceso de datos hacia el documento XML en la Figura 5.5 se muestra el método *runEjemplo()* el cual contiene los métodos principales *procesarArchXml()* el cual crea el árbol JDOM y *procesarDocumento(dato)* que recorre el árbol JDOM para realizar la búsqueda, en la Figura 5.6 se despliega la ejecución de la búsqueda usando el parser JDOM.

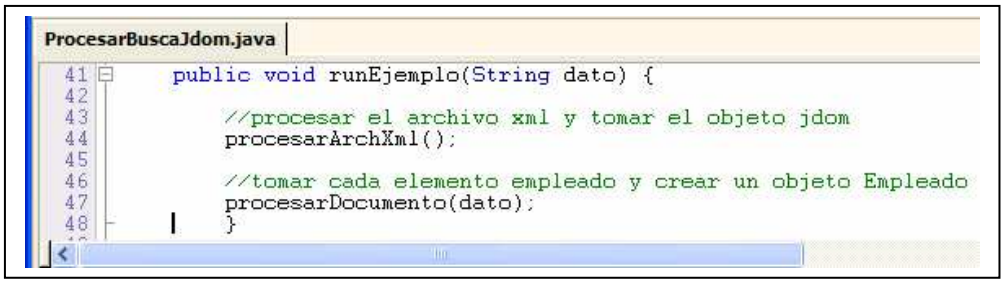

Figura 5.5. Método *runEjemplo( )*

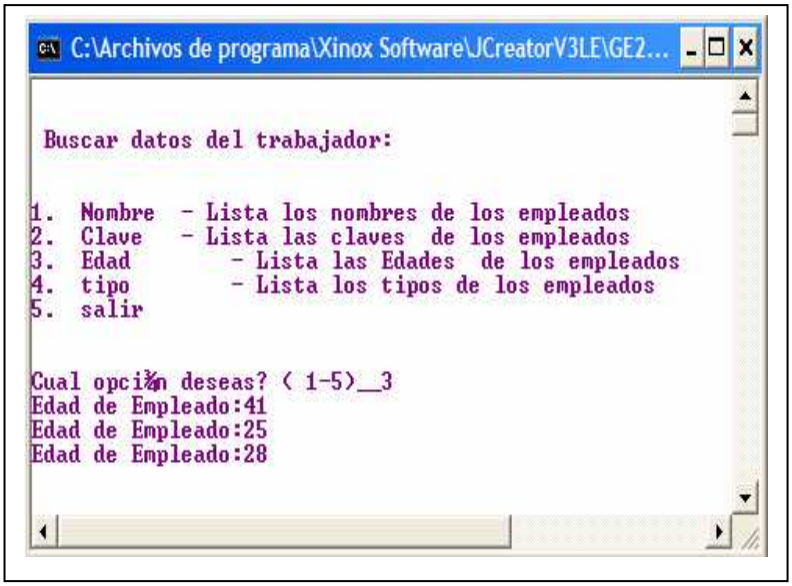

Figura 5.6. Búsqueda usando JDOM.

5.4. Comparación de rendimiento de los parser SAX, DOM y JDOM.

Para realizar las evaluaciones se compara el rendimiento de los parser SAX, DOM y JDOM en las aplicaciones de búsqueda para procesar el documento *empleadoos.xml* definido anteriormente se usa la implementación de los parser de *Xerces*, en esta evaluación se midió el tiempo para procesar el documento XML con tres, seis, nueve y trescientos treinta registros de empleados, cada documento ha sido procesado mil veces en diez ejecuciones el tiempo medido se suma al tiempo del sistema y es devuelto por el método *nanoTime()* dividido por el número total de documentos procesados.

En la Tabla 5.1 se muestran los tiempos de procesamiento que consumió cada parser y el número de registros contenidos en el documento XML.

Tabla 5.1. Tiempo de procesamiento usando SAX, DOM y JDOM

Número de registros procesados	Tiempo en nanosegundos para SAX	Tiempo en nanosegundos para DOM	Tiempo en nanosegundos para JDOM
3	58	80	66
6	72	90	70
9	72	97	75
18	105	119	88
100	155	186	140
330	185	236	203
500	247	297	250
1000	525	390	350

Se analizó el desempeño relativo de los parser ejecutando las aplicaciones de búsqueda el propósito de las evaluaciones realizadas es principalmente ilustrar el rendimiento de las diferentes técnicas del procesamiento del documento XML con SAX, DOM y JDOM, los resultados presentados pretenden subrayar la diferencia entre la facilidad de uso y el rendimiento de los modelos de procesamiento elegidos, en las Figuras 5.7, 5.8, 5.9, 5.10 y 5.11 se muestra el tiempo consumido por cada parser considerando la cantidad de registros contenidos en el documento XML.

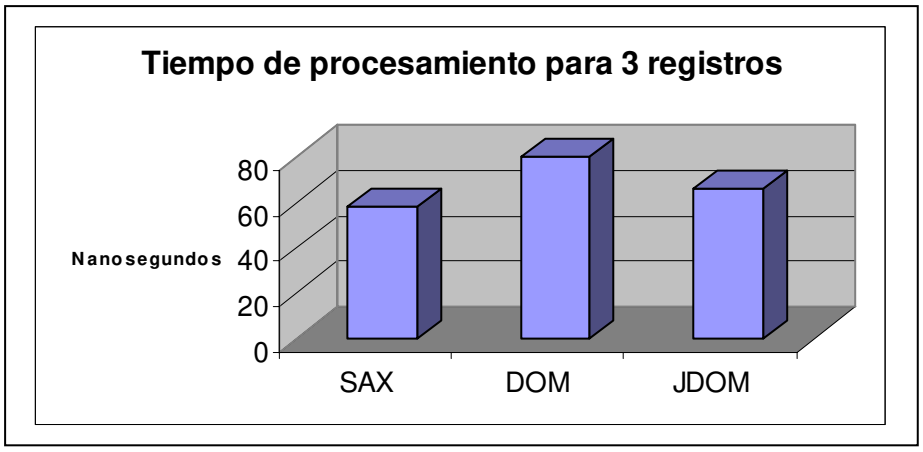

Figura 5.7. Tiempo de procesamiento del documento XML para tres registros

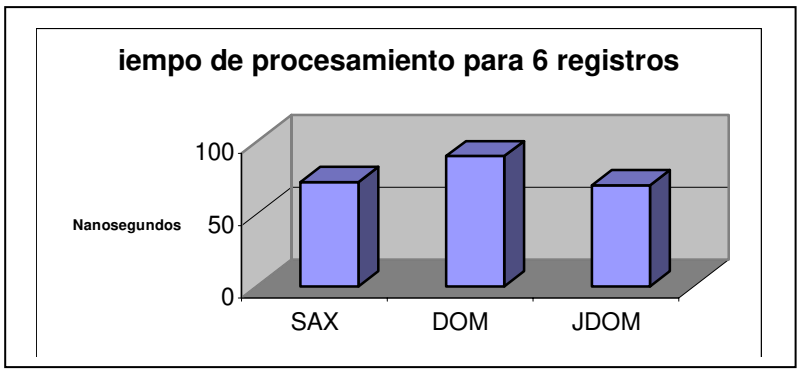

Figura 5.8. Tiempo de procesamiento del documento XML para seis registros

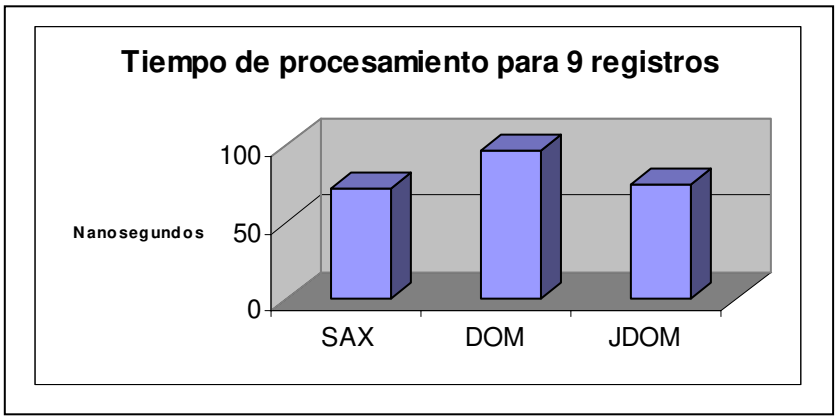

Figura 5.9. Tiempo de procesamiento del documento XML para nueve registros

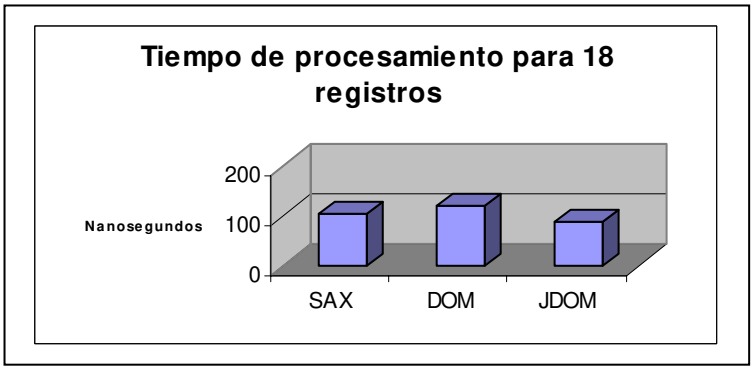

Figura 5.10. Tiempo de procesamiento del documento XML para diez y ocho registros

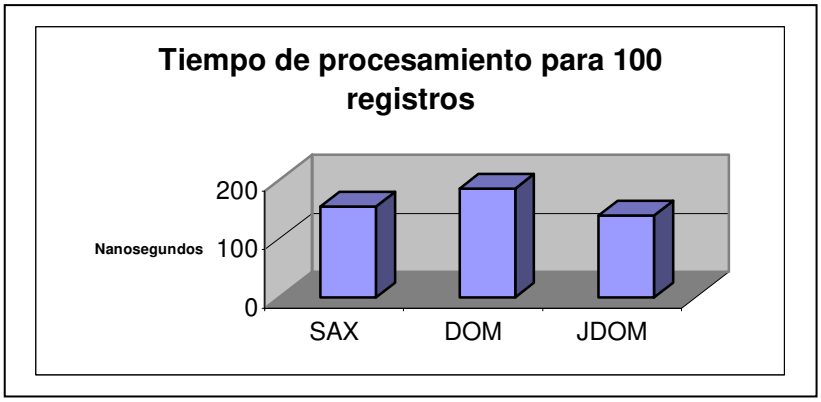

Figura 5.11. Tiempo de procesamiento del documento XML para cien registros

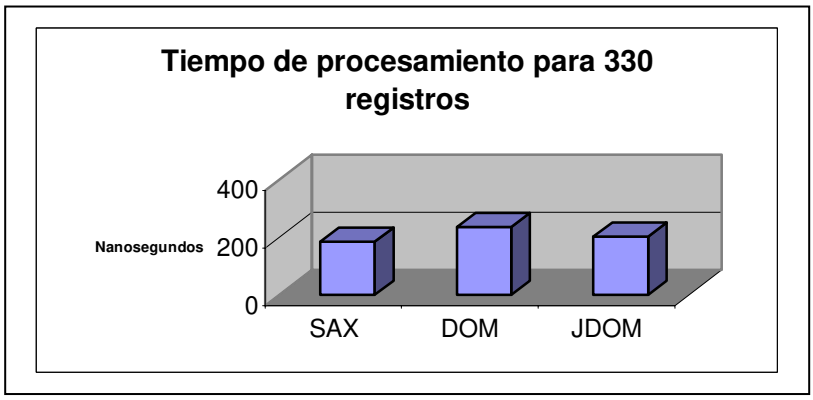

Figura 5.12. Tiempo de procesamiento del documento XML para trescientos treinta registros

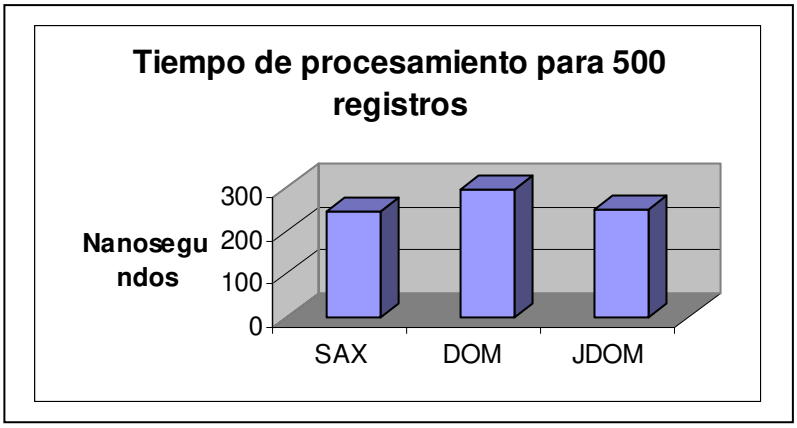

Figura 5.13. Tiempo de procesamiento del documento XML para quinientos registros

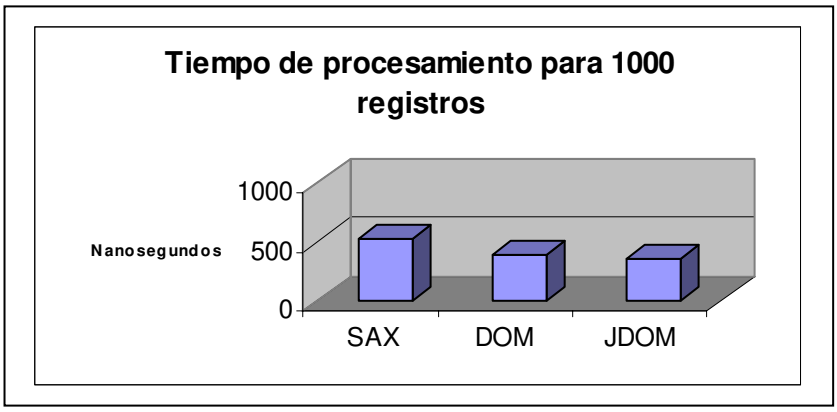

Figura 5.14. Tiempo de procesamiento del documento XML para mil registros

5.5. Análisis.

En las aplicaciones de búsqueda se desarrolla la programación de los parser SAX, DOM y JDOM se compara su desempeño resolviendo el mismo problema de tal manera que proporcionen idéntica solución, la metodología de búsqueda se mantuvo simple para analizar las capacidades de ejecución en este contexto las evaluaciones de rendimiento se presentan como micro-comparaciones.

Para interactuar con las diferentes implementaciones de los analizadores SAX, los constructores de documentos DOM y JDOM comparten una estructura común que permite comparar sus respectivos rendimientos por cada ejecución se usa una factoría para crear un analizador que a su vez son usados para procesar un documento fuente XML, en los métodos *SAXParserBusca.java*, *ProcesarBuscaDom.java, ProcesarBuscaJdom.java*, después de su ejecución se produce el tiempo medio que tarda en procesar el documento.

- Como se puede observar en las Figuras 5.12, 5.13 y 5.14 el tiempo consumido para procesar el documento XML tiene un incremento proporcional al tamaño del documento XML, en esta evaluación se midió el tiempo para procesar documentos que describen configuraciones de datos organizados en nodos de un Documento

XML cuya cantidad de registros son de tres, seis, nueve, diez y ocho, cien, trescientos treinta, quinientos y mil registros.

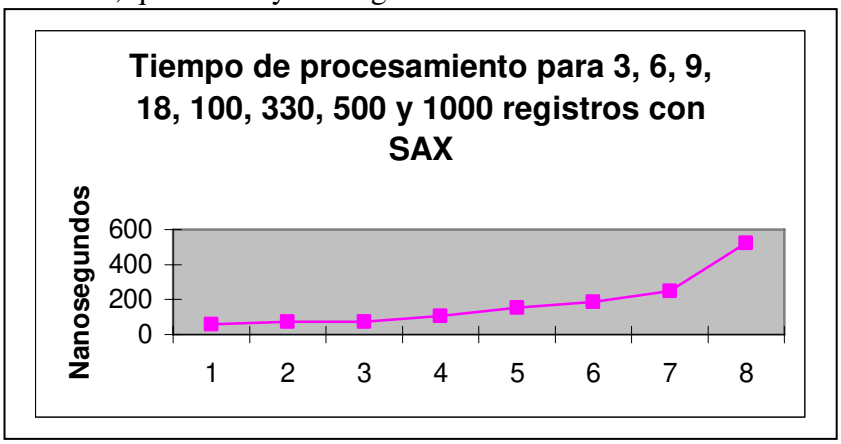

Figura 5.15. Tiempo de procesamiento con el parser SAX

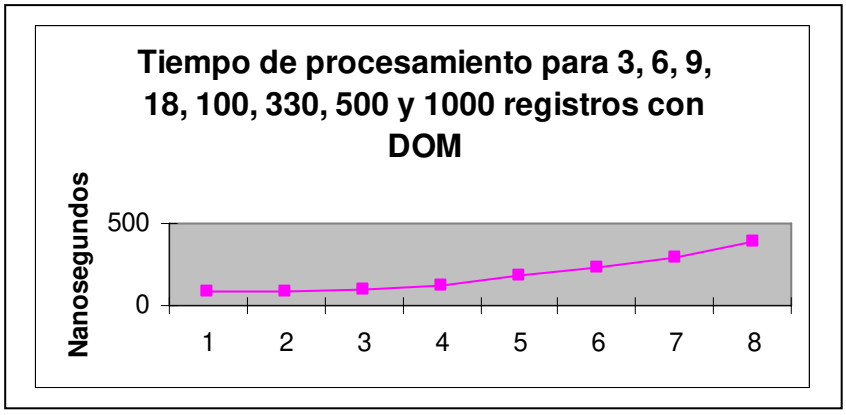

Figura 5.16. Tiempo de procesamiento con el parser DOM

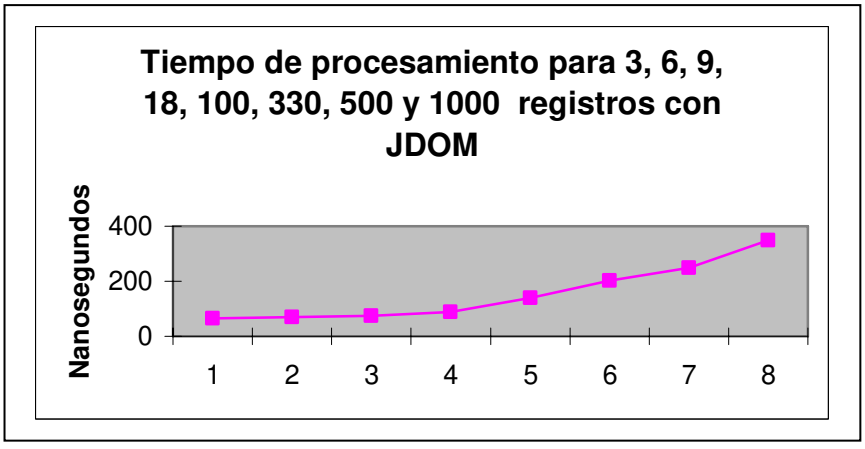

Figura 5.17. Tiempo de procesamiento con el parser JDOM

- El programa de ejemplo JDOM usa la clase *SAXBuilder* para analizar el documento fuente XML en memoria pasa a través del documento JDOM resultante y saca el resultado en formato de texto adicionalmente esta evaluación compara SAX con DOM, usando la clase *DefaultHandler* de SAX personalizado para registrar "oyentes" de eventos que capturan los eventos disparados por el analizador SAX.

- Para el procesamiento con el parser DOM el acceso a un elemento de un documento de entrada por nombre se realizó con el método *getElementsByTagName* () ya que todos los elementos con un nombre de etiqueta dado se retornan en el orden en que sean encontrados desde el nodo documento o elemento actual.

- Por cada llamada al método *getElementsByTagName( )* se tiene que atravesar totalmente el subárbol y requiere menos conocimiento de la estructura del árbol ya que se hace una búsqueda exhaustiva la búsqueda de diferentes nombres de elementos podría requerir atravesar el árbol varias veces por medio del análisis por medio del análisis de las evaluaciones y la cantidad de tiempo de procesamiento consumido por los parser JDOM es ligeramente más rápido que DOM (aproximadamente un 10%) usando la misma implementación.

- En la Figura 5.18 se observan los resultados obtenidos en cuanto a la cantidad de recursos de código que cada parser necesita para el proceso de la aplicación usando los parsers que se importan desde el *Xerces*, para la aplicación de búsqueda con JDOM se puede usar el modelo de la programación orientada a objetos, simplificando el acceso a los datos contenidos en el documento XML.

5.6. Artículo Tecnológico.

Como resultado de la investigación realizada se desarrollo la elaboración del artículo "Desarrollo de aplicaciones Web con XML y Java" [2], el cual se presento en la ponencia con el mismo nombre en el cuarto congreso internacional en innovación y desarrollo tecnológico organizado por la organización IEEE y AMIME sección Morelos.

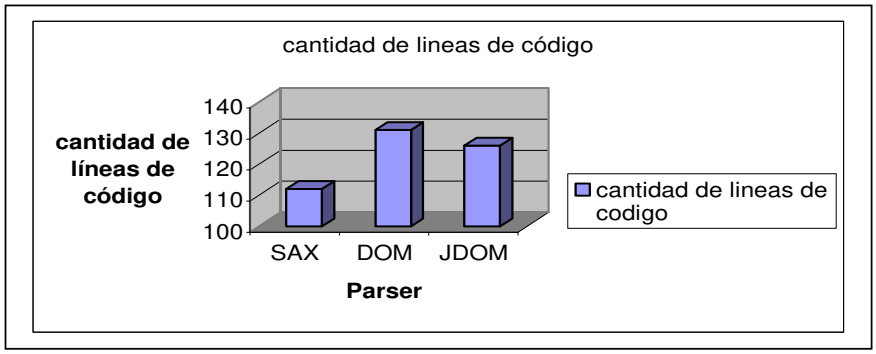

Figura 5.18. Cantidad de líneas de código de los parser SAX, DOM y JDOM

- Se programaron los recursos relacionados al manejo de eventos de SAX para procesar XML el analizador SAX ejecuta los eventos de inicio y final de los elementos de la aplicación de búsqueda al momento de procesar los datos del archivo XML ya que el analizador reporta los eventos mientras visita las diferentes partes del documento, no consume memoria por que no tiene que construir una estructura interna (de árbol como es el caso de los parsers DOM y JDOM) debido a estas características se reduce los recursos de sistema necesarios.

- El analizador DOM construye una estructura interna por la que una aplicación puede visitar cada elemento en los nodos del árbol mediante la evaluación del código se observa que si el documento XML es más grande también se incrementará la memoria utilizada por lo cual para aplicaciones grandes y complejas no es recomendable el uso de DOM se ha explicado el uso de estos parsers para el proceso de datos presentando una implementación Java.

6. CONCLUSIONES Y TRABAJOS FUTUROS.

Como se puede observar a lo largo de esta investigación el uso de JDOM simplifica y hace más eficiente el procesamiento de grandes modelos de documentos en memoria el aspecto más importante que lo hace posible, es que no usa factorías de software (promovidas por grupos de trabajo relacionados con la empresa Microsoft) ni otros modelos avanzados de programación como el MDA (Model Driven Architecture) propuesta por el OMG (Object Management Group).

6.1. Conclusiones.

Las factorías se usan para crear un objeto mediante el cual se puede realizar el procesamiento del documento XML lo que hace que la aplicación sea más lenta por la cantidad de recursos de memoria que se requieren y por ende se incrementa el tiempo de procesamiento.

JDOM y DOM crean documentos XML lo cual no se permite en SAX por ser de sólo lectura, además el procesamiento con SAX es eficiente porque no carga el documento en memoria sino que en tiempo real va accediendo a los elementos del documento, pero una desventaja es que no permite acceder a elementos del documento XML previamente procesados por medio de los analizadores de evento por lo tanto si se requiere recuperar un elemento anterior se debe ejecutar nuevamente la aplicación.

Una característica común en DOM y JDOM es que ambos cargan el documento a memoria a través de un estructura de datos árbol con lo cual se puede acceder a cada uno de los nodos en cualquier momento, sin embargo JDOM hace más eficiente la generación de código ya que se basa en el paradigma orientado a objetos principalmente la herencia y el polimorfismo.

Lo anterior se demuestra al momento que en el tiempo de procesamiento de documentos JDOM se reduce en aproximadamente un 10% del tiempo requerido por el parser DOM en cuestión a la complejidad de código SAX usa menos líneas de código pero es más complejo, con respecto a DOM como hace uso de la factoría la cantidad de líneas de código se incrementa pero disminuye la complejidad, finalmente en JDOM se reduce en gran medida la complejidad y también el número de líneas en aproximadamente un 10% con base a lo anterior JDOM es más recomendable para procesar documentos XML.

6.2. Trabajos Futuros.

El modelado de la información requiere codificar la información para diversas necesidades entre los candidatos importantes destaca XML de tal manera que se pueden utilizar servicios Web para integrar una serie de aplicaciones heterogéneas presentando los datos en diferentes formatos en el procesamiento de documentos XML.

El uso de software personalizado en muchas empresas ha dado lugar a un gran número de datos y lógica empresarial útiles pero aislados debido a la diversidad de circunstancias en las que se produce su desarrollo y al carácter siempre evolutivo de la tecnología, la creación de un ensamblado funcional de Java y XML a partir de aplicaciones desarrolladas con el parser JDOM es una tarea importante.

Una investigación a futuro considerando los resultados de la investigación documentada en esta tesis sería realizar un estudio de transacciones distribuidas en la Web utilizando JDOM y XML para el análisis y desarrollo de soluciones a problemas asociados al procesamiento de aplicaciones distribuidas y posteriormente implementar el uso de la plataforma .NET.

7. BIBLIOGRAFÍA.

1 Carlson David, *Modeling XML Applications with UML*, Addison Wesley, United States of America, Addison Wesley, Abril 2001.

2 Dorantes Hernández L. M., García Ramírez M.T., García Sánchez C.F. *Desarrollo de aplicaciones Web con XML y Java*, octubre 2006.

3 Eckel Bruce, *Piensa en Java*, Prentice Hall, España, 2001.

4 El sitio de JDOM http://www.jdom.org

5 Español MSDN http://msdn2.microsoft.com, 2006 Microsoft Corporation.

6 Extensible Markup Languaje (XML), W3C, http://www.w3.org/XML, última actualización: 2006/09/11.

7 Gutiérrez Abraham, Martínez Raúl, *XML a través de ejemplos*, Alfaomega Ra-Ma, Madrid España, 2001.

8 Java en castellano http://www.programacion.com/java , última actualización: 2006.

9 Marchal Benoit, *XML con Ejemplos*, Pearson Educación, México, 2001.

10 Maruyama Hiroshi, Tamura Kent, Uramoto Naohiko, *Sitios Web con XML y Java*, Pearson Educación, S.A., España, 2001.

11 McLaughlin Brett, *Java & XML, 2nd Edition: Solutions to Real -World Problems* Oreilly, United States of America, Septiembre 2001.

12 McLaughlin Brett, *JAVA and XML*, O' REILLY & Associates, Inc., United States of America, 2000.

13 Olivilla Ramón, *Diseño y Programación de Aplicaciones Web*, Infobook´s Ediciones, Barcelona España, 2003.

14 PTC, Centro de recursos www.arbortext.com, 2006, Parametric Technology Corporation.

15 Rusty Harold Elliotte, *Processing XML with Java A Guide to SAX, DOM, JDOM, JAXP, and TrAX*, Addison-Wesley, United States of America, 2003.

16 Sitios Web relacionados para editores XML www.usa.iconis.com/xml/xmlspy.html.

17 Vervet Logic, home of XML Pro www.vervet.com, 1997-2005 Vervet Logic.

8. GLOSARIO.

Analizador	Biblioteca de software a cargo de leer y escribir documentos XML.
API	Interfaz de Programación de Aplicaciones (Application Programming Interface)
CORBA	Arquitectura Común de Intermediarios en Peticiones a Objetos (Common Object Request Broker Architecture), es un middleware orientado a objetos.
CSS	Hojas de Estilo en Cascada (Cascading Style Sheets), es un lenguaje de hojas de estilo desarrollado originalmente para HTML.
DTD	Definición de Tipos de Documento (Document Type Definition), es el modelo de un documento XML.
EDI	Intercambio Electrónico de Datos (Electronic Data Interchange), es una tecnología utilizada para intercambiar documentos electrónicos en forma electrónica como facturas y pedidos.
Etiqueta	Elemento de marcado usadas en documentos XML
HTML	Lenguaje de Marcado de Hipertexto (HyperText Markup Language) es un formato de las páginas Web.
HTTP	Protocolo de Transferencia de Hipertexto (HyperText Transfer Protocol) es un protocolo utilizado por servidores y exploradores Web.
ISO	Organización Internacional de Estándares (International Organization Standarization) es una organización oficial que publica estándares.
Middleware	Tecnología que simplifica la construcción de aplicaciones distribuidas.
SGML	Lenguaje Estándar de Marcado Generalizado (Standard Generalized Markup Language) es el ancestro de HTML y de XML.
URL	Localizador Uniforme de Recursos (Uniform Resource Locator) es la dirección de un recurso en el servicio Web.

W3C	Consorcio de World Wide Web es la organización a cargo de la estandarización del servicio Web.
XML	Lenguaje Extensible de Marcado (extensible markup language) es un nuevo lenguaje de marcado publicado por el W3C para resolver las limitaciones de HTML.
X-Schema	Es un nombre genérico para el reemplazo propuesto de la DTD.
XSL	Lenguaje de Hojas de Estilo Extensible (Extensible Stylesheet Language) es un lenguaje de hojas de estilo desarrollado específicamente para XML.
Factoría de Software	Herramientas extensibles, son procesos y contenido para automatizar el desarrollo y mantenimiento mediante la adaptación, ensamblaje y configuración de componentes basados en frameworks (arquitecturas de software).

9. ANEXOS.

Instalación de Xerces.

Uno de los parsers de mayor uso es Xerces desarrollado por la fundación *Apache*, Xerces está disponible en los lenguajes Java y C++ Xerces ofrece un ambiente de trabajo completo para la construcción de componentes y configuraciones de los procesadores es modular y facilita el desarrollo, también provee una implementación para los parser SAX, DOM y JDOM para el procesamiento de documentos XML, en la Figura 1.a se muestra la pantalla principal en la cual se puede obtener Xerces y los procesadores SAX, DOM Y JDOM.

• Obtener la versión para Windows de la página Web cuya dirección es:
http://xml.apache.org/dist/xerces-j/

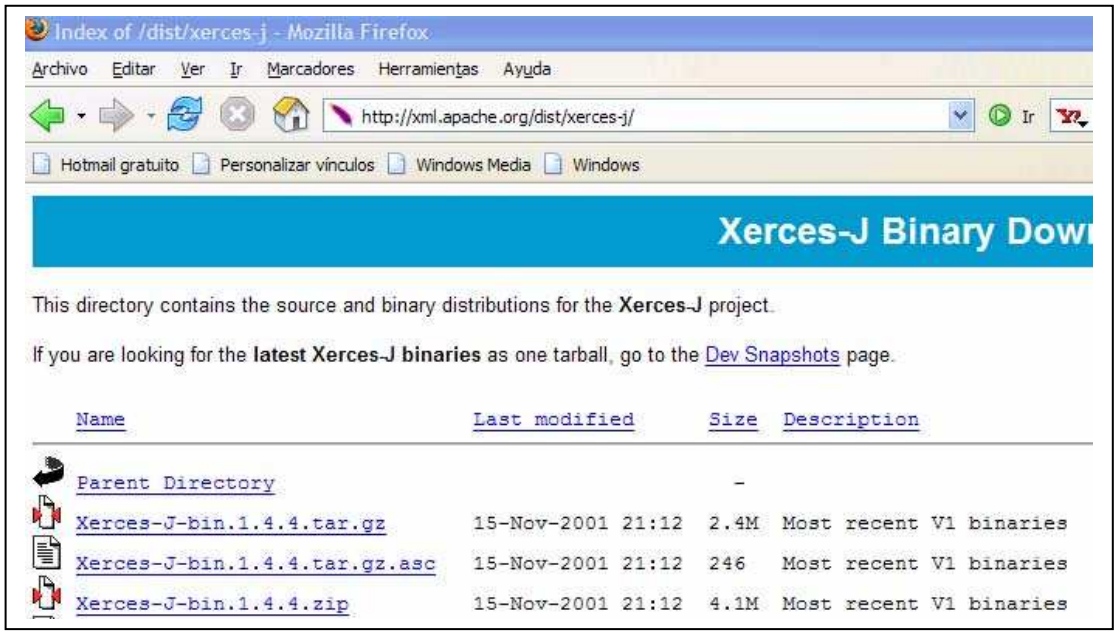

Figura 1.a. Xerces para Windows

• La documentación va integrada en la propia versión

• Descomprimir el archivo cuya extensión es "*zip*" obtenido en su directorio definitivo

Figura 1.b. Descomprimir el archivo Xerces-J-bin.1.4.4.zip

• Actualizar la variable de entorno CLASSPATH en Panel de Control, Sistema, Entorno – Añadir la ruta x:\path_de_xerces\xerces.jar

• Si se va a ejecutar alguno de los programas ejemplo que vienen con Xerces, añadir también la ruta

– x:\path_de_xerces\xercesSamples.jar

• Xerces incluye una implementación de las API de procesamiento XML, DOM, SAX y JDOM.

Programas de Aplicación.

Creación de documentos XML.

Programa *HolaJDOM.java*:

HolaJDOM.java *

```
1  import org.jdom.*;
2  import org.jdom.output.XMLOutputter;
3  import java.io.IOException;
4  public class HolaJDOM {
5    public static void main(String[] args) {
6      Element raiz = new Element("Hola_saludo");
7      raiz.setText("Hola");
8      Document doc = new Document(raiz);
9      // serializarlo en el System.out
10     try {
11       XMLOutputter serializer = new XMLOutputter();
12       serializer.output(doc, System.out);
13     }
14     catch (IOException e) {
15       System.err.println(e);
16     }
17   }
18 }
```

Programa *HolaDOM.java* utilizando el parser DOM.

HolaDOM.java *

```
3  import javax.xml.transform.*;
4  import javax.xml.transform.dom.DOMSource;
5  import javax.xml.transform.stream.StreamResult;
6  public class HolaDOM {
7    public static void main(String[] args) {
8      try {
9          DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
10         DocumentBuilder builder = factory.newDocumentBuilder();
11         DOMImplementation impl = builder.getDOMImplementation();
12         Document doc = impl.createDocument(null, "Hola_saludo", null);
13         // Crear el documento
14         Element raiz = doc.getDocumentElement();
15         Text text = doc.createTextNode("Hola");
16         raiz.appendChild(text);
17       // Serializar el documento en System.out
18       TransformerFactory xformFactory
19        = TransformerFactory.newInstance();
20       Transformer idTransform = xformFactory.newTransformer();
21       Source input = new DOMSource(doc);
22       Result output = new StreamResult(System.out);
23       idTransform.transform(input, output);
24       }
25     catch (FactoryConfigurationError e) {
26       System.out.println("No puedo ubicar la clase factory JAXP ");
```

Programa *FibonacciDOM.java* usando el parser DOM.

FibonacciDOM.java *

```
1  import org.w3c.dom.*;
2  import javax.xml.parsers.*;
3  import javax.xml.transform.*;
4  import javax.xml.transform.dom.DOMSource;
5  import javax.xml.transform.stream.StreamResult;
6  import java.math.BigInteger;
7  public class FibonacciDOM {
8    public static void main(String[] args) {
9      try {
10       // Encontrar la implementación
11       DocumentBuilderFactory factory
12         = DocumentBuilderFactory.newInstance();
13       factory.setNamespaceAware(true);
14       DocumentBuilder builder = factory.newDocumentBuilder();
15       DOMImplementation impl = builder.getDOMImplementation();
16       // Crear el documento
17       Document doc = impl.createDocument(null,
18         "Fibonacci_Numbers", null);
19       // Llenar el documento
20       BigInteger low  = BigInteger.ONE;
21       BigInteger high = BigInteger.ONE;
22       Element root = doc.getDocumentElement();
23       for (int i = 0; i < 10; i++) {
24         Element number = doc.createElement("fibonacci");
25         Text text = doc.createTextNode(low.toString());
26         number.appendChild(text);
27         root.appendChild(number);
28         BigInteger temp = high;
29         high = high.add(low);
30         low = temp;
31       }
32       // Serializar el documento en System.out
33       TransformerFactory xformFactory
34         = TransformerFactory.newInstance();
35       Transformer idTransform = xformFactory.newTransformer();
36       Source input = new DOMSource(doc);
37       Result output = new StreamResult(System.out);
38       idTransform.transform(input, output);
39     }
40     catch (FactoryConfigurationError e) {
41       System.out.println("No puedo ubicar la clase factory JAXP ");
42     }
43     catch (ParserConfigurationException e) {
44       System.out.println(
45         "No puedo ubicar la clase DocumentBuilder JAXP "
46       );
47     }
48     catch (DOMException e) {
49       System.err.println(e);
50     }
51     catch (TransformerConfigurationException e) {
52       System.err.println(e);
```

Programa *FibonacciJDOM.java* usando el parser JDOM.

FibonacciJDOM.java *

```
1  import org.jdom.*;
2  import org.jdom.output.XMLOutputter;
3  import java.math.BigInteger;
4  import java.io.IOException;
5  public class FibonacciJDOM {
6    public static void main(String[] args) {
7      Element root = new Element("Numeros_Fibonacci");
8      BigInteger low  = BigInteger.ONE;
9      BigInteger high = BigInteger.ONE;
10     for (int i = 1; i <= 5; i++) {
11       Element fibonacci = new Element("fibonacci");
12       fibonacci.setAttribute("index", String.valueOf(i));
13       fibonacci.setText(low.toString());
14       root.addContent(fibonacci);
15       BigInteger temp = high;
16       high = high.add(low);
17       low = temp;
18     }
19     Document doc = new Document(root);
20     // serializarlo en el System.out
21     try {
22       XMLOutputter serializer = new XMLOutputter();
23       serializer.output(doc, System.out);
24     }
25     catch (IOException e) {
26       System.err.println(e);
27     }
28   }
29 }
```

Programa *SAXFibonacci.java* usando el parser SAX.

SAXFibonacci.java *

```
5  import javax.xml.parsers.ParserConfigurationException;
6  import javax.xml.parsers.SAXParser;
7  import javax.xml.parsers.SAXParserFactory;
8  import org.xml.sax.Attributes;
9  import org.xml.sax.SAXException;
10 import org.xml.sax.helpers.DefaultHandler;
11 //--------------------------------------------------
12 public class SAXFibonacci extends DefaultHandler {
13     List myEmpls;
14     private String tempVal;
15     private Fibonaccii tempEmp;
16     public SAXFibonacci(){
17         myEmpls = new ArrayList();
18     }
19     public void runEjemplo() {
20         procesarDocumento();
21         imprimirDato();
22     }
23     private void procesarDocumento() {
24         //tomar una factory
25         SAXParserFactory spf = SAXParserFactory.newInstance();
26         try {
27             // tomar una nueva instancia del parser
28             SAXParser sp = spf.newSAXParser();
29             //procesar el archivo xml
30             sp.parse("fibonacci.xml", this);
```

SAXFibonacci.java *

```
31      }catch(SAXException se) {
32          se.printStackTrace();
33      }catch(ParserConfigurationException pce) {
34          pce.printStackTrace();
35      }catch (IOException ie) {
36          ie.printStackTrace();
37      }
38  }
39   Iterar a través de la lista e imprimir
40   el contenido
41  private void imprimirDato(){
42      Iterator it = myEmpls.iterator();
43      while(it.hasNext()) {
44          System.out.println(it.next().toString());
45      }
46  }
47  // Manejador de Eventos
48  public void startElement(String uri, String localName, String qName, Attribute
49      tempVal = null;
50      if(qName.equalsIgnoreCase("Fibonaccii")) {
51          //crear una nueva instancia de empleado
52          tempEmp = new Fibonaccii();
53      }
54  }
55  public void characters(char[] ch, int start, int length) throws SAXException {
56      tempVal = new String(ch,start,length);
```

SAXFibonacci.java *

```
57  }
58  public void endElement(String uri, String localName, String qName) throws SAXE
59
60      if (qName.equalsIgnoreCase("Fibonaccii")) {
61
62          myEmpls.add(tempEmp);
63          tempEmp.ponerClave(Integer.parseInt(tempVal));
64      }
65  }
66
67  public static void main(String[] args){
68      SAXFibonacci spe = new SAXFibonacci();
69      spe.runEjemplo();
70  }
```

Programa *FibonacciJDOMFormato.java* con salida formateada.

FibonacciJDOMFormato.java

```
1  import org.jdom.*;
2  import org.jdom.output.XMLOutputter;
3  import org.jdom.output.Format;
4  import java.math.BigInteger;
5  import java.io.IOException;
6
7  public class FibonacciJDOMFormato {
8  public static void main(String[] args) {
9
10 Element root = new Element("numeros_fibonacci");
11 BigInteger low = BigInteger.ONE;
12 BigInteger high = BigInteger.ONE;
13 for (int i = 1; i <= 5; i++) {
14 Element fibonacci = new Element("fibonacci");
15 fibonacci.setAttribute("indice", String.valueOf(i));
16 fibonacci.setText(low.toString());
17 root.addContent(fibonacci);
18 BigInteger temp = high;
19 high = high.add(low);
20 low = temp;
21 }
22 Document doc = new Document(root);
23 try {
24 XMLOutputter serializer = new XMLOutputter(Format.getPrettyFormat());
25 serializer.output(doc, System.out);
26 }
```

Lectura de documentos XML.

Programa *SAXParserLista.java* utilizando el parser SAX.

SAXParserLista.java *

```
1  import java.io.IOException;
2  import java.util.ArrayList;
3  import java.util.Iterator;
4  import java.util.List;
5  import javax.xml.parsers.ParserConfigurationException;
6  import javax.xml.parsers.SAXParser;
7  import javax.xml.parsers.SAXParserFactory;
8  import org.xml.sax.Attributes;
9  import org.xml.sax.SAXException;
10 import org.xml.sax.helpers.DefaultHandler;
11 //------------------------------------------------
12 public class SAXParserLista extends DefaultHandler {
13     List myEmpls;
14     private String tempVal;
15     private Empleadoo tempEmp;
16     public SAXParserLista(){
17         myEmpls = new ArrayList();
18     }
19     public void runEjemplo() {
20         procesarDocumento();
21         imprimirDato();
22     }
23     private void procesarDocumento() {
24         //tomar una factory
25         SAXParserFactory spf = SAXParserFactory.newInstance();
26         try {
```

Ejemplo2.java | MayorMenor.java | **SAXParserLista.java** *

```
27      }
28      public void runEjemplo() {
29          procesarDocumento();
30          imprimirDato();
31      }
32      private void procesarDocumento() {
33          //tomar una factory
34          SAXParserFactory spf = SAXParserFactory.newInstance();
35          try {
36              // tomar una nueva instancia del parser
37              SAXParser sp = spf.newSAXParser();
38              //ejecutar el archivo y también registrar esta clase para llamadas
39              sp.parse("empleadoos.xml", this);
40              }catch(SAXException se) {
41              se.printStackTrace();
42          }catch(ParserConfigurationException pce) {
43              pce.printStackTrace();
44          }catch (IOException ie) {
45              ie.printStackTrace();
46          }
47      }
48       /* Iterar a través de la lista e imprimir
49        *  el contenido */
50      private void imprimirDato(){
51          System.out.println("Numero de Empleados '" + myEmpls.size() + "'.");
52          Iterator it = myEmpls.iterator();
53          while(it.hasNext()) {
54              System.out.println(it.next().toString());
55          }
56      }
```

jemplo2.java | MayorMenor.java | **SAXParserLista.java** *

```
57      // Manejador de Eventos
58      public void startElement(String uri, String localName, String qName, Attributes a
59          tempVal = "";
60          if(qName.equalsIgnoreCase("Empleadoo")) {
61              //crear una nueva instancia de empleado
62              tempEmp = new Empleadoo();
63              tempEmp.ponerTipo(attributes.getValue("tipo"));
64          }
65      }
66      public void characters(char[] ch, int start, int length) throws SAXException {
67          tempVal = new String(ch,start,length);
68      }
69      public void endElement(String uri, String localName, String qName) throws SAXExce
70          if(qName.equalsIgnoreCase("Empleadoo")) {
71              // añadirlo a la lista
72              myEmpls.add(tempEmp);
73          }else if (qName.equalsIgnoreCase("Nombre")) {
74              tempEmp.ponerNombre(tempVal);
75          }else if (qName.equalsIgnoreCase("Clave")) {
76              tempEmp.ponerClave(Integer.parseInt(tempVal));
77          }else if (qName.equalsIgnoreCase("Edad")) {
78              tempEmp.ponerEdad(Integer.parseInt(tempVal));
79          }
80      }
81      public static void main(String[] args){
82          SAXParserLista spe = new SAXParserLista();
83          spe.runEjemplo();
84      }
85  }
```

Lectura de documentos XML con DOM.

Programa *DOMParserLista.java* utilizando el parser DOM.

DOMParserLista.java *

```
1  import java.io.IOException;
2  import java.util.ArrayList;
3  import java.util.Iterator;
4  import java.util.List;
5  import javax.xml.parsers.DocumentBuilder;
6  import javax.xml.parsers.DocumentBuilderFactory;
7  import javax.xml.parsers.ParserConfigurationException;
8  import org.w3c.dom.Document;
9  import org.w3c.dom.Element;
10 import org.w3c.dom.NodeList;
11 import org.xml.sax.SAXException;
12
13 public class DOMParserLista {
14     //datos de tipo No genericos
15     List myEmpls;
16     Document dom;
17     public DOMParserLista (){
18     //crea una lista para contener los objetos de los empleados
19         myEmpls = new ArrayList();
20     }
21     public void runEjemplo() {
22         //procesar el archivo xml y tomar el objeto dom
23         procesarArchXml();
24         //tomar cada elemento empleado y crear un objeto Empleado
25         procesarDocumento();
26         //Iterar a través de la lista e imprimir los datos
27         imprimirDato();
28     }
29     private void procesarArchXml(){
30     //declaración de un objeto tipo factory
```

DOMParserLista.java *

```
29     private void procesarArchXml(){
30     //declaración de un objeto tipo factory
31 DocumentBuilderFactory dbf =  DocumentBuilderFactory.newInstance();
32         try {
33     //Uso de factory para obtener una instancia del constructor    document
34     DocumentBuilder db = dbf.newDocumentBuilder();
35     //procesar usando el constructor para tomar la representación DOM del //archivo
36     dom = db.parse("empleadoos.xml");
37
38     }catch(ParserConfigurationException pce) {
39             pce.printStackTrace();
40     }catch(SAXException se) {
41             se.printStackTrace();
42     }catch(IOException ioe) {
43             ioe.printStackTrace();
44         }
45     }
46 private void procesarDocumento(){
47         //tomar el elemento raiz
48         Element docEle = dom.getDocumentElement();
49         //usar una nodelist de los elementos <employee>
50         NodeList nl = docEle.getElementsByTagName("Empleadoo");
51         if(nl != null && nl.getLength() > 0) {
52             for(int i = 0 ; i < nl.getLength();i++) {
53                 //tomar el elemento empleado
54                 Element el = (Element)nl.item(i);
55                 //tomar el objeto Employee
56                 Empleadoo e = tomarEmpleadoo(el);
57                 //añadirlo a la lista
58                 myEmpls.add(e);
```

DOMParserLista.java *

```
62  /* Se toma un elemento empleadoo y se leen los valores contenidos  en el
63   * crea un objeto Empleadoo y lo retorna
64   * @parametro empEl
65   * @return
66   */
67  private Empleadoo tomarEmpleadoo(Element empEl) {
68   //para cada elemento <empleadoo> se toman los valores tipo
69   //texto o entero de nombre, clave, edad y tipo de empleado
70       String nombre = getTextValue(empEl,"Nombre");
71       int clave = getIntValue(empEl,"Clave");
72       int edad = getIntValue(empEl,"Edad");
73       String tipo = empEl.getAttribute("tipo");
74   //Crea un nuevo Empleado con los valores leidos de los nodos de xml
75       Empleadoo e = new Empleadoo(nombre,clave,edad,tipo);
76       return e;
77  }
78  /* Se toma un elemento xml y la etiqueta nombre, revisando la etiqueta y tomando
79   * el texto del contenido
80   * por ejemplo para <empleadoo><nombre>Salvador</nombre></empleadoo> xml ubica si
81   * el Elemento apunta al nodo empleadoo y el nombre de la etiqueta es nombre, se ret
82   * @parametro ele
83   * @parametro tagName
84   * @retorna un valor de tipo String*/
85  private String getTextValue(Element ele, String tagName) {
86       String textVal = null;
87       NodeList nl = ele.getElementsByTagName(tagName);
88       if(nl != null && nl.getLength() > 0) {
89           Element el = (Element)nl.item(0);
90           textVal = el.getFirstChild().getNodeValue();
91       }
92       return textVal;
```

DOMParserLista.java *

```
100 private int getIntValue(Element ele, String tagName) {
101     return Integer.parseInt(getTextValue(ele,tagName));
102 }
103 /**
104  * Iterar a través de la lista e imprimir el contenido a la consola
105  */
106 private void imprimirDato(){
107  System.out.println("Numero de Empleadoos '" + myEmpls.size() + "'.");
108     Iterator it = myEmpls.iterator();
109     while(it.hasNext()) {
110         System.out.println(it.next().toString());
111     }
112 }
113
114 public static void main(String[] args){
115     //crear una instancia
116     DOMParserLista dpe = new DOMParserLista ();
117     //llamado al metodo
118     dpe.runEjemplo();
119 }
```

Programa *ProcesarEjemploJdom.java*

ProcesarEjemploJdom.java *

```
1  import java.io.*;
2  import java.util.*;
3  import org.jdom.*;
4  import org.jdom.input.*;
5  import org.jdom.output.*;
6  import org.jdom.Attribute;
7  import org.jdom.Comment;
8  import org.jdom.Document;
9  import org.jdom.Element;
10 import org.jdom.JDOMException;
11 import org.jdom.input.SAXBuilder;
12 import org.jdom.output.XMLOutputter;
13 public class ProcesarEjemploJdom {
14     //datos de tipo No genericos
15     List myEmpls;
16     Document doc;
17     public ProcesarEjemploJdom(){
18         //crea una lista para contener los objetos de los empleados
19         myEmpls = new ArrayList();
20     }
21     public void runEjemplo() {
22         //procesar el archivo xml y tomar el objeto jdom
23         procesarArchXml();
24         //tomar cada elemento empleado y crear un objeto Empleado
25         procesarDocumento();
26         //Iterar a través de la lista e imprimir los datos
```

ProcesarEjemploJdom.java *

```
27         imprimirDato();
28     }
29     private void procesarArchXml(){
30         try {
31         SAXBuilder constructor=new SAXBuilder(false);
32             //usar el parser Xerces
33             doc=constructor.build("empleadoos.xml");
34         }catch(Exception ioe) {
35             ioe.printStackTrace();
36         }
37     }
38     private void procesarDocumento(){
39         Element trabElement = doc.getRootElement();
40         List nl=trabElement.getChildren("Empleadoo");
41         System.out.println("Formada por:"+nl.size()+" empleados");
42         Iterator i = nl.iterator();
43         while (i.hasNext()){
44             //tomar el elemento empleado
45             Element el= (Element)i.next();
46             //tomar el objeto Empleadoo
47             Empleadoo e = tomarEmpleadoo(el);
48             //añadirlo a la lista
49             myEmpls.add(e);
50         }
51     }
```

ProcesarEjemploJdom.java *

```
53      /* Se toma un elemento empleadoo y se leen los valores *contenidos    en el,
54       * crea un objeto Empleadoo y lo retorna
55       * parametro empEl
56       * retorna un objeto */
57      private Empleadoo tomarEmpleadoo(Element empEl) {
58          String nombre = getTextValue(empEl,"Nombre");
59          int    clave  = getIntValue(empEl,"Clave");
60          int    edad   = getIntValue(empEl,"Edad");
61          String tipo   = empEl.getAttributeValue("tipo");
62          //Crea un nuevo objeto Empleadoo con los valores leidos de los nodos de xml
63          Empleadoo ele = new Empleadoo(nombre,clave,edad,tipo);
64          return ele;
65          }
66          // Se toma un elemento xml y la etiqueta nombre, revisando
67          // la etiqueta y tomando el texto del contenido
68          // por ejemplo para <empleadoo> <nombre> Salvador </nombre> </empleadoo>
69          // xml ubica si el Elemento apunta al nodo empleadoo y el nombre de la
70          // etiqueta es nombre, se retornará Salvador
71          // parametros: ele, tagName
72          // retorna un valor de tipo String
73      private String getTextValue(Element ele, String tagName) {
74          String textVal=null;
75          // Acceso a un elemento hijo
76          Element el = ele.getChild(tagName);
77          // Mostrar una posible falla
78          if(el != null) {
79                  textVal=el.getText();
80          } else {
81              System.out.println("Problema.  No se encuentra un elemento ");
82          }
83          return textVal;
```

ProcesarEjemploJdom.java *

```
85      /**
86       * llama a getTextValue y retorna un valor int
87       * parametros: ele, tagName
88       * retorna un dato tipo entero
89       */
90      private int getIntValue(Element ele, String tagName) {
91          return Integer.parseInt(getTextValue(ele,tagName));
92      }
93      /**
94   * Iterar a través de la lista e imprimir el contenido a la consola*/
95      private void imprimirDato(){
96   System.out.println("Numero de Empleadoos '" + myEmpls.size() + "'.");
97          Iterator it = myEmpls.iterator();
98          while(it.hasNext()) {
99              System.out.println(it.next().toString());
100         }
101     }
102     public static void main(String[] args){
103         //crear una instancia
104         ProcesarEjemploJdom dpe = new ProcesarEjemploJdom();
105         //llamado al metodo
106         dpe.runEjemplo();
107     }
108 }
```

Búsqueda en Documentos XML.

Programa *SAXParserBusca.java* utilizando el parser SAX.

SAXParserBusca.java *

```
1  import java.io.*;
2  import java.io.IOException;
3  import java.util.ArrayList;
4  import java.util.Iterator;
5  import java.util.List;
6  import javax.xml.parsers.ParserConfigurationException;
7  import javax.xml.parsers.SAXParser;
8  import javax.xml.parsers.SAXParserFactory;
9  import org.xml.sax.Attributes;
10 import org.xml.sax.SAXException;
11 import org.xml.sax.helpers.DefaultHandler;
12 public class SAXParserBusca extends DefaultHandler {
13     static int opcion;
14     List myEmpls;
15     static double promedio;    // donde almacenamos el tiempo promedio del algoritmo
16     private String tempVal;
17     private Empleadoo tempEmp;
18     public SAXParserBusca(){
19         myEmpls = new ArrayList();
20     }
21     public void runEjemplo(String dato) {
22             long start,end;     // donde almacenamos el valor inicial y final del ti
23             int numIter = 1000;     // numero de mediciones
24             double suma = 0;    // iniciamos el acumulador
25             for(int i=0;i<numIter;i++)
26             {
27             start = System.nanoTime();
28             procesarDocumento(dato);
29             end = System.nanoTime();
30             suma += end - start;
31             }
```

ca.java *

```
        //System.out.println(" \n\npromedio de tiempo:"+promedio);

private void procesarDocumento(String dato) {
    //tomar una factory
    SAXParserFactory spf = SAXParserFactory.newInstance();
    try {
        // tomar una nueva instancia del parser
        SAXParser sp = spf.newSAXParser();
        //ejecutar el archivo y también registrar esta clase para llamadas
        sp.parse("empleadoos.xml", this);
    }catch(SAXException se) {
        se.printStackTrace();
    }catch(ParserConfigurationException pce) {
        pce.printStackTrace();
    }catch (IOException ie) {
        ie.printStackTrace();
    }
}
// Manejador de Eventos
public void startElement(String uri, String localName, String qName, Attributes attribu
    tempVal = "";
    if(qName.equalsIgnoreCase("Empleadoo")) {
        //crear una nueva instancia de empleado
        tempEmp = new Empleadoo();
        tempEmp.ponerTipo(attributes.getValue("tipo"));
    }
}
public void characters(char[] ch, int start, int length) throws SAXException {
    tempVal = new String(ch,start,length);
}
public void endElement(String uri, String localName, String qName) throws SAXException
```

SAXParserBusca.java *

```
33                  //System.out.println(" \n\npromedio de tiempo:"+promedio);
34  }
35      private void procesarDocumento(String dato) {
36          //tomar una factory
37          SAXParserFactory spf = SAXParserFactory.newInstance();
38          try {
39              // tomar una nueva instancia del parser
40              SAXParser sp = spf.newSAXParser();
41              //ejecutar el archivo y también registrar esta clase para llamadas
42              sp.parse("empleadoos.xml", this);
43          }catch(SAXException se) {
44              se.printStackTrace();
45          }catch(ParserConfigurationException pce) {
46              pce.printStackTrace();
47          }catch (IOException ie) {
48              ie.printStackTrace();
49          }
50      }
51      // Manejador de Eventos
52      public void startElement(String uri, String localName, String qName, Attributes
53          tempVal = "";
54          if(qName.equalsIgnoreCase("Empleadoo")) {
55              //crear una nueva instancia de empleado
56              tempEmp = new Empleadoo();
57              tempEmp.ponerTipo(attributes.getValue("tipo"));
58          }
59      }
60      public void characters(char[] ch, int start, int length) throws SAXException {
61          tempVal = new String(ch,start,length);
62      }
63      public void endElement(String uri, String localName, String qName) throws SAXExc
```

SAXParserBusca.java *

```
63      public void endElement(String uri, String localName, String qName) throws SAXExc
64          if(qName.equalsIgnoreCase("Empleadoo")) {
65              // añadirlo a la lista
66              myEmpls.add(tempEmp);
67          }else
68          if (opcion ==1)
69              if (qName.equalsIgnoreCase("Nombre")) {
70              tempEmp.ponerNombre(tempVal);
71               System.out.println(" \n\nNombre:"+tempVal);
72          }
73          if (opcion==2)
74          if (qName.equalsIgnoreCase("Clave")) {
75              tempEmp.ponerClave(Integer.parseInt(tempVal));
76              System.out.println(" \n\nClave:"+tempVal);
77          }
78          if (opcion==3)
79          if (qName.equalsIgnoreCase("Edad")) {
80              tempEmp.ponerEdad(Integer.parseInt(tempVal));
81              System.out.println(" \n\nEdad:"+tempVal);
82          }
83      }
84      //_____________________________________________________________
85      public static void main(String argv[]) throws IOException {
86          String dato;
87          //crear una instancia
88          SAXParserBusca spe = new SAXParserBusca();
89          //Creación del flujo para leer datos
90          InputStreamReader isr=new InputStreamReader(System.in);
91          //Creación del filtro para optimizar la lectura de datos
92          BufferedReader br=new BufferedReader(isr);
93          System.out.println("  \n\n Buscar datos de Empleado:\n\n");
```

SAXParserBusca.java *

```
94            System.out.println(" 1. Nombre   - Lista los Nombres de Empleados ");
95            System.out.println(" 2. Clave    - Lista las Claves de Empleados ");
96            System.out.println(" 3. Edad     - Lista Edad de Empleados ");
97            System.out.println(" 4. salir    ");
98            opcion=1;
99            while ((opcion>=1)&&(opcion<=3)){
100           System.out.println(" \n\nCual opción deseas? ( 1-4)\n\n");
101           //Lectura de datos mediante el método readLine()
102           String texto1=br.readLine();
103           //Conversión a int de la String anterior para poder sumar
104           opcion=Integer.parseInt(texto1);
105           switch(opcion) {
106               case 1: spe.runEjemplo("Nombre");
107                       System.out.println(" \n\npromedio de tiempo:"+promedio);
108                       promedio=0;
109                       break;
110               case 2: spe.runEjemplo("Clave");
111               System.out.println(" \n\npromedio de tiempo:"+promedio);
112                       promedio=0;
113                       break;
114               case 3: spe.runEjemplo("Edad");
115               System.out.println(" \n\npromedio de tiempo:"+promedio);
116                       promedio=0;
117               break;
118               case 4: System.out.println( "Fin del Sistema.");break;
119               default:
120                   System.out.println( "Opción Invalida.");
121             } //fin switch
122           }//fin while
123       }
124   }
```

Búsqueda en Documentos XML usando DOM.

Programa *ProcesarBuscaDom.java* utilizando el parser DOM.

ProcesarBuscaDom.java *

```
1  import java.io.*;
2  import java.io.IOException;
3  import java.util.ArrayList;
4  import java.util.Iterator;
5  import java.util.List;
6  import javax.xml.parsers.DocumentBuilder;
7  import javax.xml.parsers.DocumentBuilderFactory;
8  import javax.xml.parsers.ParserConfigurationException;
9  import org.w3c.dom.Document;
10 import org.w3c.dom.Element;
11 import org.w3c.dom.NodeList;
12 import org.xml.sax.SAXException;
13 public class ProcesarBuscaDom {
14     //datos de tipo No genericos
15     List myEmpls;
16     Document dom;
17     static double promedio=0;   // donde almacenamos el tiempo promedio del algoritm
18     public ProcesarBuscaDom(){
19     //crea una lista para contener los objetos de los empleados
20         myEmpls = new ArrayList();
21     }
22     public void runEjemplo(String dato) {
23         long inicio,fin;     // donde almacenamos el valor inicial y final del tiemp
24         int numIter = 1000;  // numero de mediciones
25         double suma = 0;     // iniciamos el acumulador
26         for(int i=0;i<numIter;i++)
27         {
28         inicio = System.nanoTime();
```

ProcesarBuscaDom.java *

```
28         inicio = System.nanoTime();
29         //procesar el archivo xml y tomar el objeto dom
30         procesarArchXml();
31         //tomar cada elemento empleado y crear un objeto Empleado
32         procesarDocumento(dato);
33         fin = System.nanoTime();
34                 suma += fin - inicio;
35         }
36             promedio=(suma/numIter);
37     }
38     //-----------------------------------------------------------------------
39     private void procesarArchXml(){
40         //declaración de un objeto tipo factory
41         DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
42         try {
43             //Uso de factory para obtener una instancia del constructor document
44             DocumentBuilder db = dbf.newDocumentBuilder();
45             //procesar usando el constructor para tomar la representación DOM del ar
46             dom = db.parse("empleadoos.xml");
47         }catch(ParserConfigurationException pce) {
48             pce.printStackTrace();
49         }catch(SAXException se) {
50             se.printStackTrace();
51         }catch(IOException ioe) {
52             ioe.printStackTrace();
53         }
54     }
55     //---------------------------------------
56     private void procesarDocumento(String dato){
57         //tomar el elemento raiz
```

ProcesarBuscaDom.java *

```
57          //tomar el elemento raiz
58          Element docEle = dom.getDocumentElement();
59          //usar una nodelist (lista de nodos) de los elementos <Empleadoo>
60          NodeList nl = docEle.getElementsByTagName("Empleadoo");
61          if(nl != null && nl.getLength() > 0) {
62              for(int i = 0 ; i < nl.getLength();i++) {
63                  //tomar el elemento empleado
64                  Element el = (Element)nl.item(i);
65                  //tomar el objeto Empleadoo
66                  Empleadoo e = tomarEmpleadoo(el, dato);
67                  //añadirlo a la lista
68                  myEmpls.add(e);
69              }
70          }
71      }
72      //-------------------------------------------------------------------
73      /* Se toma un elemento Empleadoo y se leen los valores contenidos en el,
74       * crea un objeto Empleadoo y se retorna
75       * @parametros: empEl, dato
76       * @retorna un dato tipo Empleadoo */
77      private Empleadoo tomarEmpleadoo(Element empEl, String dato) {
78          //para cada elemento <Empleadoo> se toman los valores tipo texto o entero
79          //de nombre, clave, edad y tipo de empleado
80          String s1=null;
81          s1=new String(dato);
82          int clave=0,edad=0;
83          String nombre=null,tipo =null;
84          if (s1.equals("Nombre"))
85          {
86           nombre= tomarValor(empEl,dato);
```

ProcesarBuscaDom.java *

```
86           nombre= tomarValor(empEl,dato);
87           System.out.println("Nombre de Empleado:"+nombre);  }
88           if (s1.equals("Clave"))
89              {clave = tomarValorInt(empEl,dato);
90              System.out.println("Clave de Empleado:"+ clave); }
91              if (s1.equals("Edad"))
92               {edad = tomarValorInt(empEl,dato);
93                System.out.println("Edad de Empleado:"+edad); }
94                if (s1.equals("tipo"))
95                 {tipo = empEl.getAttribute(dato);
96                  System.out.println("tipo de Empleado:"+tipo); }
97                  //Crea un nuevo Empleadoo con los valores leidos de los nodos de xml
98                  Empleadoo e = new Empleadoo(nombre,clave,edad,tipo);
99          return e;
100     }
101     //-------------------------------------------------------------------------
102     /* En esta función se toma un elemento xml, segun el nombre de la etiqueta,
103      * se toma su contenido
104      * por ejemplo para <Empleadoo><Nombre>Salvador</Nombre></Empleadoo> xml ubica si
105      * el Elemento apunta al nodo Empleadoo y el nombre de la etiqueta es Nombre,
106      * se retornará su contenido:Salvador
107      * parametros: ele, NombreEtiqueta
108      * retorna un valor de tipo String (contenido del nodo)  */
109     private String tomarValor(Element ele, String NombreEtiqueta) {
110         String textVal = null;
111         NodeList nl = ele.getElementsByTagName(NombreEtiqueta);
112         if(nl != null && nl.getLength() > 0) {
113             Element el = (Element)nl.item(0);
114             textVal = el.getFirstChild().getNodeValue();
115         }
```

ProcesarBuscaDom.java *

```
115             }
116             return textVal;
117         }
118         //-----------------------------------------------------------
119         /* llama a getTextValue y retorna un valor int
120          * parametros: ele, tagName
121          * retorna un valor tipo entero */
122         private int tomarValorInt(Element ele, String tagName) {
123             return Integer.parseInt(tomarValor(ele,tagName));
124         }
125         //-----------------------------------------------------------
126         public static void main(String argv[]) throws IOException {
127             int num1=0;
128             String dato;
129             //crear una instancia
130             ProcesarBuscaDom dpe = new ProcesarBuscaDom();
131             //Creación del flujo para leer datos
132             InputStreamReader isr=new InputStreamReader(System.in);
133             //Creación del filtro para optimizar la lectura de datos
134             BufferedReader br=new BufferedReader(isr);
135             System.out.println("  \n\n Buscar datos del trabajador:  \n\n");
136             System.out.println("1.  Nombre  - Lista los nombres de los empleados ");
137             System.out.println("2.  Clave   - Lista las claves  de los empleados ");
138             System.out.println("3.  Edad    - Lista las Edades  de los empleados ");
139             System.out.println("4.  tipo    - Lista los tipos de los empleados ");
140             System.out.println("5.  salir    ");
141             num1=1;
142             while((num1>=1)&&(num1<5))
143             {
144              System.out.println(" \n\n opcion: ? \n\n");
```

ProcesarBuscaDom.java *

```
145             //Lectura de datos mediante el método readLine()
146             String texto1=br.readLine();
147             //Conversión a int del String texto1
148             num1=Integer.parseInt(texto1);
149             switch(num1) {
150                case 1:  dpe.runEjemplo("Nombre");
151                         break;
152                 case 2: dpe.runEjemplo("Clave");
153                         break;
154                 case 3: dpe.runEjemplo("Edad");
155                         break;
156                 case 4: dpe.runEjemplo("tipo");
157                         break;
158                 case 5: System.out.println( " Gracias por usar el sistema !!\n\n\n\n");b
159                 default:
160                     System.out.println( " Opción Invalida!!");
161
162             } //fin switch
163             }//fin while
164         }//fin main
165     }//fin class
```

Búsqueda en Documentos XML usando JDOM.

Programa *ProcesarBuscaJdom.java* utilizando el parser JDOM.

ProcesarBuscaJdom.java

```
1  import java.io.*;
2  import java.util.*;
3  import org.jdom.*;
4  import org.jdom.input.*;
5  import org.jdom.output.*;
6  import org.jdom.Attribute;
7  import org.jdom.Comment;
8  import org.jdom.Document;
9  import org.jdom.Element;
10 import org.jdom.JDOMException;
11 import org.jdom.input.SAXBuilder;
12 import org.jdom.output.XMLOutputter;
13 //-------------------------------------
14  /* Este programa ejecuta el código usando metodos para
15  * listar el contenido del archivo xml.
16  * Desde main se ejecutan los metodos
17  * Este programa se evalua con:
18  * * xerces version 1.3.0 */
19  public class ProcesarBuscaJdom {
20     //datos de tipo No genericos
21     List myEmpls;
22     Document doc;
23     public ProcesarBuscaJdom(){
24         //crea una lista para contener los objetos de los empleados
25         myEmpls = new ArrayList();
26     }
27  public void runEjemplo(String dato) {
28         procesarArchXml();
29         //tomar cada elemento empleado y crear un objeto Empleado
30         procesarDocumento(dato);
```

ProcesarBuscaJdom.java

```
31         }
32  private void procesarArchXml(){
33         try {
34             SAXBuilder constructor=new SAXBuilder(false);
35             //usar el parser Xerces
36             doc=constructor.build("empleadoos.xml");
37         }catch(Exception ioe) {
38             ioe.printStackTrace();
39         }
40     }
41  private void procesarDocumento(String dato){
42         Element trabElement = doc.getRootElement();
43         List nl=trabElement.getChildren("Empleadoo");
44         //System.out.println("Formada por:"+nl.size()+" empleados");
45         Iterator i = nl.iterator();
46         while (i.hasNext()){
47             //tomar el elemento empleado
48             Element el= (Element)i.next();
49             //tomar el objeto Empleadoo
50             Empleadoo e = tomarEmpleadoo(el,dato);
51             //añadirlo a la lista
52             myEmpls.add(e);
53             }
54     }
55      /* Se toma un elemento empleadoo y se leen los valores contenidos en el,
56       * crea un objeto Empleadoo y lo retorna
57       * @parametro empEl  */
58     private Empleadoo tomarEmpleadoo(Element empEl,String dato) {
59         String s1;
60         s1=new String(dato);
```

ProcesarBuscaJdom.java

```
61          int clave=0,edad=0;
62          String nombre=null,tipo =null;
63          if (s1.equals("Nombre") )
64          {
65               nombre = getTextValue(empEl,dato);
66                  System.out.println("Nombre de Empleado:"+nombre);  }
67          if (s1.equals("Clave"))
68          {
69              clave = getIntValue(empEl,dato);
70              System.out.println("Clave de Empleado:"+clave);  }
71          if (s1.equals("Edad") )
72          {
73              edad = getIntValue(empEl,dato);
74              System.out.println("Edad de Empleado:"+edad);  }
75          if (s1.equals("tipo") )
76          {
77              tipo = empEl.getAttributeValue(dato);
78              System.out.println("Tipo de Empleado:"+tipo);  }
79              //Crea un nuevo objeto Empleadoo con los valores leidos de los nodos de x
80              Empleadoo ele = new Empleadoo(nombre,clave,edad,tipo);
81          return ele;
82      }
83       /* Se toma un elemento xml y la etiqueta nombre, revisando la etiqueta y tomando
84       * el texto del contenido
85       * por ejemplo para <empleadoo><nombre>Salvador</nombre></empleadoo> xml ubica si
86       * el Elemento apunta al nodo empleadoo y el nombre de la etiqueta es nombre, se
87       * parametros: ele,NombreEtiqueta
88       * retorna un valor de tipo String   */
89      private String getTextValue(Element ele, String NombreEtiqueta) {
90          String textVal=null;
```

ProcesarBuscaJdom.java

```
91          // Acceso a un elemento hijo
92          Element el = ele.getChild(NombreEtiqueta);
93          // Mostrar una posible falla
94          if(el != null) {
95                  textVal=el.getText();
96          } else {
97              System.out.println("Problema.  No se encuentra un elemento ");
98          }
99          return textVal;
100     }
101      /* llama a getTextValue y retorna un valor int
102      * parametro ele, NombreEtiqueta
103      * retorna un dato tipo entero  */
104     private int getIntValue(Element ele, String NombreEtiqueta) {
105         return Integer.parseInt(getTextValue(ele,NombreEtiqueta));
106     }
107     public static void main(String[] args) throws IOException {
108         int num1=0;
109         //crear una instancia
110         ProcesarBuscaJdom dpe = new ProcesarBuscaJdom();
111         //Creación del flujo para leer datos
112         InputStreamReader isr=new InputStreamReader(System.in);
113         //Creación del filtro para optimizar la lectura de datos
114         BufferedReader br=new BufferedReader(isr);
115         System.out.println("  \n\n Buscar datos del trabajador:  \n\n");
116         System.out.println("1.  Nombre  - Lista los nombres de los empleados ");
117         System.out.println("2.  Clave   - Lista las claves  de los empleados ");
118         System.out.println("3.  Edad    - Lista las Edades  de los empleados ");
119         System.out.println("4.  tipo    - Lista los tipos de los empleados ");
120         System.out.println("5.  salir    ");
```

ProcesarBuscaJdom.java

```
121             num1=1;
122             while((num1>=1)&&(num1<5))
123             {
124             System.out.println(" \n\nCual opción deseas? ( 1-5)\n\n");
125             //Lectura de datos mediante el método readLine()
126             String texto1=br.readLine();
127             //Conversión a int de la String anterior para poder sumar
128             num1=Integer.parseInt(texto1);
129             switch(num1) {
130                case 1:
131                         dpe.runEjemplo("Nombre");
132                         break;
133                 case 2:
134                         dpe.runEjemplo("Clave");
135                         break;
136                 case 3: dpe.runEjemplo("Edad");
137                         break;
138                 case 4: dpe.runEjemplo("tipo");
139                         break;
140                 case 5: System.out.println( " Gracias por usar el sistema!!\n\n\n\n");
141                 default:
142                     System.out.println( " Opción Invalida!!");
143                  }//fin switch
144             }//fin while
145             }
146     }
```