***XML***

/\*

XML es un formato de almacenamiento e intercambio de datos basado en texto estructurado mediante etiquetas.

***Usos de XML:***

* Almacenamiento portable entre sistemas y plataformas, posibilitando la importación/exportación de bases de datos desde cualquier lenguaje.
* Formato “neutro”: estos documentos pueden transformarse a otros formatos.
* Ficheros de configuración legibles y modificables.

***APIs de Java para trabajar con XML:***

* ***JAXP (Java API for XML Processing):*** Permite procesar documentos XML utilizando distintos tipos de analizadores. Este sigue dos estándares fundamentales para procesar los documentos XML: ***DOM y SAX,*** ya implementados.
* ***JAXB (Java Architecture for XML Binding):*** Conversión entre documentos XML y beans/componentes de Java.
* ***SAAJ (SOAP with Attachments API for Java):*** Sirve para enviar mensajes de tipo SOAP desde Java.
* ***JAX-RPC (Java API for XML-RPC):*** También permite el envío de mensajes SOAP para llamar a procedimientos remotos (RPC o Remote Procedure Call).
* ***JAXR (Java API for XML Registries):*** Permite acceder a registros de servicios web almacenados en forma de XML.

***Las tres partes fundamentales de un documento XML:***

* ***Prólogo:*** Contiene la declaración del documento, en la que se indica la gramática y en una parte adicional comentarios o instrucciones.
* ***Elemento raíz:*** Una etiqueta envuelve las demás etiquetas del documento.
* ***Miscelánea:*** Parte opcional en la que pueden ir comentarios y/o instrucciones.

***Declaración de un documento XML:***

<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

juegoCaracteres haceReferenciaExterna

***Instrucciones de procesamiento:***

Estas instrucciones están destinadas al software que analice y procese el documento XML y que dé las órdenes al procesador. Comienzan con <? y cierran con ?>.

***Etiquetas y elementos de un XML:***

Todas las etiquetas abiertas deben ser cerradas sin solaparse, excepto las etiquetas vacías.

El valor de los atributos debe ir entre comillas “”.

XML es key sensitive, diferencia entre mayúsculas y minúsculas.

***Entidades predefinidas de XML:***

Algunos caracteres de XML están reservados para alguna función, en el caso de querer usar alguno de ellos será necesario usar su ***símbolo en la secuencia de escape.*** (Es posible definir entidades propias).



***Secciones CDATA:***

Para que el texto no se haga difícil de leer se usan secciones que indican al procesador que no analice esa sección del código y la trate como texto plano.

Comienza con ***“ < ! [ CDATA [ “*** y cierra con ***“ ] ] > ”.***

***Espacios de nombres:***

Existe la posibilidad de que dos usuarios definan la misma etiqueta en diferentes vocabularios, en caso de que ambos documentos se fusionen ocasionaría problemas.

Para solventar estos problemas se usa el espacio de nombres, que permite tener dos etiquetas distintas con el mismo nombre, pero en distintos vocabularios (DTD / SCHEMAS).

En el atributo xmlns de la etiqueta raíz se define un nombre corto y uno largo:

***<etiqueta-raíz xmlns:nombre-corto=nombre-largo>***

Donde el nombre corto es definido por el usuario, y si el espacio de nombres es propio, el nombre largo también lo define el usuario. El nombre largo debe ser una URI / funciona como un identificador: Como una dirección de almacenamiento que no tiene porqué existir.

***<recomendaciones xmlns:pedido="http://www.discolibro.com/pedidos"***

***xmlns:novedad="http://www.discolibro.com/novedades">***

Si en un documento bajo un determinado vocabulario introducimos una ***etiqueta de otro vocabulario,*** debemos ***indicarle el nombre corto*** del espacio de nombres.

***<pedido:dirección>C/ Cisne, 27 </pedido:dirección>***

\*/

***SAX***

/\*

El objetivo de SAX / Simple API for XML es posibilitar la construcción de analizadores / parsers de documentos XML. Este extrae información del documento, mientras DOM manipula su estructura.

Los analizadores SAX están basados en un modelo de eventos; a medida que el parser recorre el documento, se informa de la ocurrencia de eventos a un manejador de eventos / event handler, como el comienzo o el final del documento. Ejemplo:

startDocument()

startElement(): documentoXML

startElement(): cabecera

characters(): Esto es un documento XML

endElement(): cabecera

characters(): Eso es todo amigos

endElement(): documentoXML

endDocument()

Esto funciona mediante un análisis secuencial que determina que es cada elemento.

\*/

***JDBC***

/\*

La persistencia se refiere a los datos que aún cuando termina la ejecución del programa, están guardados.

JDBC es una interfaz para conectar Java con una base de datos, esta permite enviar sentencias SQL y procesar los resultados. Está formado por un API genérico y drivers.

Sus componentes son:

* Gestor de los drivers: java.sql.DriverManager
* Conexión con la base de datos: java.sql.Connection
* Sentencia a ejecutar: java.sql.Statement, java.sql.PreparedStatement
* Resultado: java.sql.ResultSet

\*\*\*SE DEBEN CERRAR AL TERMINAR\*\*\*

\*/

***Acceso a datos en Java***

/\*

Un fichero / archivo es un conjunto de bits almacenado en un dispositivo. Estos tienen nombres únicos y se organizan en carpetas, además de una extensión formada por .PDF por ejemplo.

Clases para la gestión de ficheros:

La clase File permite el manejo de ficheros.

El sistema de entrada/salida en Java funciona mediante clases que se implementan en el paquete java.io. Estas abstraen el flujo / stream para tratar la comunicación de información entre una fuente y un destino.

Cualquier programa que tenga que obtener información abrirá un stream y se escribirá la información en serie. Se definen dos tipos de flujos:

* Flujos de bytes(8 bits): Operaciones de E/S de bytes para archivos binarios, las clases que lo permiten son InputStream y OutputStream.
* Flujo de caracteres (16 bits): Realizan operaciones de E/S de caracteres para archivos de texto, es posible por las clases Reader y Writer, que están orientadas a unicode de 16 bits.

La ***clase inputStream*** representa las siguientes clases:

***ByteArrayInputStream:*** Permite usar un espacio de almacenamiento intermedio de memoria.

***StringBufferInputStream:*** Convierte un String en un InputStream.

***FileInputStream:*** Para leer información de un fichero.

***PipedInputStream:*** implementa concepto de tubería.

La ***clase OutputStream*** representa las siguientes clases:

***ByteArrayOutputStream:*** Crea un espacio de almacenamiento intermedio. todos los datos que se envían al flujo se ubican en ese espacio.

***FileOutputStream:*** Para enviar información a un fichero.

***PipedOutputStream:*** Para implementar el concepto de tubería.

Las ***clases Reader y Writer*** manejan flujos de caracteres Unicode. Las clases más importantes son:

***FileReader y FileWriter:*** Acceso a fichero, lectura y escritura de caracteres.

***CharArrayReader y CharArrayWriter:*** Acceso, lectura y escritura de flujo de caracteres en un array de caracteres.

***BufferedReader y BufferedWriter:*** Se utilizan para evitar que cada lectura o escritura acceda directamente al fichero, ya que utilizan un buffer intermedio entre memoria y el stream.

Hay dos formas de acceso a la información de un fichero:

***Acceso secuencial:*** Los datos se leen y se escriben en orden como en una cinta de video. Tenemos las clases FileInputStream y FileOutputStream para ficheros binarios y para caracteres FileReader y FileWriter.

***Acceso directo:*** Permite acceder directamente a un dato o registro sin necesidad de leer los anteriores y se puede acceder a la información en cualquier orden. Tenemos las clases RandomAccessFile.

Las operaciones básicas sobre ficheros son:

***Creación del fichero:*** El fichero se crea en el disco con un nombre.

***Apertura del fichero:*** Para manejar el fichero lo primero es abrirlo.

***Cierre del fichero:*** El fichero se ha de cerrar cuando el programa no lo vaya a utilizar.

***Lectura de los datos del fichero:*** Transferencia de la información del fichero a la memoria principal, normalmente a alguna variable del programa.

***Escritura de datos en el fichero:*** Proceso de transferir información de memoria (por medio de una variable) al fichero.

Las ***operaciones típicas*** una vez abierto un el fichero son:

***Altas:*** añadir un nuevo registro al fichero.

***Bajas:*** Consiste en eliminar del fichero un registro existente.

***Modificaciones:*** consiste en cambiar parte del contenido del registro.

***Consultas:*** Consiste en buscar en el fichero

OPERACIONES SOBRE FICHEROS SECUENCIALES.

En los ficheros secuenciales los registros se insertan en orden cronológico, es decir, un registro se inserta a continuación del último insertado. ***Operaciones secuenciales:***

***Consultas:*** Es necesario empezar la lectura desde el primer registro hasta localizar el registro buscado.

***Altas:*** Las altas se realizan al final del último registro insertado, es decir añade los datos al final del fichero.

***Bajas:*** Para dar de baja un registro de un fichero es necesario leer todos los registros y escribirlos en un fichero auxiliar excepto el que deseemos dar de baja, posteriormente se borra el original y se renombra el auxiliar por el nombre del fichero original.

***Modificaciones:*** Consiste en localizar el registro a modificar,efectuar la modificación y reescribir el fichero inicial en otro fichero auxiliar que incluya el registro y modificarlo. El proceso es similar a las bajas.

Los ficheros secuenciales se usan típicamente en aplicaciones de procesos por lotes, como el respaldo de datos o backup.

OPERACIONES SOBRE FICHEROS ALEATORIOS.

Las operaciones en ficheros aleatorios son las vistas anteriormente pero teniendo en cuenta que para acceder a un registro hay que localizar la posición o dirección donde se encuentra.

CLASES PARA GESTIÓN DE FICHEROS DE TEXTO

Para trabajar con ficheros de texto alfanuméricos usaremos las clases FileReader para leer caracteres, y FileWriter para escribir los caracteres en el fichero, dentro del try-catch, FileReader tiene la excepción FileNotFoundException y FileWriter IOException.

FileReader no contiene métodos que nos permitan leer líneas completas, pero BufferedReader si.

La clase PrintWriter hace uso del método print(“string”) para escribir una línea y println para añadir salto de línea.

CLASES PARA GESTIÓN DE FICHEROS BINARIOS

Los ficheros binarios almacenan secuencias de dígitos binarios que no son legibles directamente por el usuario, estos ocupan menos espacio en disco. Se usan las clases FileInputStream y FileOutputStream.

Los métodos proporcionados por ***FileInputStream*** son:

***int read();***lee un byte y lo devuelve. si ha llegado al final del fichero devuelve -1.

***int read(byte[] buff);*** lee hasta buff.length bytes y lo almacena en buff. Devuelve la cantidad de bytes leídos o -1 si fin de fichero.

***int read(byte[] buff, int desplazamiento, int n);*** lee n bytes desde desplazamiento guardándolo en buff. devuelve el número de bytes leído

Los métodos proporcionados por ***FileOutputStream*** son:

***void write(int b);*** escribe un byte.

***void write(byte [] buff);*** escribe b.length bytes.

***void write(byte [] buff, int desplazamiento, int n);*** escribe n bytes a

partir de buff más el desplazamiento

Para que ***no sobreescriba y pueda seguir añadiendo al final del fichero,*** tenemos que añadir true en los parámetros ***FileOutputStream(fichero,true);***

Para leer y escribir ***tipos de datos concretos*** usaremos las clases DataInputStream y DataOutputStream.

Para escribir y leer caracteres con codificación unicode UTF (caracteres de un String de 16 bits codificada en 8 bits.), Leer: String readUTF(); Escribir: void writeUTF(String str);

File fichero = new File(“direccion.txt”);

FileInputStream filein = new FileInputStream(fichero);

DataInputStream dataIS = new DataInputStream(filein);

FileOutputStream fileout = new FileOutputStream(fichero);

DataOutputStram dataOS = new DataOutputStram(fileout);

***OBJETOS SERIALIZABLES***

***Java permite guardar objetos en ficheros binarios,*** para ello, el objeto tiene que ***implementar la interfaz Serializable*** que dispone de una serie de métodos con los que podremos guardar y leer objetos en ficheros binarios. Los más importantes son:

***void readObject(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException: para leer un objeto.***

***void writeObject(ObjectOutputStream stream) throws IOException: Para escribir un objeto.***

La serialización de objetos en Java permite tomar cualquier objeto que implemente la interfaz Serializable y convertirlo en una secuencia de bits, que puede ser posteriormente restaurada para regenerar el objeto original.

Import java.io.Serializable;

public class Persona implements Serializable{

private String nombre;

private int edad; }

\*/

/\*

Para acceder aleatoriamente con RandomAccessFile (File f, String modoAcceso) necesitamos saber que :

Un int pesa 4 bytes.

Un double pesa 8 bytes.

En un String cada caracter pesa 2 bytes.

Los métodos de acceso de la clase se indican con r o rw.

\*/