ACCESO A DATOS

CAPÍTULO 1

Manejo de ficheros

José Eduardo Córcoles
Francisco Montero

Ra-Ma®

ÍNDICE DE CONTENIDOS

OBJETIVOS

- A) utilizar clases para la gestión de ficheros y directorios.
- B) valorar las ventajas y los inconvenientes de las distintas formas de acceso.
- C) utilizar clases para recuperar información almacenada en un fichero XML.
- D)utilizar clases para almacenar información en un fichero XML.
- E) utilizar clases para convertir a otro formato información contenida en un fichero XML.
- F) gestionar las excepciones.
- G) probar y documentar las aplicaciones desarrolladas.

1.1 FORMAS DE ACCESO A UN FICHERO. CLASES ASOCIADAS

- En Java, en el paquete java.io, existen varias clases que dan soporte para trabajar con ficheros desde diferentes perspectivas:
- Según el tipo de contenido:
  - Ficheros de caracteres.
  - Ficheros binarios.
- Según el modo de acceso:
  - Ficheros secuenciales.
  - Ficheros aleatorios.

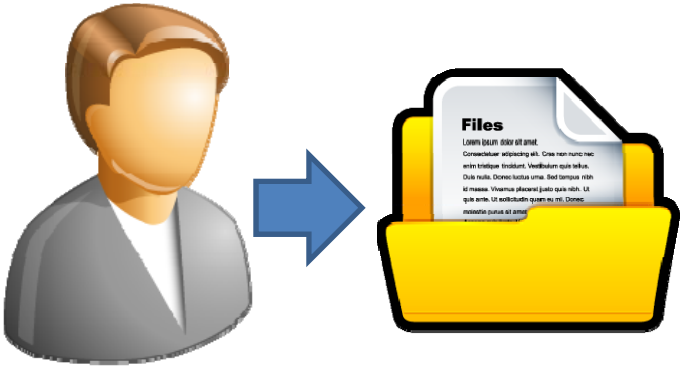
1.2 GESTIÓN DE FLUJOS DE DATOS

- En Java, el acceso a ficheros es tratado como un flujo (*stream*) de información entre el programa y el fichero.
- Para comunicar un programa con un origen o destino de cierta información (fichero) se usan flujos de información.
- Un flujo no es más que un objeto que hace de intermediario entre el programa y el origen o el destino de la información.
- Con independencia del tipo de flujo que maneje, todos los accesos se hacen más o menos de la misma manera:
  - Para leer.
  - Para escribir.

1.2.1 CLASE FILEWRITER

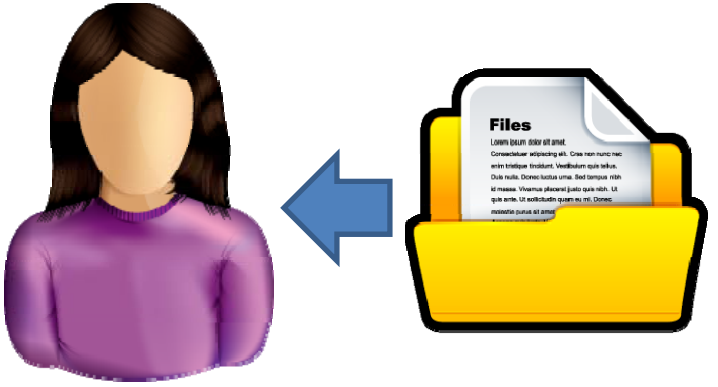
- El flujo **FileWriter** permite escribir caracteres en un fichero de modo secuencial. Esta clase hereda los métodos de la clase Writer. Los constructores principales son:
  - **FileWriter** (***String*** *ruta,* ***boolean*** *añadir*).
  - **FileWriter** (***File*** *fichero*).

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



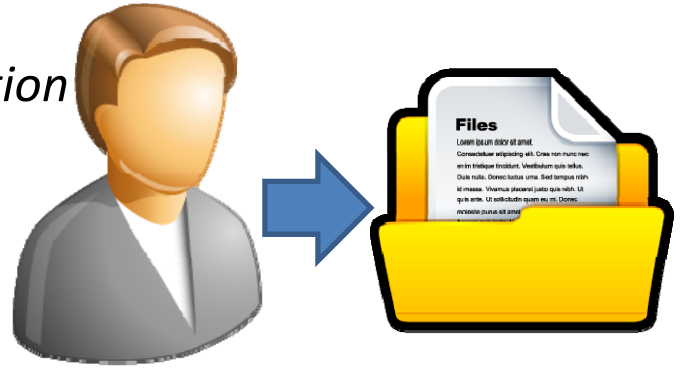
1.2.2 CLASE FILEREADER

- El flujo **FileReader** permite leer caracteres desde un fichero de modo secuencial. Esta clase hereda los métodos de la clase Reader. Los constructores principales son:
  - **FileReader** (***String*** *ruta*)
  - **FileReader** (***File*** *fichero*)



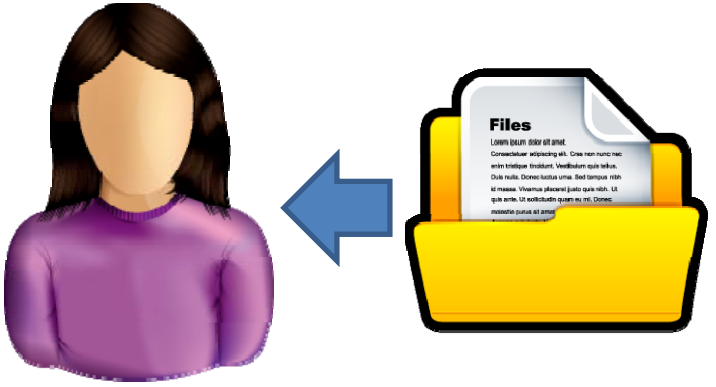
1.2.3 CLASE FILEOUTPUSTREAM

- El flujo **FileOutputStream** permite escribir bytes en un fichero de manera secuencial.
- Sus constructores tienen los mismos parámetros que los mostrados para **FileWriter**: el fichero puede ser abierto vacío o listo para añadirle datos a los que ya contenga.
- Ya que este flujo está destinado a ficheros binarios (bytes) todas las escrituras se hacen a través de un buffer (array de bytes).
  - *public void* **write** (*byte*[] b) **throws** *IOException*



1.2.4 CLASE FILEINPUTSTREAM

- El flujo **FileInputStream** permite leer bytes en un fichero de manera secuencial. Sus constructores tienen los mismos parámetros que los mostrados para **FileReader**.
- El método más popular de FileInputStream es el método **read**() que acepta un array de bytes (buffer):
  - *public int* **read** *(byte[]* cbuf*)* ***throws*** *IOException*



1.2.5 RANDOMACCESSFILE

- El flujo **RandomAccessFile** permite acceder directamente a cualquier posición dentro del fichero. Proporciona dos constructores básicos:
  - **RandomAccessFile** (***String*** *ruta,* ***String*** *modo*)
  - **RandomAccessFile** (***File*** *fichero,* ***String*** *modo*)

- El parámetro modo especifica para qué se abre el archivo: “r” solo lectura o “rw” lectura y escritura.

1.2.6 EJEMPLO DE USO DE FLUJOS

```
import java.io.FileWriter;
public void EscribeFicheroTexto() {
   //Crea el String con la cadena XML
   String texto =
        "<Libros><Libro><Titulo>El Capote</Titulo></Libro></Libros>";
   //Guarda en nombre el nombre del archivo que se creará.
   String nombre = "libros.xml";
   try{
     //Se crea un Nuevo objeto FileWriter
     FileWriter fichero = new  FileWriter (nombre);
     //Se escribe el fichero
     fichero.write(texto + "\r\n");
     //Se cierra el fichero
     fichero.close();
   }catch (IOException ex){ System.out.println("error al acceder al fichero");  }
}
```

Actividad 1.1

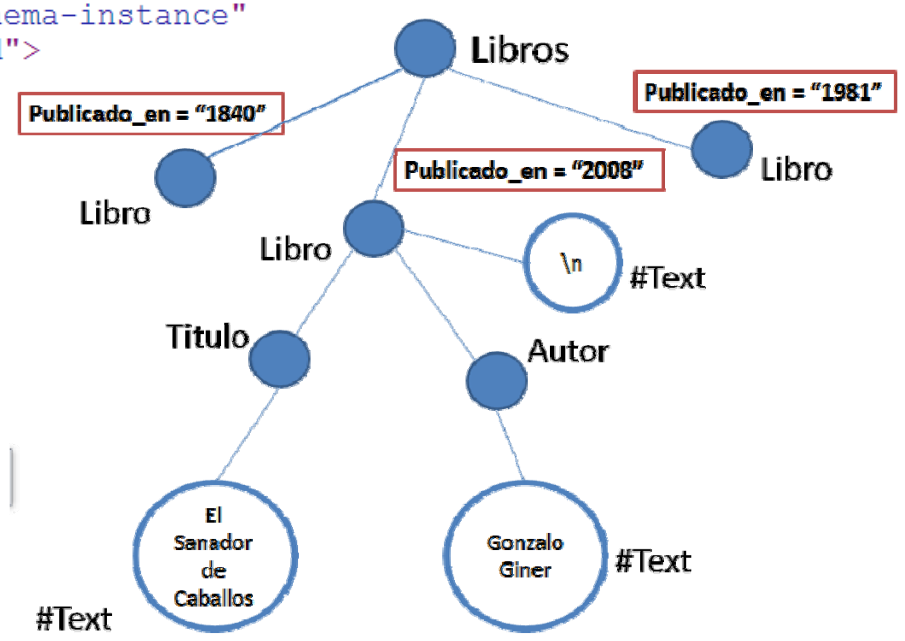
1.3 TRABAJO CON FICHEROS XML (EXTENDED MARKUP LANGUAGE)

- **XML** (eXtended Markup Language – Lenguaje de marcas extendido) ofrece la posibilidad de representar la información de forma neutra, independiente del lenguaje de programación y del sistema operativo empleado.
- Desde un punto de vista a "bajo nivel" XML no es otra cosa que un fichero de texto.
- Sin embargo, desde un punto de vista a "alto nivel" XML no es un fichero de texto cualquiera.
- A continuación se describe el acceso a datos con las tecnologías más extendidas: **DOM**, **SAX** y **JAXB**.

1.4 ACCESO A DATOS CON DOM (DOCUMENT OBJECT MODEL)

- **DOM** (*Document Object Model*) es una interfaz de programación que permite analizar y manipular dinámicamente y de manera global el contenido, el estilo y la estructura de un documento.

```
▼<Libros xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="LibrosEsquema.xsd">
  ▼<Libro publicado_en="1840">
     <Titulo>El Capote</Titulo>
     <Autor>Nikolai Gogol</Autor>
   </Libro>
  ▼<Libro publicado_en="2008">
     <Titulo>El Sanador de Caballos</Titulo>
     <Autor>Gonzalo Giner</Autor>
   </Libro>
  ▼<Libro publicado_en="1981">
     <Titulo>El Nombre de la Rosa</Titulo>
     <Autor>Umberto Eco</Autor>
   </Libro>
 </Libros>
```



1.4.1 DOM Y JAVA

- DOM ofrece una manera de acceder a documentos XML tanto para ser leído como para ser modificado.
- Una aplicación Java que desea manipular documentos XML accede a la interfaz JAXP (Java API for XML Processing) porque le ofrece una manera transparente de utilizar los diferentes *parsers* que hay para manejar XML.
- Algunos métodos que facilita para manipular un árbol DOM son:
  - *getFirstChild, getNextSibling, getNodeType, getAttributes, getNodeName, getNodeValue*

1.4.2 ABRIR DOM DESDE JAVA

```
public int abrir_XML_DOM (File fichero)    {
      Document doc=null;//Representa al árbol DOM
      try{
        //Se crea un objeto DocumentBuiderFactory.
         DocumentBuilderFactory    factory=DocumentBuilderFactory.newInstance();
         //Indica que el modelo DOM no debe contemplar los comentarios     //que tenga  el XML.
          factory.setIgnoringComments(true);
         //Ignora los espacios en blanco que tenga el documento
          factory.setIgnoringElementContentWhitespace(true);
         //Se crea un objeto DocumentBuilder para cargar en él la //estructura de árbol DOM a partir del XML
         //seleccionado.
          DocumentBuilder builder=factory.newDocumentBuilder();
        //Interpreta (parsear) el documento XML (file) y genera el DOM //equivalente.
          doc=builder.parse(fichero);
        //Ahora doc apunta al árbol DOM listo para ser recorrido.
          return 0;
      } catch (Exception e){  e.printStackTrace();  return -1;    }
  }
```

1.4.3 RECORRER UN ÁRBOL DOM

```
public String recorrerDOMyMostrar(Document doc){
    String datos_nodo[]=null;
    String salida="";
    Node node;
    //Obtiene el primero nodo del DOM (primer hijo)
    Node raiz=doc.getFirstChild();
    //Obtiene una lista de nodos con todos los nodos hijo del raíz.
    NodeList nodelist=raiz.getChildNodes();
    //Procesa los nodos hijo
    for (int i=0; i<nodelist.getLength(); i++) {
       node = nodelist.item(i);
       if (node.getNodeType()==Node.ELEMENT_NODE){
          //Es un nodo libro
           datos_nodo = procesarLibro(node);
           salida=salida + "\n " + "Publicado en: " + datos_nodo[0];
           salida=salida + "\n " + "El autor es: " + datos_nodo[2];
           salida=salida + "\n " + "El título es: " + datos_nodo[1];
        }    }   return salida;
  }
```

1.4.3 RECORRER UN ÁRBOL DOM

```
protected String[] procesarLibro(Node n){
      String datos[]= new String[3];
      Node ntemp=null;
      int contador=1;
       //Obtiene el valor del primer atributo del nodo (uno en este ejemplo)
      datos[0]=n.getAttributes().item(0).getNodeValue();
       //Obtiene los hijos del Libro (titulo y autor)
      NodeList nodos=n.getChildNodes();
       for (int i=0; i<nodos.getLength(); i++) {
          ntemp = nodos.item(i);
          if(ntemp.getNodeType()==Node.ELEMENT_NODE){
          //IMPORTANTE: para obtener el texto con el título y autor se accede al nodo //TEXT hijo de ntemp y se
          //saca su valor.
               datos[contador] =  ntemp.getChildNodes().item(0).getNodeValue();
               contador++;
          }
     }
     return datos;
  }
```

1.4.4 MODIFICAR Y SERIALIZAR

```
public int annadirDOM ( …)
 try {
     //Se crea un nodo tipo Element  y nombre 'titulo'
      Node ntitulo=doc.createElement("Titulo");
     //Se crea un nodo tipo texto con el título del libro
      Node ntitulo_text=doc.createTextNode(titulo);
     //Se añade el nodo de texto con el título como
     //hijo del elemento Titulo
     ntitulo.appendChild (ntitulo_text);
../..
     //Finalmente, se obtiene el primer nodo del
     //documento y a él se le añade como hijo el nodo
     //libro que ya tiene colgando todos sus
     //hijos y atributos creados antes.
      Node raiz=doc.getChildNodes().item(0);
      raiz.appendChild(nlibro);
      return 0;
 } catch (Exception e){ e.printStackTrace(); return -1;}
  }
```

```
public int guardarDOMcomoFILE()  {
     try{
     //Crea un fichero llamado salida.xml
     File archivo_xml = new File ("salida.xml");
     //Especifica el formato de salida
     OutputFormat format = new
      OutputFormat(doc);
     //Especifica que la salida esté indentada.
     format.setIndenting(true);
    //Escribe el contenido en el FILE
     XMLSerializer serializer =
      new XMLSerializer (new FileOutputStream(
      archivo_xml), format);
     serializer.serialize (doc);
     return 0;
      }   catch (Exception e) {
       return -1;
     }
}
```

Actividad 1.2

1.5 ACCESO A DATOS CON SAX (SIMPLE API FOR XML)

- **SAX** (Simple API for XML) es otra tecnología para poder acceder a XML desde lenguajes de programación.
- **SAX** se usa:
  - SAX ofrece una alternativa para leer documentos XML de manera secuencial.
  - SAX, a diferencia de DOM, no carga el documento en memoria, sino que lo lee directamente desde el fichero.
- **SAX** sigue los siguientes pasos básicos:
  - Se le dice al parser SAX qué fichero quiere que sea leído
  - El documento XML es traducido a eventos.
  - Los eventos generados pueden controlarse con métodos Callbacks.
  - Para implementar los callbacks basta con implementar la interfaz ContentHandler.

1.5.1 ABRIR XML CON SAX DESDE JAVA

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

1.5.2 RECORRER XML CON SAX.

```
public String recorrerSAX (File fXML,ManejadorSAX sh,SAXParser parser ){
 try{
        parser.parse(ficheroXML,sh);
        return sh.cadena_resultado;
        } catch (SAXException e) {
         e.printStackTrace();  return "Error al parsear con SAX";
 } catch (Exception e) {
     e.printStackTrace();  return "Error al parsear con SAX";
 }
}
```

Actividad 1.3

1.6 ACCESO A DATOS CON JAXB (BINDING)

- **JAXB** (no confundir con la interfaz de acceso JAXP) es una librería de (Un)-Marshalling.
- El concepto de Serialización o Marshalling que ya ha sido introducido, es el proceso de almacenar un conjunto de objetos en un fichero.
- Unmarshaling es justo el proceso contrario: convertir en objetos el contenido de un fichero.
- JAXB es capaz de obtener de un esquema XML una estructura de clases que le da soporte en Java.
- En las siguientes secciones se muestra cómo se puede implementar el uso de JAXB en Java.

1.6.1 ¿CÓMO CREAR CLASES JAVA DE ESQUEMAS XML?

- Partiendo de un esquema XML, el proceso de crear la estructura de clases Java que le de soporte es muy sencillo con JAXB.
  - Con JDK 1.7.0 se puede obtener las clases asociadas a un esquema XML con la aplicación xjc.
  - Con el IDE NetBeans 7.1.2 se puede obtener las clases asociadas a un esquema XML.

- Pasos utilizando Netbeans:
  1. Añadir el fichero con el esquema XML al proyecto.
  2. Seleccionar Archivo->Nuevo Fichero para añadir un nuevo elemento al proyecto.
  3. Identificar los campos: la localización del esquema XML sobre el que se crearán las clases Java y el paquete en el que se recogerán las clases nuevas creadas.
  4. Pulsar Finish.

1.6.2 ABRIR XML CON JAXB

```
public int abrir_XML_JAXB(File fichero, Libros misLibros) {
    JAXBContext contexto;
    try {
       //Crea una instancia JAXB
       contexto = JAXBContext.newInstance(Libros.class);
       //Crea un objeto Unmarsheller.
       Unmarshaller u=contexto.createUnmarshaller();
       //Deserializa (unmarshal) el fichero
       misLibros=(Libros) u.unmarshal(fichero);
       return 0;
    } catch (Exception ex) {ex.printStackTrace(); return -1; }
  }
```

1.6.3 RECORRER UN XML DESDE JAXB

```
public String recorrerJAXByMostrar(){
    String datos_nodo[]=null;
    String cadena_resultado="";
    //Crea una lista con objetos de tipo libro.
    List<Libros.Libro> lLibros=misLibros.getLibro();
    //Recorre la lista para sacar los valores.
    for (int i=0; i<lLibros.size(); i++){
       cadena_resultado = cadena_resultado+"\n“ +Publicado en: “
     +lLibros.get(i).getPublicadoEn();
       cadena_resultado= cadena_resultado + "\n" +"El Título es: " + lLibros.get(i).getTitulo();
       cadena_resultado= cadena_resultado + "\n" +"El Autor es: " + lLibros.get(i).getAutor();
       cadena_resultado = cadena_resultado +"\n ----------------------------";
    }
    return cadena_resultado;
  }
```

Actividad 1.4

1.7 PROCESAMIENTO DE XML: XPATH (XML PATH LANGUAGE).

- La alternativa más sencilla de consultar información dentro de un documento XML es mediante el uso de **XPath** (XML Path Language), una especificación de la W3C para la consulta de XML.
- Con XPath se puede seleccionar y hacer referencia a texto, elementos, atributos y cualquier otra información contenida dentro de un fichero XML.
- En el capítulo 5 se presentará Xquery, como el lenguaje más destacado y potente para la consulta de XML en Bases de datos XML nativas.
- XQuery está basado en XPath 2.0

1.7.1 LO BÁSICO DE XPATH

- XPath comienza con la noción ***contexto actual***.
- El contexto actual define el conjunto de nodos sobre los cuales se consultará con expresiones XPath.
- Existen cuatro alternativas para determinar el contexto actual para una consulta XPath:
  - (./) usa el nodo en el que se encuentra actualmente como contexto actual.
  - (/) usa la raíz del documento XML como contexto actual.
  - (.//) usa la jerarquía completa del documento XML desde el nodo actual como contexto actual.
  - (//) usa el documento completo como contexto actual.

1.7.1 LO BÁSICO DE XPATH

```
▼<Libros xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="LibrosEsquema.xsd">
  ▼<Libro publicado_en="1840">
     <Titulo>El Capote</Titulo>
     <Autor>Nikolai Gogol</Autor>
   </Libro>
  ▼<Libro publicado_en="2008">
     <Titulo>El Sanador de Caballos</Titulo>
     <Autor>Gonzalo Giner</Autor>
   </Libro>
  ▼<Libro publicado_en="1981">
     <Titulo>El Nombre de la Rosa</Titulo>
     <Autor>Umberto Eco</Autor>
   </Libro>
 </Libros>
```

- Ejemplos:
  - /Libros/Libro/Autor
  - //Autor
  - /Libros/*/Autor
  - /Libros/Libro/@publicado_en
  - //@ publicado_en
  - /Libros/Libro/Titulo[.='El Capote']
  - /Libros/Libro[./Titulo='El Capote']
  - /Libros/Libro[./@publicado_en>1900]

1.7.2 XPATH DESDE JAVA

- Desde Java existen librerías que permiten la ejecución de consultas XPath sobre documento XML.
- Las clases necesarias para ejecutar consultas XPath son:
  - XPathFactory disponible en el paquete javax.xml.xpath.*: esta clase contiene un método compile() que comprueba si la sintaxis de una consulta XPath es correcta y crea un expresión XPath (XPathExpression).
  - XPathExpression disponible también en el paquete javax.xml.xpath.*: esta clase contiene un método evaluate() que ejecuta un XPath.
  - DocumentBuilderFactory disponible en el paquete javax.xml.parsers.*, y Document del paquete org.w3c.xml.*.

Actividad 1.5

1.7.2 XPATH DESDE JAVA

```
public int EjecutaXPath() {
 try {
    //Crea el objeto XPathFactory
    xpath = XPathFactory.newInstance().newXPath();
    //Crea un objeto DocumentBuilderFactory para el DOM (JAXP)
     DocumentBuilderFactory factory =  DocumentBuilderFactory.newInstance();
    //Crear un árbol DOM (parsear) con el archive LibrosXML.xml
     Document XMLDoc = factory.newDocumentBuilder().parse(new InputSource (
new FileInputStream("LibrosXML.xml")));
     //Crea un XPathExpression con la consulta deseada
     exp = xpath.compile(“/Libros/*/Autor”);
```

Actividad 1.5

1.7.2 XPATH DESDE JAVA

```
 //Ejecuta la consulta indicando que se ejecute sobre el DOM y que devolverá
 //el resultado como una lista de nodos.
Object result= exp.evaluate(XMLDoc, XPathConstants.NODESET);
 NodeList nodeList = (NodeList) result;
 //Ahora recorre la lista para sacar los resultados
 for (int i = 0; i < nodeList.getLength(); i++) {
   salida = salida + "\n" +
  nodeList.item(i).getChildNodes().item(0).getNodeValue();
 }
   System.out.println(salida);
   return 0;
  }  catch (Exception ex) { System.out.println("Error: " + ex.toString());
      return -1;  }
}
```

Actividad 1.5

1.8 CONCLUSIONES Y PROPUESTAS PARA AMPLIAR

- En este capítulo se han mostrado diferentes formas de acceso a ficheros. Lejos de pretender profundizar en todas las posibilidades de cada acceso, lo que se ha buscado ha sido dar una visión global sobre el tratamiento de ficheros con Java.
- El lector interesado en profundizar en las tecnologías expuestas:
  1. Trabajar más en profundidad los flujos para el tratamiento de archivos en Java.
  2. Conocer otros modos de acceso a documentos XML, como jDOM.

1.9 RESUMEN DEL CAPÍTULO

- En este capítulo se ha abordado el acceso a ficheros como manera más básica de hacer persistentes datos de una aplicación. El capítulo tiene dos partes diferenciadas:
  - La primera parte abarca el acceso a ficheros con las clases que ofrece java.io. Esta alternativa es la de más bajo nivel de todas las soluciones que se dan en este y en el resto de capítulos para el acceso a datos sin embargo, sí es cierto que conocer bien el acceso a fichero (tipos y modos) es obligado cuando se trabaja con persistencia.
  - La segunda parte abarca el acceso a un tipo concreto de fichero de texto secuencial: XML. Que el lector conozca y maneje las diferentes alternativas de acceso a ficheros XML desde Java ha sido el objetivo perseguido. Los conocimientos en esta tecnología serán muy útiles para entender algunos de los siguientes capítulos de este libro, sobre todo lo relacionado con el marshalling, XPath y DOM.

ACCESO A DATOS

CAPÍTULO 1

Manejo de ficheros

José Eduardo Córcoles
Francisco Montero

Ra-Ma®