Emmagatzemament amb documents XML

1. Introducció

En primer lloc, com a classe de treball base per als nostres exemples, anem a definir la classe `Modul` , un *Bean* que implementarà la interfície `Serializable` , per si en un futur volem escriure a/desde fitxers:

```
class Modul implements Serializable{
    String nom;
    int hores;
    double nota;

    public Modul(){
        // Constructor buit
    }

    public Modul(String nom, int hores, double nota){
        this.nom=nom;
        this.hores=hores;
        this.nota=nota;
    }

    public String getModul() {return this.nom;}
    public int getHores() {return this.hores;}
    public double getNota() {return this.nota;}
}
```

També tenim aquesta estructura per a poder generar els mòduls

```
public class Moduls2{

    // Definim els vectors per inicialitzar dades
    String[] moduls={"Accés a Dades", "Programació de serveis i processos", "Desenvolup
    int[]  hores={6, 3, 6, 5, 5, 3};
    double[] notes={8.45, 9.0, 8.0, 7.34, 8.2, 7.4};
```

Quan volem guardar dades que puguen ser llegides per diferents aplicacions i plataformes, el més adient és fer ús de formats estàndards d'emmagatzemament, que múltiples aplicacions puguen entendre. Un cas molt concret són els llenguatges de marques, i el més conegut és l'estàndard `XML` (*eXtensible Markup Language*).

Amb els documents XML estructurem la informació intercalant marques o etiquetes entre la informació. Aquestes etiquetes tenen un principi i un final, i poden aniar-se dins d'altres, així com

contenir informació textual. Com que la informació serà textual, no tenim el problema de la diferent representació de dades, ja que qualsevol dada, siga del tipus que siga es passarà a text. Per tal d'evitar també el problema dels diferents sistemes de codificació de text, XML permet incloure a la capçalera del document la codificació que s'ha utilitzat per guardar-lo.

La forma d'emmagatzemar la informació en XML, de forma jeràrquica s'assembla molt a la forma que ho fan els objectes en una aplicació, de manera que aquests poden traduir-se d'una forma relativament còmoda a un document XML.

Ací tenim les dades que volem emmagatzemar:

Mòdul	Hores	Qualificació
Accés a Dades	6	8.45
Programació de serveis i processos	3	9.0
Desenvolupament d'interfícies	6	8.0
Programació Multimèdia i dispositius mòbils	5	7.34
Sistemes de Gestió Empresarial	5	8.2
Empresa i iniciativa emprenedora	3	7.4

Aquesta es podría expressar de diferents formes amb XML:

- **Utilitzant només etiquetes**:

```
<curs>
    <modul>
      <nom>Accés a Dades</nom>
      <hores>6</hores>
      <qualificacio>8.45</qualificacio>
    </modul>
    <modul>
      <nom>Programació de serveis i processos</nom>
      <hores>3</hores>
      <qualificacio>9.0</qualificacio>
    </modul>
    <modul>
      <nom>Desenvolupament d'interfícies</nom>
      <hores>6</hores>
      <qualificacio>8.0</qualificacio>
    </modul>
    <modul>
      <nom>Programació Multimèdia i dispositiud mòbils</nom>
      <hores>5</hores>
      <qualificacio>7.34</qualificacio>
    </modul>
    <modul>
      <nom>Sistemes de Gestió Empresarial</nom>
      <hores>5</hores>
      <qualificacio>8.2</qualificacio>
    </modul>
    <modul>
      <nom>Empresa i iniciativa emprenedora</nom>
      <hores>3</hores>
      <qualificacio>7.4</qualificacio>
    </modul>
</curs>
```

- **Utilitzant etiquetes i atributs**:

```
<curs>
    <modul nom="Accés a Dades" hores="6" qualificacio="8.45" >
    <modul nom="Programació de serveis i processos" "hores"=3 qualificacio="9.0" >
    <modul nom ="Desenvolupament d'interfícies" hores="6" qualificacio="8.0" >
    </modul>
    <modul nom="Programació Multimèdia i dispositiud mòbils" hores="5" qualificacio="7,
    <modul nom="Sistemes de Gestió Empresarial" hores="5" "qualificacio"=8.2 />
    <modul nom="Empresa i iniciativa emprenedora"  hores="3" qualificacio="7.4" />
    </modul>
</curs>
```

2. Analitzadors XML

Un parser o analitzador XML és una classe que permet analitzar un fitxer XML i extreure'n la informació d'ell, relacionant-la segons la seua posició en la jerarquia.

Els analitzadors, segons la seua forma de funcionar poden ser:

- **Analitzadors seqüencials o sintàctics,** que van extraient el contingut segons es van descobrint les etiquetes d'obertura i tancament. Són molt ràpids, però tenen el problema que cal llegir tot el document per tal d'accedir a una part concreta. En Java existeix l'analitzador `SAX` (*Simple API for XML*) com a analitzador seqüencial.
- **Analitzadors jeràrquics,** que solen ser els més utilitzats, i que guarden totes les dades del document XML en memòria, en forma d'estructura jerarquitzada (`DOM` o *Model d'Objectes del Document*, sent els preferits per a aplicacions que hagen de llegir les dades de forma més contínua.

3. El Model d'Objectes del Document (`DOM`)

El DOM (Document Object Model) és l'estructura especificada pel `W3C` on s'emmagatzema la informació dels documents XML. El DOM ha estat lligat sobretot al món web, amb HTML i Javascript com a principals impulsors. En Java, el DOM s'implementa fent ús d'interfícies.

La interfície principal del DOM en Java és `Document` , i representa **tot el document XML**. Com que es tracta d'una interfície, aquesta podrà implementar-se en diverses classes.

> ***Recordeu...***
>
> Una interfície és una espècie de plantilla per construïr classes, i es compon generalment d'un conjunt de declaracions de capçaleres de mètodes, sense implementar, que especifiquen la forma de comportar-se per una o diverses classes. A més, una classe pot implementar una o més interfícies. En aquest cas, la classe haurà de declarar i definir tots els mètodes de cadascuna de les interfícies, o declarar-se com a classe abstracta.
>
> Tampoc hi ha que confondre una interfície amb una classe abstracta, ja que hi ha algunes diferències, com que la interfície té tots els mètodes abstractes; no pot declarar variables d'instància; una classe pot implementar vàries interfícies, però no heretar de vàries superclasses; i la interfície no té per què pertànyer a cap jerarquía, de manare que classes que no tinguen cap relació d'herència poden implementar la mateixa interfície.

A banda de `Document` , el W3C també defineix la classe abstracta `DocumentBuilder` , que permet crear el DOM a partir de l'XML. A més, s'especifica la classe `DocumentBuilderFactory` , que ens permet *fabricar* `DocumentBuilder` s, ja que al ser abstracta no es pot instanciar directament.

Cal dir, com a advertència, que Java ofereix moltes llibreries des d'on importar Document. Les llibreríes que anem a utilitzar per parsejar XMLs seran:

- La llibrería `java.xml.parsers.*` , que oferiran les classes `DocumentBuilderFactory` i `DocumentBuilder` , i
- La llibrería `org.w3c.dom.*` per a la classe `Document` .

3.1. `DocumentBuilder` i `DocumentBuilderFactory`

Com hem comentat, `DocumentBuilder` defineix una API per obtenir instàncies del DOM a partir d'un document XML. Per tal d'obtindre una instància de la classe, s'ha de recórrer a la factoría `DocumentBuilderFactory` , i concretament al mètode `newDocumentBuilder()` :

```
DocumentBuilderFactory.newDocumentBuilder();
```

Per altra banda, per tal de llegir i interpretar el document XML, la classe `DocumentBuilderFactory` proporciona el mètode `parse` , que parseja un XML indicat per un `File` , i retorna un objecte `Document` .

Veiem-ho tot amb un exemple. Anem a treballar emmagatzemant dades sobre els mòduls del curs amb XML. Per a això, s'ha creat una classe anomenada `XMLLib` per tal de treballar amb documents XML. Després explicarem el funcionament del paquet en concret, però ara anem a centrar-nos en obrir documents XML i la seua posterior lectura.

El següent mètode d'aquesta classe ens servirà, per tal d'obrir un document XML, parsejar-lo i retornar el DOM generat en un `Document` :

```
public Document ObreXML(String nom) throws IOException,SAXException, ParserConfiguratic

    // Creem una instància de DocumentBuilderFactory
    DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
    // Amb la instància de DocumentBuilderFactory creem un DocumentBuilder
    DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
    //I utilitzem el mètode "parse" de DocumentBuilder per obtindre el document
    Document doc = dBuilder.parse(new File(nom));

    return doc;
}
```

Cal dir que la funció anterior podría haver-se simplificat sense utilitzar les declaracions intermitges:

```
public Document ObreXML(String nom) throws IOException,SAXException, ParserConfiguratic
     return DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(nom);
}
```

Per altra banda, la classe `DocumentBuilder` ens permet també crear un DOM nou, amb el mètode `newDocument()` . Açò ens servirà després per tal d'emmagatzemar els documents XML. El primer

que haurem de fer és crear un DOM nou amb `newDocument()`, anar afegint els elements i després emmagatzemar-lo. En apartats posteriors, veurem com fer tot açò. De moment anem a centrar-nos en la interpretació i lectura del DOM.

Podeu trobar més informació sobre les classes `DocumentBuilder` i `DocumentBuilderFactory` a l'API de l'OpenJDK:

- http://cr.openjdk.java.net/~iris/se/11/latestSpec/api/java.xml/javax/xml/parsers/DocumentBuilder.html
- http://cr.openjdk.java.net/~iris/se/11/latestSpec/api/java.xml/javax/xml/parsers/DocumentBuilderFactory.html

3.2. Classes i mètodes del DOM

Fins ara hem vist com obrir i parsejar un document XML amb `DocumentBuilder` per crear un objecte de tipus `Docuement`. En aquest apartat veurem com treballar amb aquest document per tal d'accedir als diferents elements.

Com sabem, el DOM té una estructura jeràrquica, formada per nodes. Els diferents tipus de **nodes** que ens podem trobar són:

- `Document`, que és el node principal i representa tot l'XML,
- `Element`, que representa les diferents etiquetes (inclòs l'arrel),
- `TextElement`, que representa el contingut d'una etiqueta de text,
- `Attribute`, que representa els atributs.

Totes aquestes interfícies deriven de la interfície `Node`, pel que heretaran els seus atributs i mètodes, i a més, aportaran atributs i mètodes propis.

Veiem els mètodes més importants de cada interfície:

- **Mètodes de la interfície `Node`**:

Mètode	Descripció
Mètodes relacionats amb l'obtenció d'informació	
`String getNodeName`	Obté el nom del node actual
`short getNodeType()`	Obté el tipus del node (ELEMENT_NODE, ATTRIBUTE_NODE, TEXT_NODE...
`String getNodeValue()`	Obté el valor del node
`NodeList getChildNodes()`	Obté una llista amb els nodes fills

Mètode	Descripció
`Node getFirstChild()`	Retorna el primer fill
`Node getLastChild()`	Retorna l'últim fill
`NamedNodeMap getAttributes()`	Retorna una llista amb els atributs del node
`Node getParentNode()`	Retorna el node pare
`String getTextContent()`	Retorna el text contingut en l'element i el dels seus descendents
`boolean hasChildNodes()`	Retorna cert si el node té algun fill
`boolean hasAttributes()`	Retorna cert si el node té algun atribut
Mètodes relacionats amb l'escriptura	
`Node appendChild(Node node)`	Afig un node nou com a últim node dels fills
`void removeChild(Node node)`	Elimina el node indicat dels nodes fills

- **Mètodes de la interfície `Element`**:

Mètode	Descripció
Mètodes relacionats amb l'obtenció d'informació	
`String getAttribute(String nom)`	Retorna el valor de l'atribut indicat pel nom
`NodeList getElementsByTagName(String nom)`	Retorna una llista dels nodes descendents que coincideixen amb el nom indicat
`boolean hasAttribute(String nom)`	Retorna cert si l'element té l'atribut indicat
Mètodes relacionats amb l'escriptura	
`void setAttribute(String nom,String valor)`	Afig un atribut a l'element, amb el nom i el valor indicats
`void removeAttribute(String nom)`	Elimina l'atribut indicat pel nom

- **Mètodes de la interfície `Document`**:

Mètode	Descripció
Mètodes relacionats amb l'obtenció d'informació	
`Element getDocumentElement()`	Retorna l'element arrel del document
`NodeList getElementsByTagName(String nom)`	Retorna una llista dels nodes descendents que coincideixen amb el nom indicat
Mètodes relacionats amb l'escriptura	
`Element createElement(String nom)`	Crea un nou element amb el nom indicat
`Text createTextNode(String text)`	Crea un nou element de text
`Node appendChild(Node node)`	Afig un nou node fill

Els objectes de tipus `NodeList`, que representen una llista de nodes, ofereix el mètode `item` per accedir als diferents nodes de la llista, indicant el seu ordre.

3.3. Lectura de documents XML

Veiem tot el comentat a l'apartat anterior amb el nostre cas pràctic sobre els mòduls. En la nostra classe `XMLLib` hem creat un mètode `MostraXML`, que rep un element de tipus `Document` (tal com ens el retorna `ObreXML`) i mostrarà el seu contingut per pantalla:

```
public void MostraXML(Document doc) throws IOException
```

Per tal de començar a llegir el document, el primer que haurem de fer és obtenir l'arrel del document, amb `getDocumentElement()`, que retorna un objecte de tipus `Element`:

```
Element arrel = doc.getDocumentElement();
```

Amb aquest element arrel, ja podríem mostrar tot el contingut amb `getTextContent()`:

```
System.out.println(arrel.getTextContent());
```

Però el que ens interessa és recórrer tot el DOM i accedir als seus elements. Per a això, a partir d'aquest element arrel, seguirem els següents passos:

1. Buscarem totes les etiquetes `<modul>` amb `getElementsByTagName`. Aquest mètode ens torna una llista de nodes (objecte de tipus `NodeList`).
2. Caldrà recórrer la llista de nodes (`NodeList`) per accedir a cada element. Per a això cal fer ús del mètode `item()`, que ens retornarà un element de tipus `Node`, i que caldrà convertir a

`Element` de forma explícita.

3. Per a cada element, accedirem al nom del node per mostrar el nom i l'ordre, amb `getNodeName()`
4. Busquem les diferents etiquetes que es troben dins de cada mòdul ('nom', 'hores' i 'qualificacio') amb `getElementsByTagName` .
5. El pas anterior ens haurà donat de nou un `NodeList` per a cada tipus d'etiqueta. Com que només tindrem un element, només cal accedir a l' `item(0)` .
6. Cal tindre en compte que amb l'anterior tindrem la primera (i única) etiqueta 'nom', 'hores' o 'qualificacio' del mòdul, però encara no estem en el contingut, ja que aquest és un element de tipus `TEXT_NODE` . Per accedir a ell, haurem d'accedir al primer fill de l'etiqueta (`getFirstChild()`) i obtenir el seu valor amb `getNodeValue`

```
    // Obtindrem una llista de nodes (Pas 1)
    NodeList moduls = arrel.getElementsByTagName("modul");

    // Per a cada node (Pas 2)
    for (int i = 0; i < moduls.getLength(); i++) {
        Element el = (Element) moduls.item(i);

        // Mostra el nom del node (Pas 3)
        System.out.println(el.getNodeName() + " " + (i + 1));

        // I mostrem el valor de les diferents etiquetes (Passos 4, 5 i 6)

        System.out.println("Nom: " + el.getElementsByTagName("nom").item(0).getFirstChi

        System.out.println("Hores: " + el.getElementsByTagName("hores").item(0).getFirs
        System.out.println("Qualificació: " + el.getElementsByTagName("qualificacio").i
        System.out.println();

    }

}
```

3.4. Escriptura de documents XML

Anem ara a la part d'escriptura dels documents XML. Pera això, partirem d'un fitxer que ja conté la informació en format binari dels mòduls, el llegirem, i importarem la seua informació a format XML. La funció tindrà la següent capçalera:

```
public void ImportaObj(String fitxer) throws IOException,ParserConfigurationException,
```

Com veiem, rep un string, que farà referència al fitxer en format binari que anem a tractar.

El primer que haurem de fer és llegir el fitxer d'objectes mitjançant un `ObjectInputStream` :

```
ObjectInputStream f = new ObjectInputStream(new FileInputStream(fitxer));
```

I crear un `Document` Xml buit, ajudant-nos de les classe `DocumentBuilder` i `DocumentBuilderFactory`:

```
Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
```

Una vegada tenim el document buit, creem l'element arrel (curs), i l'afegim al document:

```
Element arrel = doc.createElement("curs");
doc.appendChild(arrel);
```

Recordem que accedirem al fitxer d'objectes, pel que haurem de conéixer exactament com és la classe que volem llegir, i accedir als mètodes corresponents per tal d'obtenir-ne informació.

Per a això, en primer lloc, cal definir un objecte de tipus mòdul:

```
Modul m;
```

I anirem llegint el fitxer d'objectes amb el mètode readObject de File:

```
m = (Modul) f.readObject();
```

Una vegada hem llegit un objecte, crearem l'etiqueta que engloba a cadascun d'ells: l'etiqueta mòdul:

```
Element modul = doc.createElement("modul");
```

I dins d'ella, i a mesura que extraiem les diferents propietats de l'objecte `Modul`, anirem creant nodes fills i afegint-los al mòdul. Per exemple, per al nom del mòdul:

```
Element nom = doc.createElement("nom");
nom.appendChild(doc.createTextNode(m.getModul()));
modul.appendChild(nom);
```

Com veiem, hem creat un objecte de tipus `Element` amb l'etiqueta 'nom', i li hem afegit com a fill un node de tipus text (`TEXT_NODE`), que hem extret directament de l'objecte `Modul m` amb la seua funció pròpia `getModul()` . A més, hem afegit aquesta etiqueta a l'etiqueta `<modul>` , amb `appendChild` .

Haurem de fer el mateix per a les hores de cada mòdul i la qualificació, però per a això, haurem de tindre en compte que els mètodes `getHores` i `getNota` no tornen un `String` , sinò un `int` i un

`double` , pel que caldrà convertir-los a text:

```
Element hores = doc.createElement("hores");
hores.appendChild(doc.createTextNode(Integer.toString(m.getHores())));
modul.appendChild(hores);

Element qualificacio = doc.createElement("qualificacio");
qualificacio.appendChild(doc.createTextNode(Double.toString(m.getNota())));
modul.appendChild(qualificacio);
```

Tot aquest procediment, el posarem dins un bucle que llisca tot el fitxer d'objectes. Una vegada tinguem llegit cadascun dels mòduls, els haurem d'afegir a l'element arrel amb:

```
arrel.appendChild(modul);
```

I ja tindrem en arrel el nostre document XML. Ara ens quedaría convertir aquest objecte de tipus `Element` en una cadena de text per tal de poder escriure'l al disc. Per això farem ús de la utilitat `Transformer` .

Transformer

Java ens oferix la utilitat `Transformer` per convertir informació entre diferents formats jeràrquics, com per exemple, l'objecte `Document` que conté el DOM del nostre XML, a un fitxer de text XML.

La classe `Transformer` , igual que `DocumentBuilder` és també una classe abstracta, pel que també requereix d'un *factory*, per poder-se instanciar.

La classe `Transformer` treballa amb dos tipus adaptadors. Els adaptadors són classes que fan compatibles jerarquies diferents. Aquests adaptadors són `Source` i `Result` . Les classes que implementen aquests adaptadors s'encarregaran de fer compatibles els diferents tipus de contenidors al que requerisca la classe `Transformer` . Així doncs, i per clarificar, disposem de les classes `DOMSource` , `SAXSource` o `StreamSource` , que són adaptadors del contenidor de la font d'informació per a DOM, SAX o Stream; i de `DOMResult` , `SAXResult` i `StreamResult` com a adaptadors equivalents al contenidor destí.

Per al nostre cas, com que tenim un DOM i el volem convertir a Stream, necessitarem un `DomSource` i un `StreamResult` . Veiem el codi necessari per fer açò:

```
Transformer trans = TransformerFactory.newInstance().newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(new FileOutputStream(fitxer+".xml"));
```

El primer que hem fet és crear un objecte de tipus `Transformer` amb el mètode `newTransformer()` d'una instància (`newInstance()`) de la *factoría* de Transformers `TransformerFactory` .

Després hem definit l'origen (`source`) i el resultat (`result`) per a la transformació, sent l'origen un `DomSource` creat a partir del `doc` que conté el nostre document, i el resultat un `StreamResult` , que escriurà l'stream en disc a través d'un `FileOutputStream` .

I finalment, fem la transformació d'un element a altre, el que automàticament generarà el fitxer XML d'eixida:

```
trans.transform(source, result);
```

4. Binding XML

La tècnica del Binding consisteix a generar classes Java amb formats concrets, com per exemple XML, de manera que cada etiqueta o atribut d'XML es correspon amb una propietat de certa classe. Aquesta correspondència s'anomena *mapat*.

En Java existeixen diferents llibreries per al mapat o binding: JAXB, JuBX, XMLBinding, etc. JAXB (Java Architecture for XML Binding) és una potent biblioteca que s'ha incorporat en l'estàndard des de JDK 6, però s'ha suprimit en la versió 11, i es suggereix que s'incloga com a paquet de tercers. JAXB fa ús d'anotacions per aconseguir la informació necessària per mapar el binding. Les anotacions són classes especials de Java que permeten associar informació i funcionalitat als objectes, sense interferir en l'estructura del model de dades. Les anotacions poden associar-se a un paquet, a una classe, a un atribut o a un paràmetre, i es declaren amb el símbol @ davant del nom de l'anotació. Quan el compilador detecta una anotació, crea una instància i la injecta dins l'element afectat, sense interferir en la classe en sí. Quan una aplicació necessita de la informació de les anotacions, poden obtenir la instància injectada.

Per exemple, en la classe Mòdul que teníem definida, faríem ús de l'anotació `@XmlRootElement` per indicar l'element arrel del mòdul, i les anotacions `@XmlElement` , per indicar que els setters de la classe escriuran també elements XML.

```
@XmlRootElement
class Modul {

        String nom;
        int hores;
        double nota;

        public String getNom() { return nom; }
        @XmlElement
        public void setNom(String nom) { this.nom = nom; }


        public int getHores() { return hores; }
        @XmlElement
        public void setHores(int hores) { this.hores = hores; }

        public double getNota() { return nota; }
        @XmlElement
        public void setNota(double nota) { this.nota = nota;}

}
```

Amb açò tindríem només la classe amb anotacions preparada per guardar un mòdul com a document XML. Per guardar tota la jerarquía hauriem de crear la classe Curs, que contindria un ArrayList de mòduls.

Pel que fa a aquest curs, no aprofundirem més en aquesta tècnica, ja que per als nostres objectius, ens és suficient amb el parseig d'XML que hem vist en apartats anteriors.