- El tipus de dada que es desa a les posicions de la darrera dimensió serà el mateix per a tots. Per exemple, no es poden tenir reals i enters barrejats.

En llenguatge Java, per afegir noves dimensions per sobre de la segona, cal anar afegint claus `[]` addicionals a la declaració, una per a cada dimensió.

```
paraulaClauTipus[]...[] identificadorVariable = new paraulaClauTipus[midaDim1
    ]...[midaDimN];
```

Per exemple, per declarar amb valors per defecte un *array* de quatre dimensions, la primera de mida 5, la segona de 10, la tercera de 4 i la quarta de 20, caldria la instrucció:

```
...
  int[][][][] arrayQuatreDimensions = new int[5][10][4][20];
  ...
```

L'accés seria anàleg als *arrays* bidimensionals, però usant quatre índexs envoltats per parells de claus.

```
...
  int valor = arrayQuatreDimensions[2][0][1][20];
  ...
```

En qualsevol cas, es tracta d'estructures complexes que no se solen usar, excepte en casos molt específics.

1.4 Tractament de fitxers XML

Una de les problemàtiques més importants amb què haureu de tractar quan utilitzeu fitxers per desar o recuperar les dades dels vostres programes és escollir de quina manera s'estructuren dins el fitxer. En quin format i en quin ordre estan escrites les dades, de manera que sempre que es llegeixin o s'escriguin és imprescindible seguir exactament aquest format. En cas contrari, en el millor dels casos, es produirà un error, i en el pitjor, les dades llegides o escrites seran totalment incorrectes. Això és un problema ja que, a menys que es disposi de documentació detallada, és molt difícil que, només amb un fitxer donat, es pugui esbrinar amb detall quina mena de dades conté. Per intentar pal·liar aquest problema, va sorgir el llenguatge XML.

XML són les inicials en anglès d'**eXtensible Markup Language** o llenguatge de marques extensible.

Aquests materials han estan generats usant un llenguatge de marcat.

Un llenguatge de marques és un conjunt de paraules i de símbols per descriure la identitat o la funció dels components d'un document (per exemple, "aquest és un paràgraf", "aquesta és una llista", "això és el títol d'aquesta figura"...). Aquestes marques permeten facilitar el processament de les dades a tractar dins el fitxer.

Per exemple, transformar el document a diferents formats de la sortida de pantalla, mitjançant un full d'estil.

Alguns llenguatges de marques (especialment els utilitzats en processadors de text) només descriuen les aparences al seu lloc ("això és cursiva", "això és negreta"...), de manera que aquests sistemes només es poden utilitzar per a la visualització, i per poca cosa més. Un dels llenguatges més populars de marcat, d'acord a aquesta definició, és HTML, el llenguatge emprat per codificar pàgines web. XML permet dur aquesta idea més enllà de la visualització i facilitar la comprensió de qualsevol document de dades en els vostres programes, independentment de la seva funció, no només per indicar com s'han de representar visualment.

Per veure clarament aquest fet, el millor és mostrar ja directament un exemple d'un mateix conjunt de dades relatives a un objecte estructurades sense XML i amb XML, sense ni tan sols haver presentat abans com s'estructura un document XML. D'aquesta manera, sense cap coneixement previ, encara queden més patents els seus objectius i quins avantatges presenta. En el primer cas, s'ha triat representar els valors de l'objecte com a text separat per comes:

```
Sense XML:
  Producte 1, 1, 10.0, 4, true
Amb XML:
<Producte id="1" aLaVenda="true">
  <Nom>Producte 1</Nom>
  <Preu>10.0</Preu>
  <Estoc>4</Estoc>
</Producte>
```

Atès el primer cas, sense tenir cap documentació prèvia, ¿és possible esbrinar a què es refereixen exactament els valors "4" o "1"? Quin és el nom de la classe a què pertany l'objecte? La veritat és que caldria tenir molta imaginació, per no dir directament que és impossible. XML es basa en la idea que a cada dada dins el document se li afegeix una informació extra (les marques) que, si s'ha triat estratègicament, permet saber exactament el seu propòsit. El document generat és autoexplicatiu.

Com es pot veure de l'exemple anterior, XML i HTML són molt semblants. Si mai heu vist el codi font d'una pàgina web, XML us pot resultar familiar, encara que sigui el primer cop que veieu un document XML. Això no és casualitat, ja que ambdós són variacions d'un llenguatge de marcat més genèric anomenat SGML (*Standard Generalized Markup Language*, o llenguatge de marques estàndard generalitzat). Ara bé, entre HTML i XML hi ha algunes diferències importants. Entre les més destacades hi ha les següents:

- HTML serveix únicament per indicar com representar visualment unes dades (normalment a navegadors web). XML serveix per a qualsevol tipus de processament de dades.
- En HTML la llista de marques possibles ja està prefixada. En XML no; vosaltres us podeu inventar les que vulgueu.
- HTML sol tenir una sintaxi més laxa, on alguns errors en el format de les marques no afecten el seu processament. XML és molt més estricte.

1.4.1 Estructura d'un document XML

Un document XML emmagatzema un conjunt de dades d'acord a un seguit de marques que permeten identificar de manera senzilla el propòsit d'aquestes dades. Ara bé, les marques són definides pel seu creador, d'acord a la mena d'informació que vol emmagatzemar. Cada aplicació usarà documents XML amb marques diferents segons el que li resulti més convenient. Per tant, serà vostra la tasca de definir-les, usant noms adients que permetin identificar amb claredat la funció de les dades emmagatzemades.

Tot i que documents d'aplicacions diferents, com és d'esperar, tindran un format diferent, la seva estructura serà semblant, ja que tots han de seguir l'estructura i les regles bàsiques que marca el llenguatge XML. Des d'un punt de vista genèric, tot document XML es correspon a una estructura jeràrquica (en forma d'arbre), on es parteix d'un node arrel i d'aquest sorgeixen nodes fills que es van propagant fins arribar a nodes fulla, o els darrers dins la jerarquia. Els tipus de nodes més importants són els següents:

- De text, que contenen la informació final a emmagatzemar. La majoria de nodes fulla solen ser d'aquest tipus. Es representen dins el document directament usant una cadena de text que representa les dades en qüestió.
- Elements, que defineixen agrupacions lògiques de continguts caracteritzades per un nom. Es representen usant una parella de marques d'inici i de fi, amb format `<NomElement>...</NomElement>`.
- Atributs, parelles de nom i de valor, sempre associades a un element concret. S'escriuen dins la marca d'inici de l'element, usant el format `nom=''valor''`. Si, en un element, n'hi ha més d'un, se separen amb un espai en blanc.
- Sovint, a l'inici del document, s'hi afegeix pròleg, en realitat opcional, que dóna certa informació sobre el format del fitxer (versió, codificació...).

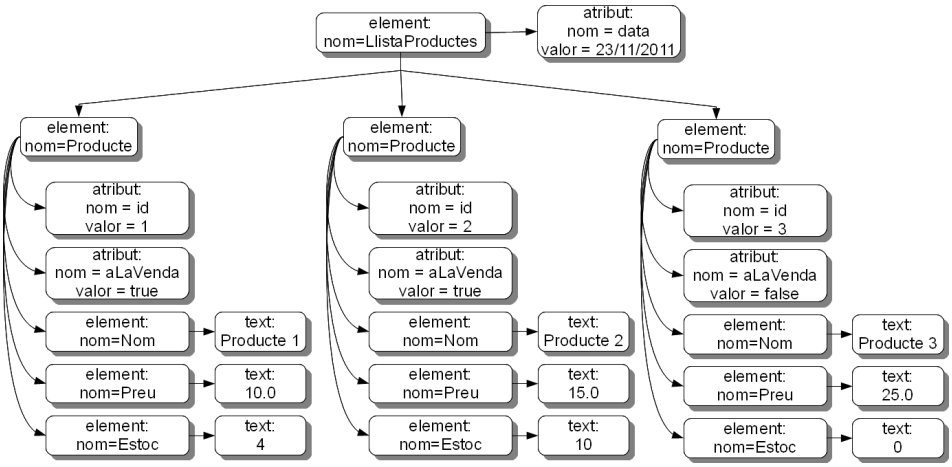
Els nodes d'un document XML han de seguir unes normes en la seva estructura perquè es consideri que el document és correcte (o ben format). Primer de tot, només hi pot haver un únic element arrel. D'altra banda, atès qualsevol element, dins seu, entre el símbol d'inici i final, només hi poden haver, o altres elements complets, ja sigui un o molts, o text. De fet, aquesta inclusió és el que defineix la relació jeràrquica entre elements (pare = qui inclou, fills = els que estan inclosos). Finalment, les marques d'inici i de final dels diferents elements sempre han d'estar correctament niuades. Per exemple, el següent cas seria incorrecte d'acord a aquesta darrera norma:

```
1 <Producte><Nom></Producte></Nom>
```

Bàsicament, les normes es poden resumir així: que tots els elements han de conformar sempre una estructura jeràrquica correcta. La figura 1.7 mostra la representació jeràrquica del document XML que es mostra tot seguit.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<LlistaProductes data="23/11/2011">
<Producte id="1" aLaVenda="true">
  <Nom>Producte 1</Nom>
  <Preu>10.0</Preu>
  <Estoc>4</Estoc>
</Producte>
<Producte id="2" aLaVenda="true">
  <Nom>Producte 2</Nom>
  <Preu>15.0</Preu>
  <Estoc>10</Estoc>
</Producte>
<Producte id="3" aLaVenda="false">
  <Nom>Producte 3</Nom>
  <Preu>25.0</Preu>
  <Estoc>0</Estoc>
</Producte>
</LlistaProductes>
```

Figura 1.7. Estructura jeràrquica dels nodes d'un document XML



Algú una mica observador pot adonar-se que, tot plegat, és molt semblant a un mapa d'objectes. I és que XML i orientació a objectes són dos temes que, tot i que en realitat de manera casual, semblen fets l'un per l'altre.

1.4.2 Lectura de fitxers XML

Atès que l'estructura i la sintaxi d'un document XML és sempre la mateixa exactament, només varia el nom i els valors dels nodes, és possible dissenyar mecanismes genèrics per llegir-lo. El codi no ha de ser *ad hoc* depenent de cada tipus d'aplicació. Això és un gran avantatge a l'hora de fer programes.

Existeixen diferents mètodes per fer el tractament de documents XML. Aquest apartat se centra només en l'anomenat DOM (*Document Object Model*, o model d'objectes de document), que sol ser el més popular i senzill d'usar quan la quantitat de dades contingudes al document és moderada. Per tractar documents molt grans és preferible usar altres sistemes, com per exemple l'anomenat SAX (*Simple API for XML*, o API simple per a XML).

La senzillesa del DOM recau en el fet que es basa a llegir qualsevol document XML i processar-lo per convertir-lo en una estructura d'objectes a memòria, el model DOM, idèntica a l'estructura del document original, exactament tal com s'ha presentat. Diferents objectes de tipus `Node` estan entrellaçats entre si, contenint un seguit de mètodes que permeten consultar els seus noms, valors o altres nodes dins l'estructura, de manera que és possible fer-hi recorreguts. El procés de creació d'aquest model sempre és igual independentment del contingut del fitxer XML, pel que es pot generalitzar. Ara bé, un cop es disposa del model, ja és tasca vostra fer recorreguts i indicar què cal cercar, ja que això dependrà de cada aplicació concreta.

Creació i invocació del parser DOM Java

Dins les darreres versions, Java ja incorpora un seguit de classes auxiliars que permeten processar un document XML automàticament usant el model DOM. No és necessari que vosaltres tracteu el text que hi ha dins un fitxer XML. El resultat d'aquest procés proporciona una estructura d'objectes d'acord al model DOM, que sí que haureu de recórrer i analitzar per dur a terme la tasca encomanada. Normalment, extreure'n certa informació.

Per poder començar a treballar, per tant, primer cal llegir el fitxer i processar-lo, mitjançant un processador XML. Dins el *package* `javax.xml.parsers` es poden trobar les classes que permeten dur a terme aquesta tasca. Per al cas del model DOM, la classe que us interessa és `DocumentBuilder` (constructor de documents). Aquesta classe té la particularitat, però, que no es pot instanciar directament, sinó que cal emprar una classe auxiliar anomenada `DocumentBuilderFactory` per obtenir-ne objectes.

Un cop conegudes les classes que hi intervenen, el codi per poder obtenir un processador i llegir un fitxer XML sempre ve a ser el mateix, que es mostra tot seguit:

```
1  import javax.xml.parsers.*;
2  import org.w3c.dom.*;
3
4    //...
5
6    DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
7    Document document = null;
8    try {
9      DocumentBuilder  builder = builderFactory.newDocumentBuilder();
10     File f = new File ("elMeuFitxer.xml");
11     document = builder.parse(f);
12   } catch (Exception ex) {
13     System.out.println("Error en la lectura del document: " + ex);
14   }
```

Estrictament, `Document` és una interfície. La instància creada pertany a la classe `Node`.

A l'hora de llegir un document XML, és molt important controlar excepcions, ja que a part dels possibles errors que poden succeir en accedir al sistema de fitxers (el fitxer no existeix, hi ha un error al sistema d'entrada/sortida, el fitxer està protegit...), també cal afegir aquells vinculats a la sintaxi del mateix document. Si el document no és un fitxer XML o no en segueix de manera estricta la sintaxi, també es produeix una excepció i se n'interromp el processament.

Si no ha succeït cap error, a la variable "document" es disposa d'un objecte de la classe `org.w3c.dom.Document`, a partir del qual es pot accedir a tota l'estructura d'objectes (nodes i atributs) que conformen les diferents parts d'un document XML.

Objectes dins el model DOM Java

L'objecte `Document` representa un document complet XML. Ara bé, per analitzar el contingut del fitxer, s'ha d'obtenir el contingut dels seus nodes. Cal tenir en compte que tot objecte DOM pot contenir una gran quantitat de diferents nodes connectats en una estructura d'arbre. De totes maneres, el primer pas sempre és l'obtenció de l'element arrel, a partir del qual es poden iniciar els recorreguts:

```
1 Element arrel = document.getDocumentElement();
```

A la documentació oficial de Java podeu trobar la llista detallada de tots els mètodes disponibles als elements DOM.

Fixeu-vos que aquesta crida retorna un objecte de tipus `Element` (un element del document XML). Sobre un element es poden cridar diversos mètodes. Els més típics són:

- `String getTagName()`, consulta el nom de l'element.
- `String getAttribute(String name)`, donat un nom d'un atribut, si aquest existeix a l'element, retorna el seu valor.
- `NodeList getElementsByTagName(String name)`, proporciona una llista de tots els elements descendents (fills, néts...), partint des d'aquest element, donat nom concret. Es pot usar "*" com a comodí per cercar-los tots. Si no n'hi ha cap, la llista és buida.

A l'hora de treballar amb un model DOM és important anar amb compte amb la jerarquia d'herència de les diferents classes que el conformen, ja que si bé tots els nodes de l'arbre són objectes `Node` (superclasse), només alguns són `Element` (subclasse). Els nodes que són atributs o text no pertanyen a aquesta classe.

La llista completa de constants amb tipus de node la trobareu a la documentació de la classe `Node`

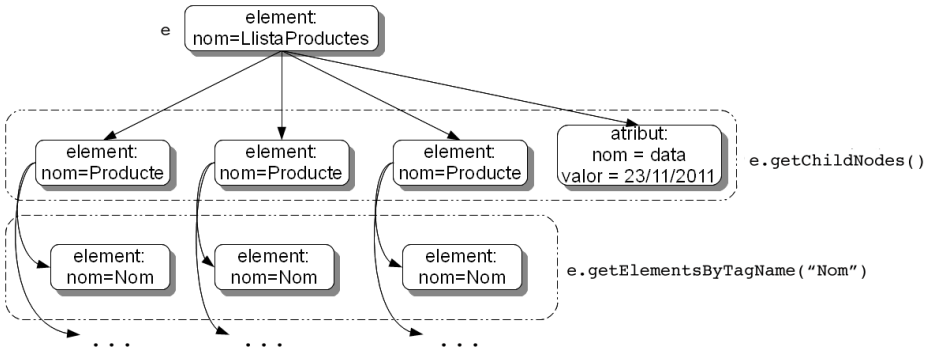
En el cas de la classe `Node`, alguns mètodes interessants per manipular-los són:

- `getNodeType()`, retorna el tipus de node (element, text, atribut...). La classe `Node` disposa de tot un seguit de constants amb les quals es pot comparar el resultat d'invocar aquest mètode per discriminar un node per tipus. Per exemple, `Node.ATTRIBUTE_NODE` indica que és un node del tipus atribut.
- `NodeList getChildNodes(String name)`, proporciona una llista de nodes fill d'aquest element amb un nom concret. Es pot usar "*" com a comodí per demanar tots els fills. Si no n'hi ha cap, la llista és buida. Cal anar amb molt de compte amb aquest mètode, ja que la llista és de nodes, no d'elements, per la qual cosa també pot incloure atributs i nodes de text.

Recordeu que `Element` hereta de `Node` i, per tant, tots aquests mètodes també es poden invocar des d'un objecte d'aquest tipus. També val la pena fer èmfasi en

una diferència important entre `getChildNodes` i `getElementsByTagName` que sovint es passa per alt. El primer només consulta els nodes fills immediatament inferiors, mentre que el segon recorre tot el subarbre cercant elements amb un nom concret, no només els fills. Aquest comportament s'esquematitza a la figura 1.8, on es mostra el resultat d'invocar els dos mètodes sobre l'element arrel del document.

FIGURA 1.8. Diferències d'invocar getChildNodes i getElementsByTagName.



A mode d'exemple, el següent codi permet recórrer un fitxer XML amb una llista de productes i comptar quants n'hi ha a la venda. Per veure les particularitats dels mètodes `getElementsByTagName` i `getChildNodes`, es mostra com fer-ho amb cadascun dels mètodes. Fixeu-vos com, en el primer cas, es pot garantir que els nodes recuperats a la llista són sempre objectes `Element`, mentre que en el segon cas no es pot donar per segur, ja que l'element LlistaProductes també té un atribut, i cal discriminar.

```
Element e = document.getDocumentElement();
  NodeList l = e.getElementsByTagName("Producte");
  for (int i = 0; i < l.getLength(); i++) {
    Element elem = (Element)l.item(i);
    String venda = elem.getAttribute("aLaVenda");
    if ("true".equals( venda) ) {
      nreArticles++;
    }
  }
  System.out.println("Articles a la venda: " + nreArticles);
```

```
Element e = document.getDocumentElement();
  int nreArticles = 0;
  NodeList l = e.getChildNodes();
  for (int i = 0; i < l.getLength(); i++) {
    Node n = l.item(i);
    if (n.getNodeType() == Node.ELEMENT_NODE) {
      Element elem = (Element)n;
      String venda = elem.getAttribute("aLaVenda");
      if ("true".equals( venda) ) {
        nreArticles++;
      }
    }
  }
  System.out.println("Articles a la venda: " + nreArticles);
```

Extracció de text d'un element DOM

De fet, la manera com s'estructuren els diferents tipus de nodes en un model DOM fa que el mecanisme per obtenir el text que hi ha dins un element DOM sigui una mica més complicat del que pot semblar a simple vista, o si més no del que ens agradaria que fos. Val la pena estudiar aquest fet amb detall. El punt més important és que un text dins un element pot estar dispers en diversos nodes de tipus textos dins el model DOM. Per tant, per obtenir tot el text, cal obtenir tots els nodes fill d'un element (usant `getNodeList`), veure quins són de text, i concatenar el contingut de cadascun d'ells.

Dins la classe `Node`, les constants que identifiquen nodes que contenen text són `CDATA_NODE` i `TEXT_NODE`.

Vegem-ne un exemple, en què es llisten els noms dels productes que hi ha al document.

```
Element e = document.getDocumentElement();
  //Obtenir tots els nodes del document anomenats "nom"
  NodeList llistaElems = e.getElementsByTagName("Nom");
  //Recorregut d'elements anomenats "Nom"
  for (int i = 0; i < llistaElems.getLength(); i++) {
    Element elem = (Element)llistaElems.item(i);
    NodeList llistaFills = elem.getChildNodes();
    //Recorregut de nodes fill d'un element (text, atributs, etc.)
    String text = "";
    for (int j = 0; j < llistaFills.getLength(); j++) {
      Node n = llistaFills.item(j);
      //Mirar si els nodes són de text
      if ( (n.getNodeType() == Node.TEXT_NODE)||
          (n.getNodeType() == Node.CDATA_SECTION_NODE) ) {

        text += n.getNodeValue();
      }
      System.out.println(text);
    }
  }
```

1.4.3 Generació de fitxers XML

D'entrada, atès que un fitxer XML no és més que text, sempre es pot generar usant qualsevol mecanisme ofert per Java que permeti escriure cadenes de text en un fitxer. Ara bé, si es vol garantir la seva correctesa abans d'escriure'l, el més convenient és fer exactament el pas invers que en el procediment de lectura. Primer es genera un model DOM, usant les classes corresponents, i tot seguit aquest model s'escriu sobre un fitxer.

Creació del model DOM Java

La creació del model DOM parteix de la generació de l'objecte que representa el document, mitjançant la classe `DocumentBuilder`, i després usa crides als mètodes adients per anar-hi afegint els nodes que correspongui. Primer caldrà afegir l'element arrel al document (només un), i a partir d'aquí caldrà anar afegint tots els elements fills que calgui, amb els seus atributs i el text corresponents, fins

a tenir l'estructura finalitzada. La classe Element proporciona tres mètodes amb els quals afegir atributs, text i crear relacions pare-fill entre dos elements:

- `void setAttribute(String name, String value)`.
- `void setTextContent(String text)`.
- `void appendChild(Node n)`.

Un fet important al llarg d'aquest procés és que la classe `Element` tampoc no es pot instanciar directament amb una sentència `new`. Per crear un element nou cal cridar el mètode `createElement(String nom)`, que ofereix l'objecte document generat. Això es deu al fet que, internament, un document ha de controlar tots els nodes que conté, i per tant, ha de controlar el procés de generació de tot node que hi pertanyi.

El codi següent serveix d'exemple de tot aquest procés. En aquest codi, es genera una part del document XML on s'enumeren productes. Només hi ha un producte, però afegir-ne més consistiria, simplement, a repetir el mateix els cops que fes falta.

```
1  import javax.xml.parsers.*;
2  import org.w3c.dom.*;
3
4    //...
5
6    //Creació del document
7    DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
8    DocumentBuilder  builder = builderFactory.newDocumentBuilder();
9    Document doc = builder.newDocument();
10
11   //Element arrel
12   Element arrel = doc.createElement("LlistaProductes");
13   doc.appendChild(arrel);
14
15   //Element fill de l'arrel
16   Element prod = doc.createElement("Producte");
17   prod.setAttribute("id", "1");
18   prod.setAttribute("aLaVenda", "true");
19
20   //Elements fill de l'anterior
21   Element fill = doc.createElement("Nom");
22   fill.setTextContent("Producte 1");
23   prod.appendChild(fill);
24
25   fill = doc.createElement("Preu");
26   fill.setTextContent("10.0");
27   prod.appendChild(fill);
28
29   fill = doc.createElement("Estoc");
30   fill.setTextContent("4");
31   prod.appendChild(fill);
32
33   arrel.appendChild(prod);
```

Escriptura del model a fitxer

Java ofereix un mecanisme genèric per enviar un document generat usant un model DOM al sistema de sortida de l'aplicació. Mitjançant aquest mecanisme és

possible desar el document a fitxer. Ara bé, al tractar-se d'un mecanisme genèric també és possible triar altres destinacions per a les dades, com la sortida estàndard, de manera que també es pot imprimir el resultat per pantalla.

La classe que s'encarrega de gestionar aquesta tasca principalment és l'anomenada `Transformer`, dins el *package* `javax.xml.transform`. Com en el cas de la construcció de documents, aquesta classe no es pot instanciar directament, sinó que cal l'ajut d'una classe auxiliar anomenada `TransformerFactory`.

El codi per desar el document a fitxer és sempre el mateix, i és el següent:

```
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

...

try {
  TransformerFactory tf = TransformerFactory.newInstance();
  Transformer transformer = tf.newTransformer();
  DOMSource source = new DOMSource(doc);
  File f = new File("nouDoc.xml");
  StreamResult result = new StreamResult(f);
  transformer.transform(source, result);

} catch (Exception ex) {
  System.out.println("Error desant fitxer: " + ex);
}
```

Per enviar el document a la sortida estàndard, n'hi ha prou de canviar la instanciació de l'objecte `StreamResult`, i, en lloc d'usar un fitxer, la variable "f" en aquest cas, escriure-hi "System.out" (sense les cometes).

Si proveu aquest codi i obriu el document resultant, veureu un fet que potser us resulta curiós: tot està escrit en una sola línia de text i l'ordre d'alguns atributs no es correspon amb l'ordre d'invocació dels mètodes durant la creació del document.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><LlistaProductes><
    Producte aLaVenda="true" id="1"><Nom>Producte 1</Nom><Preu>10.0</Preu><
    Estoc>4</Estoc></Producte></LlistaProductes>
```

Aquest fet no us ha d'alarmar, ja que una de les propietats del processament de dades en XML és que els salts de línia o sagnats en el text no tenen cap influència. Només se solen posar per tal de millorar la llegibilitat, però des del punt de vista de discriminar els nodes dins l'estructura, són irrellevants. El processador els ignora. Per això, en escriure el document, ja directament no es tenen en compte. A més a més, cal considerar que en un document XML l'ordre d'aparició d'atributs o elements no modifica la semàntica de les dades (quan se cerqui una informació, estarà allà igualment), i per tant, tampoc no és un problema.