

Java și XML

- **Introducere în XML**
- **Modelul DOM**
- **DOM API**
- **Modelul SAX**
- **SAX API**
- **Serializarea unui flux SAX**
- **Folosirea filtrelor**
- **Parsare mixta SAX+DOM**

Ce este XML ?

Extensible Markup Language

- Tehnologie pentru aplicații Web.
- Organizare structurată a informației.
- Separă conținutul de prezentare.
- Asigură portabilitatea datelor.
- Simplifică tranzacțiile pe Web.
- Căutare îmbunătățită.
- Folosește taguri și attribute.
- Permite definirea de tag-uri proprii.
- Standard pentru reprezentarea bazelor de date de dimensiuni mici și medii pe Web

Structura unui fișier XML

- Prologul

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

- Instrucțiuni de procesare

- Formatul general

```
<?aplicatie instructiuni?>
```

- Reguli de transformare

```
<?xml-stylesheet type="text/xsl"  
                href="fisier.xsl"?>
```

- Datele propriu-zise

```
<nod_radacina>  
</nod_radacina>
```

- Comentarii

```
<!-- Comentariu -->
```

Exemplu de document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Angajatii firmei -->
<personal>
  <angajat id='1' activ='da'>
    <!-- Dom' director -->
    <nume>Popescu</nume>
    <functie>director</functie>
    <contact>
      <telefon>123456</telefon>
      <email>popescu@firma.ro</email>
    </contact>
  </angajat>

  <angajat id='2' activ='nu'>
    <nume>Ionescu</nume>
    <functie>programator</functie>
  </angajat>
</personal>
```

XML versus HTML

- XML : **descrierea** și interschimbarea datelor.
- HTML: **prezentarea** datelor.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Angajatii firmei -->
- <personal>
  - <angajat id="1" activ="da">
    <!-- Dom' director -->
    <nume>Popescu</nume>
    <functie>director</functie>
  - <contact>
    <telefon>123456</telefon>
    <email>popescu@firma.ro</email>
  </contact>
  </angajat>
  - <angajat id="2" activ="nu">
    <nume>Ionescu</nume>
    <functie>programator</functie>
  </angajat>
</personal>
```

Sintaxa XML

- Tagurile trebuie obligatoriu inchise
- $\langle tag/ \rangle = \langle tag \rangle \langle /tag \rangle$
- Tagurile sunt case-sensitive
- Elementele trebuie imbricate corespunzator
- Trebuie să existe obligatoriu un tag radacina, care trebuie să fie și unic
- Atributele trebuie puse între apostrofuri sau ghilimele
- Spatiile sunt pastrate ca atare
- CR/LF este convertit automat la LF
- Comentariile sunt de forma:

`<-- Comentariu XML -->`

Elemente XML

- **Extensibile**
- **Relații de înrudire:**
părinte - fii - frați
- **Au conținut**
 - mixt
 - simplu
 - vid
- **Reguli de denumire**
 - Pot fi formate din orice caractere.
 - Nu trebuie să înceapă cu o cifră sau caracter de punctuație.
 - Nu trebuie să înceapă cu literele `xml` (sau `XML`, `xMl`, etc).
 - Nu pot conține spații.

Attribute XML

- în interiorul tagului
- nume=valoare
- între apostrofuri sau ghilimele

```
<personal>  
  <angajat id='1' activ="da">  
    <nume>Popescu</nume>  
  </persoana>  
  <angajat id="2" activ='nu'>  
    <nume>Ionescu</nume>  
  </persoana>  
</personal>
```

Attribute - meta-informații

Entități

&numeEntitate;

Character	Referință
&	&
<	<
>	>
”	"
'	'

&#codCharacterUnicod;

Character	Referință
ă	ă
î	î
ș	ş
ț	ţ
â	â

Secțiuni CDATA

```
<![CDATA[ text ]]>
```

```
<![CDATA[alfa & beta]]>
```

```
<![CDATA[  
  <atentie>  
    <acestea> <nu sunt> <elemente>  
    <ci doar un text obisnuit>  
  </atentie>  
]]>
```

```
<![CDATA[Traseu:  
  <1> Iasi -----> Mamaia  
  <2> Mamaia -----> Predeal  
  <3> Predeal -----> Iasi  
]]>
```

Validarea documentelor XML

- **Reguli DTD** (Document Type Definition)
- **Scheme XML** (XML Schema)

Documentele XML se împart în:

- **invalide**
- **bine formate**
- **valide**

Un exemplu de fișier DTD:

```
<! ELEMENT elem (subelem\item)>  
<! ELEMENT elem (elem1, elem2, ...)>  
<! ELEMENT elem (#PCDATA)>  
<!ATTLIST elem atribut CDATA #REQUIRED>
```

Specificarea sa într-un fișier XML:

```
<!DOCTYPE nod_radacina SYSTEM "fisier.dtd">
```

Vizualizarea documentelor XML

- **Fișier de prezentare**

```
<?xml-stylesheet type="text/css"
                href="fisier.css"?>
```

- **Fișier de transformare**

```
<?xml-stylesheet type="text/xsl"
                href="fisier.xsl"?>
```

Spații de nume

```
xmlns:prefix="spatiu_nume"
```

```
<-- folosim explicit un prefix -->  
<t:program xmlns:t="spatiu_nume1">  
  <t:autor>Ion</t:autor>  
  <t:limbaj>Java</t:limbaj>  
</t:program>
```

```
<-- folosim un prefix implicit-->  
<program xmlns="spatiu_nume2">  
  <zi>joi</zi>  
  <ora>16</ora>  
  <actiune>curs</actiune>  
</program>
```

Uzual sunt URL-uri

Importanța XML

- Text simplu
- Semantic
- ”Stilabil”
- Reutilizabil
- Legături
- Ușor de procesat
- Ierarhic

Modele de utilizare XML

- Procesarea tradițională de date
- Programare pe bază de documente (Document-Driven)
- Arhivare (serializare)
- Abordare orientată-obiect (Binding)

Java și XML

”Write Once, Run Anywhere”

- Date - XML
- Aplicație - Java

1. Definirea regulilor de validare
2. Generarea (crearea) documentelor
3. Procesarea (parsarea)
4. Vizualizarea informației
5. Actualizarea informației

Arhitectura unei aplicații XML:

UI — Aplicație XML / Parser XML —
Date în format XML

Parsere

Cod pentru procesare XML

1. Crearea unui obiect de tip parser
2. Trimiterea documentului XML ca argument parserului
3. Procesarea rezultatelor

Criterii de clasificare

- Validarea: cu / fără
- Modelul: DOM / SAX
- Limbajul: Java, C++, Perl, Python, etc.

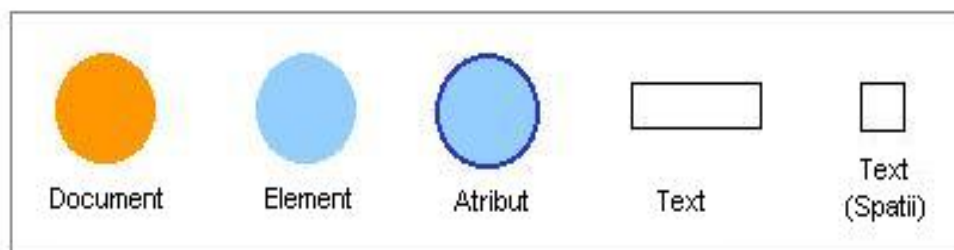
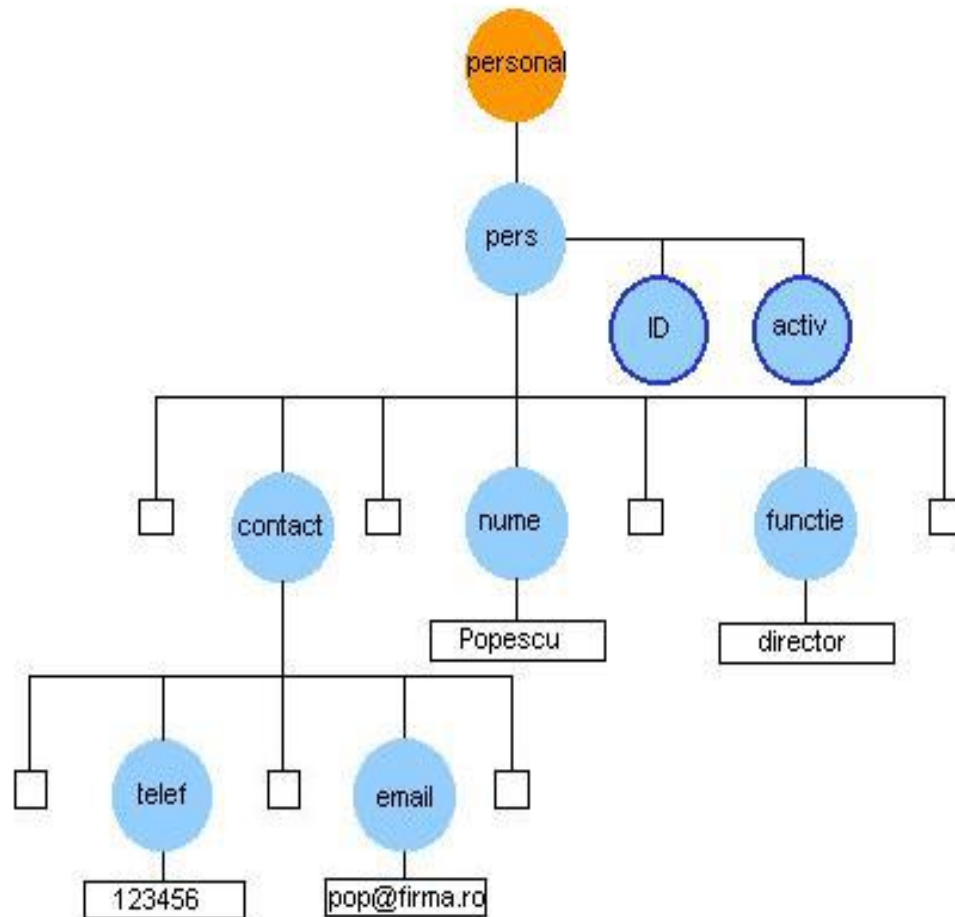
DOM (Document Object Model)

- Structură arborescentă
- Conține întreaga informație a documentului
- Format din noduri
- Nodurile pot fi elemente, texte, attribute, spații, etc.
- DOM descrie nodurile și relațiile
- Permite actualizarea informațiilor
- Parselele DOM sunt ”tree-based”

Avantaj: soluție completă de procesare

Dezavantaj: nu este scalabil

Exemplu



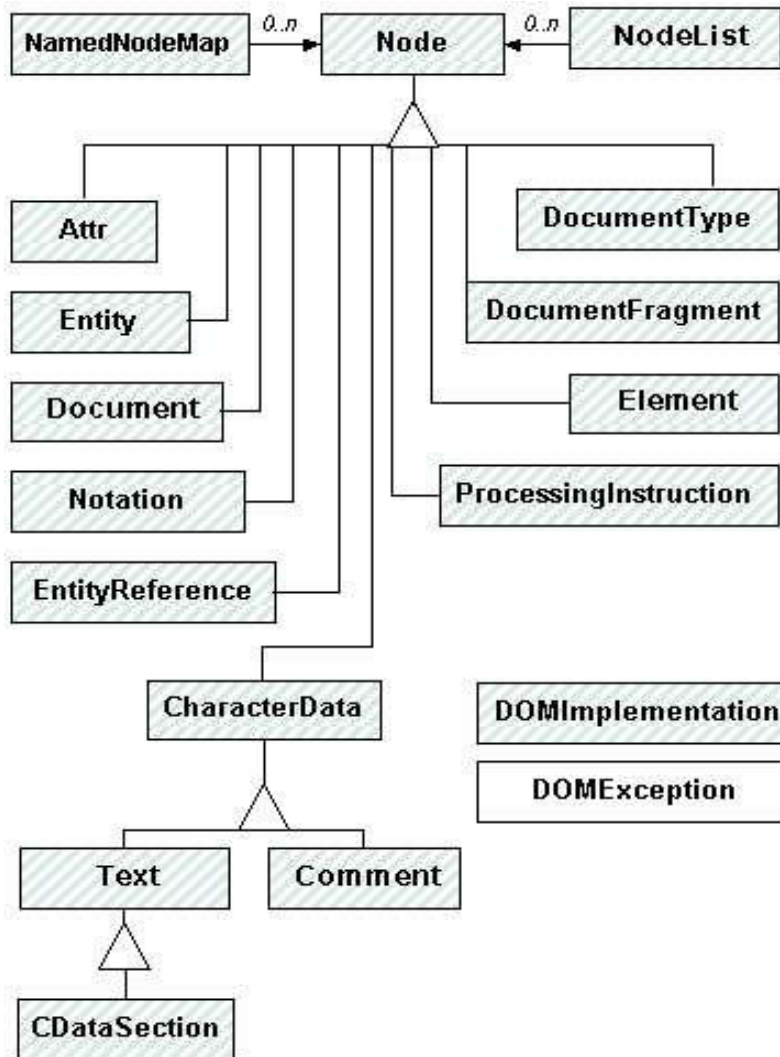
DOM API

- Interfețe pentru reprezentarea informațiilor dintr-un document XML: elemente, texte, etc.
- Metode și proprietati necesare pentru manipularea acestora.

Distribuția JAXP: dom.jar, jaxp_api.jar

- javax.xml.parsers
- org.w3c.dom

Interfața Node



Constante

`Node.ELEMENT_NODE`, `Node.TEXT_NODE`...

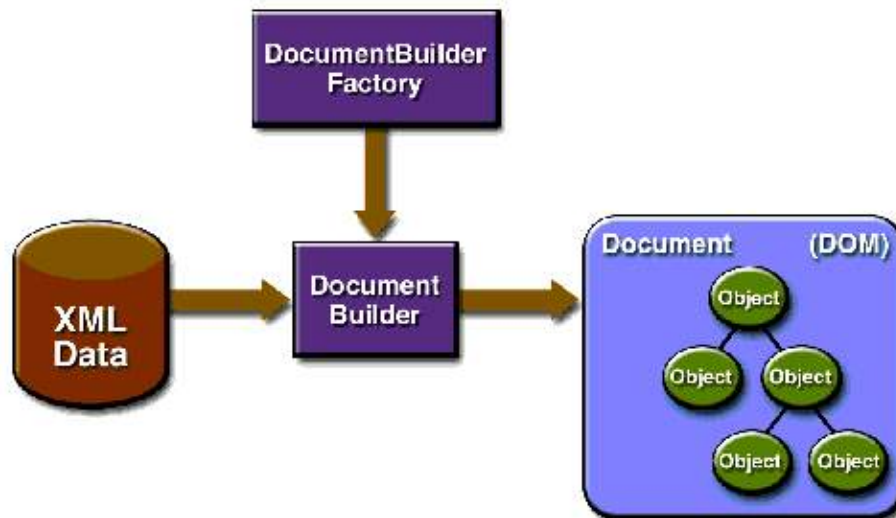
Crearea arborelui

```
// 1. Crearea unui obiect DocumentBuilderFactory
DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);

// 2. Crearea unui obiect DocumentBuilder
DocumentBuilder db = dbf.newDocumentBuilder();

// 3. Crearea arborelui corespunzator
//    unui fisier specificat
Document doc = db.parse(new File(umeFisier));

// Crearea unui arbore vid:
Document doc = db.newDocument();
```



Parcurgerea arborelui

1. Recursiv

```
private void parcurge(Node n) {  
  
    int type = n.getNodeType();  
  
    switch (type) {  
        case Node.DOCUMENT_NODE:  
            //proceseaza  
  
        case Node.ELEMENT_NODE:  
            //proceseaza  
            NamedNodeMap atts = n.getAttributes();  
            for (int i = 0; i < atts.getLength(); i++) {  
                Node att = atts.item(i);  
                parcurge(att);  
            }  
            break;  
  
        case Node.TEXT_NODE:  
            //proceseaza  
    }  
}
```

```

//Afiseaza fii, daca exista
//varianta 1:
NodeList children = n.getChildNodes();
for(int i=0 ;i < children.getLength(); i++) {
    Node child = (Node) children.item(i);
    parcurge(child);
}

//varianta 2:
for (Node child = n.getFirstChild(); child != null;
    child = child.getNextSibling()) {
    parcurge(child);
}
}

```

Extragerea informației din nod:
getNodeName, getNodeValue

2. Lista nodurilor în preordine

```

NodeList nodes =
    doc.getElementsByTagName("element");

```

Modificarea informațiilor din noduri

create[Tip]Node, setNodeValue,
appendChild, removeChild, replaceChild

```
// Obținem sau cream radacina arborelui
Element root = doc.getDocumentElement();
if (root == null) {
    root = doc.createElement("personal");
    doc.appendChild(root);
}
// Adaugam nod
Element angajat = doc.createElement("angajat");
angajat.setAttribute("id", "2");
angajat.setAttribute("activ", "da");
root.appendChild(angajat);

Element nume = doc.createElement("nume");
nume.appendChild(
    doc.createTextNode("Ionescu"));
angajat.appendChild(nume);

...
```

Afişarea documentelor XML folosind DOM

```
doc.normalize();  
Element root =  
    doc.getDocumentElement();  
System.out.println(root.toString());
```

SAX (Simple API for XML)

- Parcurgere secvențială a documentului XML
- Generare de evenimente
- Nu mai este construit întreg arborele
- Abordare eficientă: memorie
- Nivelul de procesare al parserului este minimal
- Parselele SAX sunt ”event-based”

Avantaje:

- Eficient pentru documente mari
- Consum redus de memorie

Dezavantaje:

- Funcționalitate redusă
- Informația este desprinsă din context

SAX API

JAXP: sax.jar, jaxp_api.jar

- javax.xml.parsers
- org.xml.sax
- org.xml.sax.helpers

- Crearea unui "event-handler"
- Crearea unui parser SAX
- Asocierea dintre handler și parser
- Parcurgerea documentului și tratarea evenimentelor generate

Crearea unui "event-handler"

- Implementare **ContentHandler**
- Extindere **DefaultHandler**

Metode uzual supradefinite:

- `startDocument`
- `startElement`
- `characters`
- `endDocument`

```
public class MyEventHandler extends DefaultHandler {
    public void startDocument() throws SAXException {
        ...
    }
    ...
}
```

Crearea unui parser

```
//Pas 1
SAXParserFactory spf =
    SAXParserFactory.newInstance();

//Pas 2
SAXParser saxParser = spf.newSAXParser();
XMLReader xmlReader = saxParser.getXMLReader();

//Pas 3
xmlReader.setContentHandler(
    new MyEventHandler());

//Pas 4
xmlReader.parse(
    new File(umeFisierXML).toURL().toString());
// sau
InputSource source = new InputSource(umeFisierXML);
xmlFilter.parse(source);
```

Inițializarea și finalizarea
startDocument, endDocument

Procesarea elementelor

```
void startElement(String namespaceURI,  
                  String localName,  
                  String qName,  
                  Attributes atts)  
    throws SAXException  
void endElement(...)
```

Procesarea blocurilor de text

```
void characters(char[] ch,  
               int start,  
               int length)  
    throws SAXException  
...  
String text = new String(ch, start, length);
```

Tratarea erorilor

- Implementare **ErrorHandler**
- Extindere **DefaultHandler**

Metode ce pot fi supradefinite:

- `warning`
- `error`
- `fatalError`

```
public class MyErrorHandler extends DefaultHandler {  
    public void error (SAXParseException e) {}  
    public void warning (SAXParseException e) {}  
    public void fatalError (SAXParseException e) {}  
}  
  
...  
xmlReader.setErrorHandler(new MyErrorHandler());
```

Serializarea unui flux SAX

```
//1. Crearea unui obiect Serializer
Serializer serializer =
    SerializerFactory.getSerializer(
        OutputProperties.getDefaultMethodProperties("xml"));

//2. Stabilirea destinatiei
serializer.setOutputStream(System.out);

//3. Handlerul este cel oferit de Serializer
xmlReader.setContentHandler(serializer.asContentHandler());

//4. Parcurgerea fisierului XML
InputSource source = new InputSource("fisier.xml");
xmlReader.parse(source);
```

Folosirea filtrelor



```

//1. Crearea unui clase de tip XMLFilter
public class MyFilter extends XMLFilterImpl {
    MyFilter(){}
    MyFilter(XMLReader parent) {
        super(parent);
    }
    public void startElement(...) {
        //altereaza informatia
        super.startElement(...);
    }
}

//2. Crearea unui instante a filtrului si stabilirea parii
// (handlerul care ar trata normal evenimentele)
xmlReader = saxParser.getXMLReader();
MyFilter xmlFilter = new MyFilter();
xmlFilter.setParent(xmlReader);

//3. Stabilirea handlerului ca fiind obiectul de tip filt
xmlFilter.setContentHandler(new MyEventHandler());
xmlFilter.setErrorHandler(new MyErrorHandler());

//4. Parcurgerea documetului XML
InputSource source = new InputSource("fisier.xml");
xmlFilter.parse(source);

```

SAX + DOM

La fiecare pas la procesării vor fi generate evenimente ce furnizează nu doar informația unui nod ci, în mod controlat, arborele DOM corespunzător aceluși nod. De exemplu, aplicația poate analiza atributele nodului și determina dacă dorește sau nu arborele aferent.

O implementare a acestui model este **SAXDOMIX**
<http://www.devsphere.com>.