

Metode avansate de programare

Informatică Româna, 2017-2018, Curs 8

Fișiere XML

Ce este un document XML?

- Un document XML este un **arbore** ce contine:
 - **noduri frunza** - noduri cu date caracter
 - **noduri element**:
 - etichetate cu un nume si o multime de attribute, fiecare din ele avand un nume si o valoare,
 - acestea pt contine unul sau mai multi copii.

Structura unui document xml

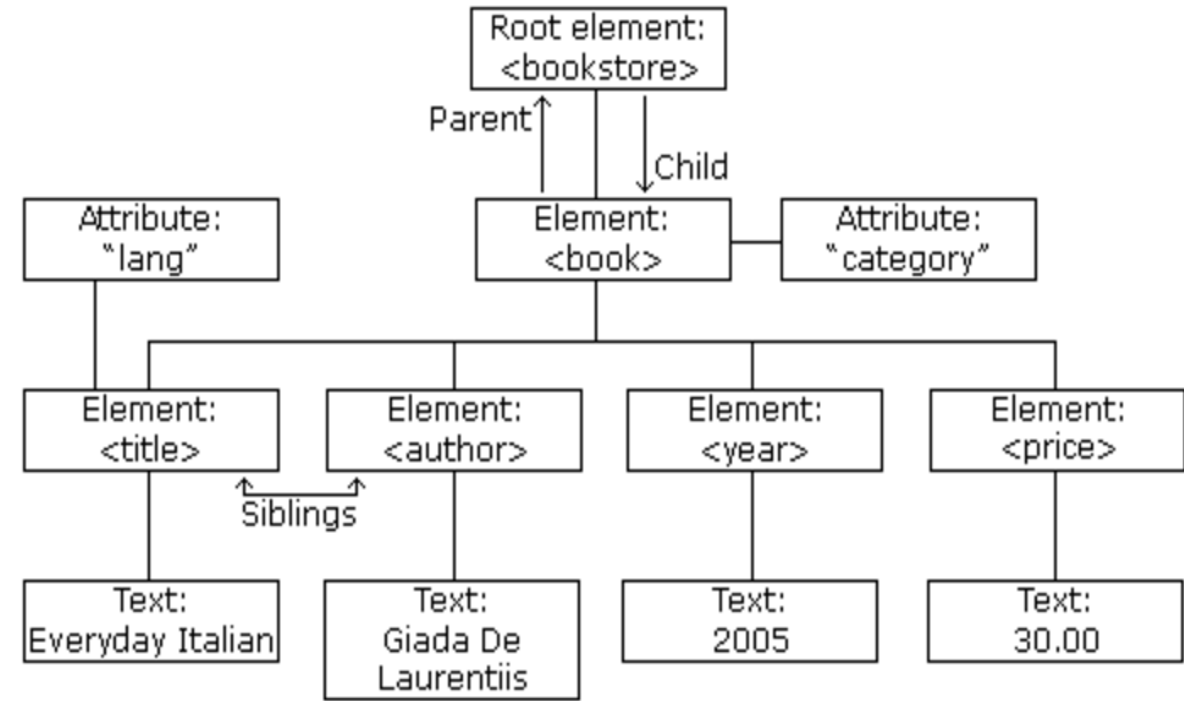
Un fisier XML cuprinde urmatoarele sectiuni:

- **Prolog (optional):** ex. `<?xml version="1.0" encoding="UTF-8"?>` Informeaza ca urmeaza descrierea unui fisier XML ce respecta versiunea de specificatie 1.0 iar setul de caractere utilizat este codificat **UTF-8**
- **Definitia tipului de document (optionala)** – `<!DOCTYPE note SYSTEM "note.DTD">`
Precizeaza ca fisierul *note.DTD* contine **declaratia tipului de document** (DTD-ul), document ce are ca radacina tag-ul **note**. Acesta este un set de reguli ce definesc structura unui fisier XML.

```
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>
```
- **Elementul radacina** `<note> ... </note>`

Structura arborelui XML

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday </title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```



XML element

- **Element**=Blocul de baza al unui document XML.
 - Pot fi folosite atat pentru a retine informatii, cat si pentru definirea structurii doc XML.
- **Element**= orice declaratie `<tag> ...</tag>`
- Un Element poate contine:
 - Text
 - Atribute
 - Alte elemente
 - Combinatii de text, attribute si elemente
 - Un element poate fi **vid** `<tag/>`



Node.ELEMENT_NODE, **Node.ATTRIBUTE_NODE**,
Node.COMMENT_NODE ...

Exemplu

```
<bookstore>  
  <book category="cooking">  
    <title lang="en">Everyday Italian</title>  
    <author>Giada De Laurentiis</author>  
    <year>2005</year>  
    <price>30.00</price>  
  </book>  
  <book category="children">  
    <title lang="en">Harry Potter</title>  
    <author>J K. Rowling</author>  
    <year>2005</year>  
    <price>29.99</price>  
  </book>  
</bookstore>
```

<title>, <author>, <year>, and <price> au **continut text**
<bookstore> and <book> **au continut elemente**
<book> are un **atribut** (category="children").

Attribute

- Atributele au rolul de a descrie elementele.
- Atributele sunt **localizate in tag-ul de start al unui element**, imediat dupa numele elementului
- Pentru un element pot exista oricate atribute, atat timp cat sunt declarate corect.

```
<nume_tag numeAtribut1="valoare1" ... numeAtributN="valoareN" ...  
</nume_tag>
```

```
<student id="2">  
  <firstName>"Popescu2"</firstName>  
  <lastName>"Andrei2"</lastName>  
  <email> "andrei2@yahoo.com" </email>  
</student>
```

Comentarii

- `<!-- Acesta este un comentariu -->`

Referințe la entități

&nume_entitate; În XML, entitățile sunt unitati de text, unde o unitate poate fi orice, de la un singur caracter la un întreg document sau chiar o referință la un alt document.

Una dintre cele mai frecvente utilizări ale referințelor la entități este atunci când se dorește folosirea unor caractere care ar duce la apariția unor confuzii pentru analizorul XML. În acest caz există cinci entități predefinite în XML:

Entitate	Referința la entitate
<	<
>	>
&	&
'	'
"	"e;

Sectiuni CDATA

- Sectiunile CDATA sunt utilizate pentru a include blocuri de text continand caractere care altfel ar fi recunoscute ca marcaje.
- O restrictie de sintaxa este faptul ca in interiorul sectiunilor CDATA nu poate sa apara sirul ']]'. Încă un lucru de retinut este ca sectiunile CDATA nu pot fi incluse unele in altele.

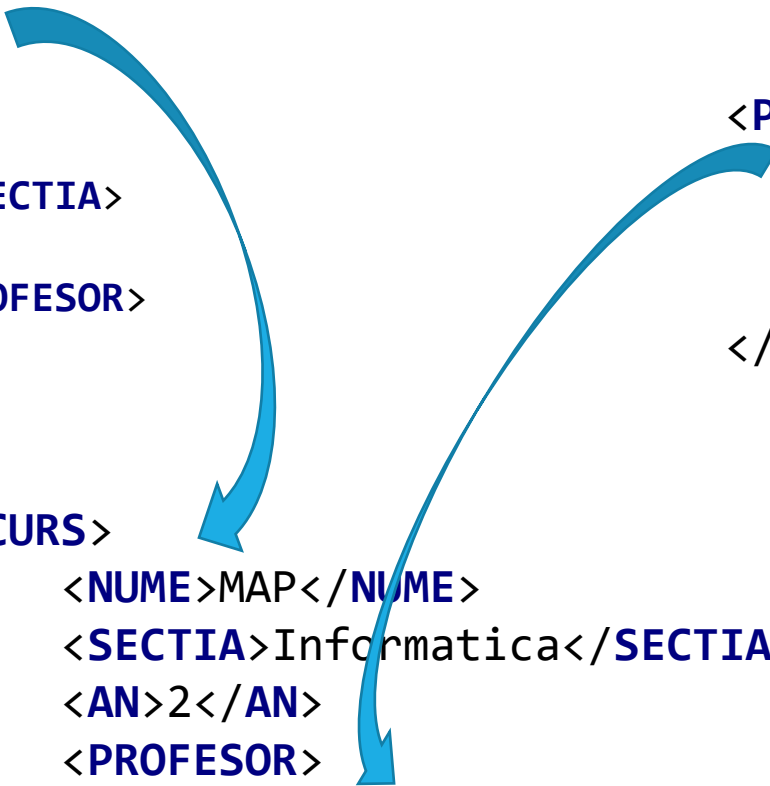
```
<?xml version="1.0" encoding="UTF-8"?>
<exemplu>
Un exemplu de creare a unui tabel in HTML:
<![CDATA[
    <table align="center">
        <tr>
            <td>Coloana 1</td>
            <td>Coloana 2</td>
        </tr>
    </table>
]]>
</exemplu>
```

Domenii de nume (namespaces)

Scenariu:

```
<CURS>  
  <NUME>MAP</NUME>  
  <SECTIA>Informatica</SECTIA>  
  <AN>2</AN>  
  <PROFESOR> Popescu</PROFESOR>  
</CURS>
```

```
<PERSOANA>  
  <NUME>Popescu</NUME>  
  <PRENUME>Dan</PRENUME>  
  <AN>1993</AN>  
</PERSOANA>
```



```
<CURS>  
  <NUME>MAP</NUME>  
  <SECTIA>Informatica</SECTIA>  
  <AN>2</AN>  
  <PROFESOR>  
    <PERSOANA>  
      <NUME>Popescu</NUME>  
      <PRENUME>Dan</PRENUME>  
      <AN>1993</AN>  
    </PERSOANA>  
  </PROFESOR>  
</CURS>
```

Domenii de nume

- Pentru a putea face distincție între elementele ce reprezintă concepte diferite, le vom încadra pe fiecare într-un **domeniu de nume** (namespace). Fiecarui domeniu de nume îi vom asocia un **URI** unic.
- Un **URI (Universal Resource Identifier)** este o generalizare a unui URL. În timp ce un URL identifică o locație un URI identifică o resursă.
 - De exemplu, un URI pentru o persoană poate fi codul numeric personal. Asta nu înseamnă că putem să vizualizăm o persoană în browser specificându-i cnp-ul.
 - Acest URI nu poate avea întotdeauna la un fișier anume. URI-ul definește un domeniu de nume pur formal.

Definirea domeniilor de nume - atributul *xmlns*

```
<C:CURS xmlns:C="kurs.xsd" xmlns:P="persoana.xsd" >
  <C:NUME>Programare in XML</C:NUME>
  <C:SECTIA>Informatica</C:SECTIA>
  <C:AN>3</C:AN>
  <C:PROFESOR>
    <P:PERSONA >
      <P:NUME>Mihai</P:NUME>
      <P:PRENUME>Gabroveanu</P:PRENUME>
      <P:AN>1976</P:AN>
    </P:PERSONA>
  </C:PROFESOR>
</C:CURS>
```

Documente bine formate (Well-Formed Documents)

- Un document XML este un document bine format daca satisface urmatoarele conditii sintactice:
 - are exact un singur element radacina (root element)
 - fiecare element are un tag de inceput si unul de sfarsit
 - tag-urile sunt inchise corect, adica **NU** sunt de forma:

```
<autor><nume>Elliotte Rusty Harold</autor></nume>
```
 - Primul tag deschis trebuie sa fie ultimul care este inchis. Tag-urile trebuie inchise exact in ordinea inversa a deschiderii lor, altfel va fi semnalata eroare.
 - Numele atributelor sunt unice in cadrul unui element
- Cu alte cuvinte un document XML este bine format daca respecta regulile sintactice descrise de standardul XML.

Validarea documentelor xml

- Un document XML este valid daca:
 - Este bine format
 - Respecta o schema, care stabileste un set de reguli referitoare la modul de definire a respectivului document XML
- Exista trei tipuri de scheme de definire a documentelor XML:
 - **DTD** - The original Document Type Definition
 - **XML Schema** - An XML-based alternative to DTD
 - **Relax NG** – Regular Language for XML Next Generation

XML DTD

- http://www.w3schools.com/xml/xml_dtd.asp
- Este cea mai veche schema de format de fisier XML
- Un doc DTD specifica **tipurile de tag-uri care pot fi incluse intr-un document XML si ordinea acestora**

DTD extern `<!DOCTYPE element_radacina SYSTEM "SYSTEM-URI">`

```
<!-- message.dtd file -->
<?xml version="1.0"?>
<!ELEMENT message (to, from, heading, body)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

```
<!-- message.xml file -->
<?xml version="1.0" encoding="UTF-8"?>
<!-- this is a comment -->
<!DOCTYPE message SYSTEM "message.dtd">
<message>
  <to>Aprogramatoarei</to>
  <from>Yan</from>
  <heading>Reminder</heading>
  <body>Don't forget to do your MAP labs
  </body>
</message>
```


DTD - intern

- ```
<?xml version="1.0" encoding="UTF-8"?>
<!--
<!DOCTYPE message SYSTEM "message.dtd" -->
<!DOCTYPE message [
 <!ELEMENT message (to, from, heading, body)>
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT heading (#PCDATA)>
 <!ELEMENT body (#PCDATA)>
]>
<message>
 <to>Aprogramatoarei</to>
 <from>Yan</from>
 <heading>Reminder</heading>
 <body>Don't forget to do your MAP labs</body>
</message>
```

# DTD

```
<!-- message.xml file -->
<?xml version="1.0"?>
1) <!DOCTYPE message [
2) <!ELEMENT message (to, from, heading, body)>
3) <!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

```
<message>
 <to>Aprogramatoarei</to>
 <from>Yan</from>
 <heading>Reminder</heading>
 <body>text </body>
</message>
```

## unde:

- 1) definește ca acest document este de tip **message**
- 2) definește elementul **message** ca fiind format din patru elemente: **to**, **from**, **heading**, **body**
- 3) definește elementul **from** ca fiind unul de tip **#PCDATA** (text)

# DTD Declararea elementelor - ELEMENT

`<!ELEMENT nume-element (continut-element)>`

Continutul unui element este specificat de cateva cuvinte cheie:

- **#PCDATA** pentru parsarea caracterelor text
- **EMPTY** pentru continut vid
- **ANY** pentru orice continut
- **|** pentru alternative
- **()** pentru grupuri
- **\*** pentru orice numar
- **+** cel putin o data
- **?** optional

# DTD Declararea atributelor - ATTLIST

```
<!ATTLIST element-name attribute-name attribute-type #DEFAULT default-value>
<!ATTLIST element-name attribute-name attribute-type #FIXED fixed_value>
<!ATTLIST element-name attribute-name attribute-type (Val1|Val2|..) default_val>
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```

Tip	Descriere
CDATA	Date caracter
ENTITY	Valoarea este o entitate
ENTITIES	Valoarea este o lista de entitati
ID	Valoarea este un id unic
IDREF	Valoarea este o referinta la un alt id
IDREFS	Valoarea este o lista de referinte la alte id-uri
NMTOKEN	Valoarea este un nume valid XML
NMTOKENS	Valoarea este o lista de nume valide XML
NOTATION	Numele unei notati
(val1 val2 ...)	Lista de valori
xml:	Valoarea este una predefinita in XML

```
<messages >
 <message>
 <to username="aprog" >Aprogramatoarei</to>
 <from>Serban </from>
 <heading>Reminder</heading>
 <body>Don 't forget to do your MAP labs </body>
 </message>
</messages>
```

```
<!DOCTYPE messages [
 <!ELEMENT messages (message)*>
 <!ELEMENT message (to|from|heading|body)*>
 <!ELEMENT to (#PCDATA)>
 <!--ATTLIST to
 username ID #REQUIRED-->
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT heading (#PCDATA)>
 <!ELEMENT body (#PCDATA)>
]>
```

# Declararea entitatilor - ENTITY

<!ENTITY entity-name entity-value>

[http://xmlwriter.net/xml\\_guide/entity\\_declaration.shtml](http://xmlwriter.net/xml_guide/entity_declaration.shtml)

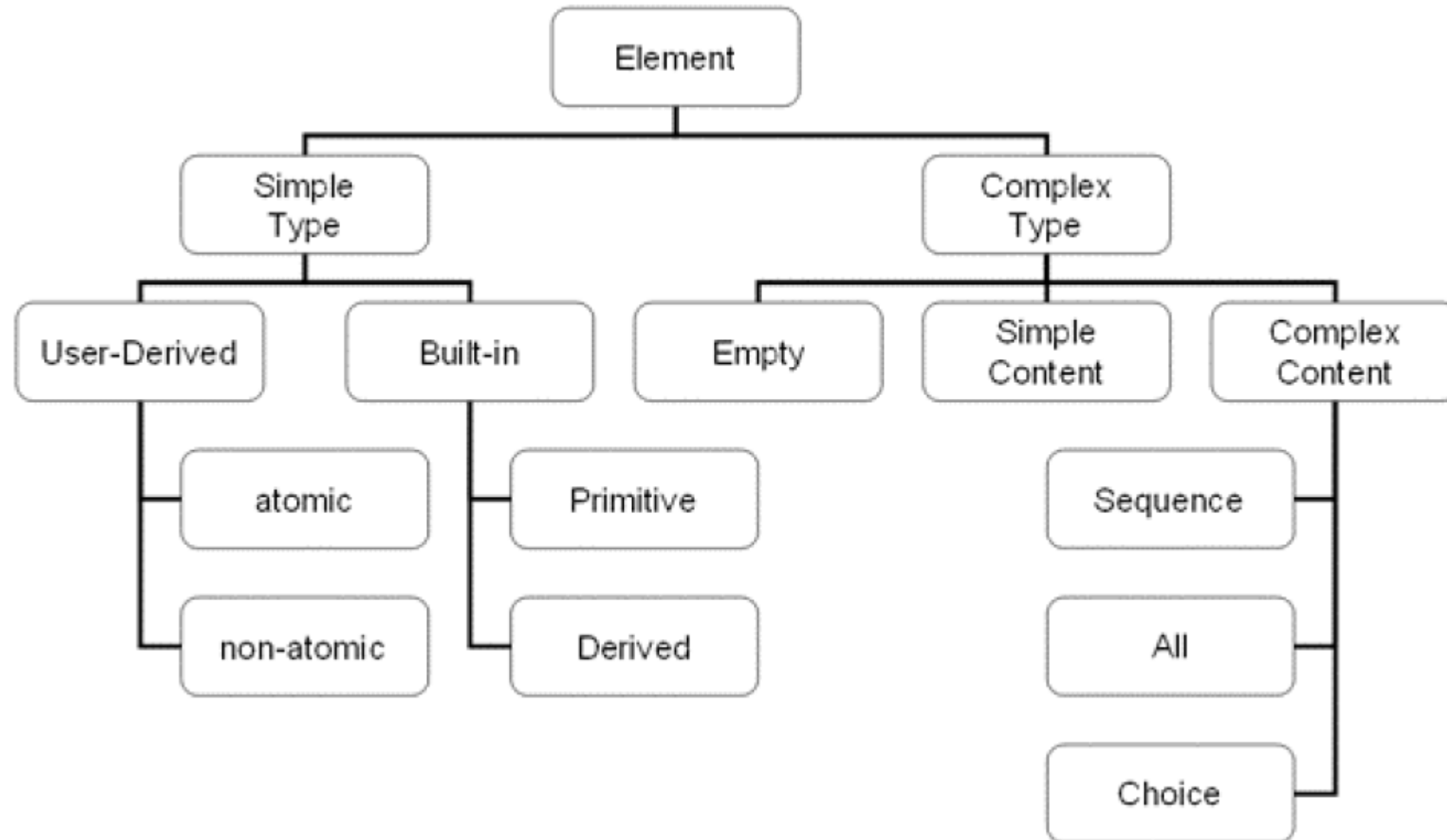
```
<!DOCTYPE messages [
 <!ELEMENT messages (message)*>
 <!ELEMENT message (to|from|heading|body)*>
 <!ELEMENT to (#PCDATA)>
 <!ATTLIST to
 username CDATA #REQUIRED>
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT heading (#PCDATA)>
 <!ELEMENT body (#PCDATA)>
 <!ENTITY semnatura "CSerban">
]>
<messages >
 <message>
 <to username="aprogramatoarei" >Aprogramatoarei</to>
 <from>Serban </from>
 <heading>Reminder</heading>
 <body>Don't forget to do your MAP labs &semnatura;</body>
 </message>
</messages>
```

# XML Schema

O XML Schema:

- definește elementele care pot să apară într-un document
- definește atributele pe care pot să le aibă elementele dintr-un document
- definește care elemente au copii și care sunt aceștia
- definește ordinea copiilor unui element
- definește numărul de copii
- definește dacă un element este vid sau poate conține un text
- definește tipurile elementelor și atributelor
- definește valorile implicite și fixe ale elementelor și atributelor
- [http://www.w3schools.com/xml/xml\\_schema.asp](http://www.w3schools.com/xml/xml_schema.asp)

# XML Schema elements



# Referirea la o xml schema

Spre deosebire de referirea la un DTD intr-un document XML, referire care se facea inainte de elementul radacina:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE messages SYSTEM "message.dtd">
<messages>
 <message> ... </message>
</messages>
```

Referirea la o XML Schema intr-un document XML se face in cadrul elementului radacina:

```
<messages xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance 1)
 xsi:noNamespaceSchemaLocation="message.xsd"> 2)
 <message>
 ...
 </message>
</messages>
```

1) Referirea la instanta de schema XML utilizata

2) Locatia unde poate fi gasita definitia schemei



# Definirea unui fisier XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

</xs:schema>
```

```
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Indica faptul ca elementele si tipurile de date utilizate in schema (schema, element, complexType, sequence, string, boolean, etc.) provin din namespace-  
**"http://www.w3.org/2001/XMLSchema".**

De asemenea specifica ca aceste elemente si tipuri de date ce provin din namespace-ul  
**"http://www.w3.org/2001/XMLSchema" trebuie sa fie prefixate cu xs:**

# Definirea unui element simplu

Un element se numeste *element simplu* daca:

- Contine numai informatie text (date caracter)
- Nu are attribute
- Nu contine alte elemente
- Nu este vid

**Observatie:** informatia text continuta de element poate sa fie de unul dintre tipurile predefinite in XML Schema (boolean, string, date, etc.), sau poate fi de un tip definit de noi.

```
<NUME>Popescu Maria</NUME>
<VARSTA>21</VARSTA>
<DATA_NASTERII>1985-09-23</DATA_NASTERII>
```

```
<xs:element name="NUME" type="xs:string"/>
<xs:element name="VARSTA" type="xs:integer"/>
<xs:element name="DATA_NASTERII" type="xs:date"/>
```

**Observatie:** Elementele simple pot avea in anumite situatii **valori implicite** SAU o valoare **fixa**.

```
<xs:element name="TIP-IMPRIMANTA" type="xs:string" default="alb-negru"/>
<xs:element name="CETATENIA" type="xs:string" fixed="romana"/>
```

# Definirea atributelor

```
<xs:attribute name="cetatenia" type="xs:string" default="romana" />
```

```
<xs:attribute name="cetatenia" type="xs:string" fixed="romana" />
```

```
<xs:attribute name="cetatenia" type="xs:string" fixed="romana" use="optional"/>
```

```
<xs:attribute name="cetatenia" type="xs:string" fixed="romana" use="required"/>
```

# Definirea restricțiilor/fațetelor

## Restricții pentru numere

```
<xs:element name="nota">
 <xs:simpleType>
 <xs:restriction base="xs:integer">
 <xs:minInclusive value="1">
 <xs:maxInclusive value="10">
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

# Definirea restricțiilor/fațetelor

## Restricții pentru stringuri

```
<xs:element name="INITIALE">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:pattern value="[A-Z][A-Z][A-Z]"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

# Definirea restricțiilor/fațetelor

**Restricții pe o multime de valori** - elementul ANOTIMP poate lua ca valori numai: primavara, vara, toamna si iarna

```
<xs:element name="ANOTIMP">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:enumeration value="primavara"/>
 <xs:enumeration value="vara"/>
 <xs:enumeration value="toamna"/>
 <xs:enumeration value="iarna"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

# Definirea elementelor complexe

Un element se numeste **element complex** daca contine alte elemente si/sau attribute. De asemenea elementele vide sunt considerate elemente complexe.

```
<xs:element name="PERSOANA">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="NUME" type="xs:string"/>
 <xs:element name="PRENUME" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

Definirea elementelor vide    <PRODUS cod="1345" />

```
<xs:element name="PRODUS">
 <xs:complexType>
 <xs:attribute name="cod" type="xs:positiveInteger"/>
 </xs:complexType>
</xs:element>
```

# Definirea elementelor complexe

## Utilizarea tipurilor declarate

```
<xs:element name="PERSONA">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="NUME" type="xs:string"/>
 <xs:element name="PRENUME" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<xs:complexType name="persoanaType">
 <xs:sequence>
 <xs:element name="NUME" type="xs:string"/>
 <xs:element name="PRENUME" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
<xs:element name="PERSONA" type="persoanaType"/>
```

**Warning** - Nu putem utiliza o referinta la un tip daca el este local unui alt tip!



# Definirea elementelor complexe

**Extinderea unui tip** - Putem construi un tip complex prin extinderea unui alt tip complex

```
<xs:complexType name="tip-nou">
 <xs:complexContent>
 <xs:extension base="alt-tip">
 ... caracteristici noi...
 </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

**Observatie:** `complexContent` semnaleaza ca intentionam sa **extindem** un tip complex!

# Indicatori de ordine

**Indicatorul de secventa `xs:sequence`** - elementele ce trebuie sa apara intr-o anumita ordine

**Indicatorul all `xs:all`** Ordinea in care trebuie sa apara elementele intr-un tip complex nu conteaza

**Indicatorul de alegere `xs:choice`** - Poate sa apara doar unul dintre elemente

```
<xs:element name="CARTE">
 <xs:complexType>
 <xs:choice>
 <xs:element name="AUTOR" type="persoana"/>
 <xs:element name="EDITOR" type="persoana"/>
 </xs:choice>
 </xs:complexType>
</xs:element>
```

# Indicatori de aparitie

maxOccurs / minOccurs

```
<xs:element name="PERSOANA">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="NUME" type="xs:string"/>
 <xs:element name="NUME-COPIL" type="xs:string" maxOccurs="5"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

# Parseare XML

## Dom

- un API bazat pe o structura arborescenta, oferind interfețe pe componentele arborelui (care este un document DOM), cum ar fi interfața Document, interfața Node, interfața NodeList, interfața Element, interfața Attr etc...
- Un parser DOM creează o structură arborescenta în memorie din documentul de intrare, oferind întregul document, indiferent de cât de mult este necesar clientului.

## SAX

- Parserul SAX este un API bazat pe evenimente. De obicei, un API pe bazat pe evenimente furnizează interfețe de tipul handler. Există patru interfețe handler: ContentHandler, DTDHandler, EntityResolver și ErrorHandler.

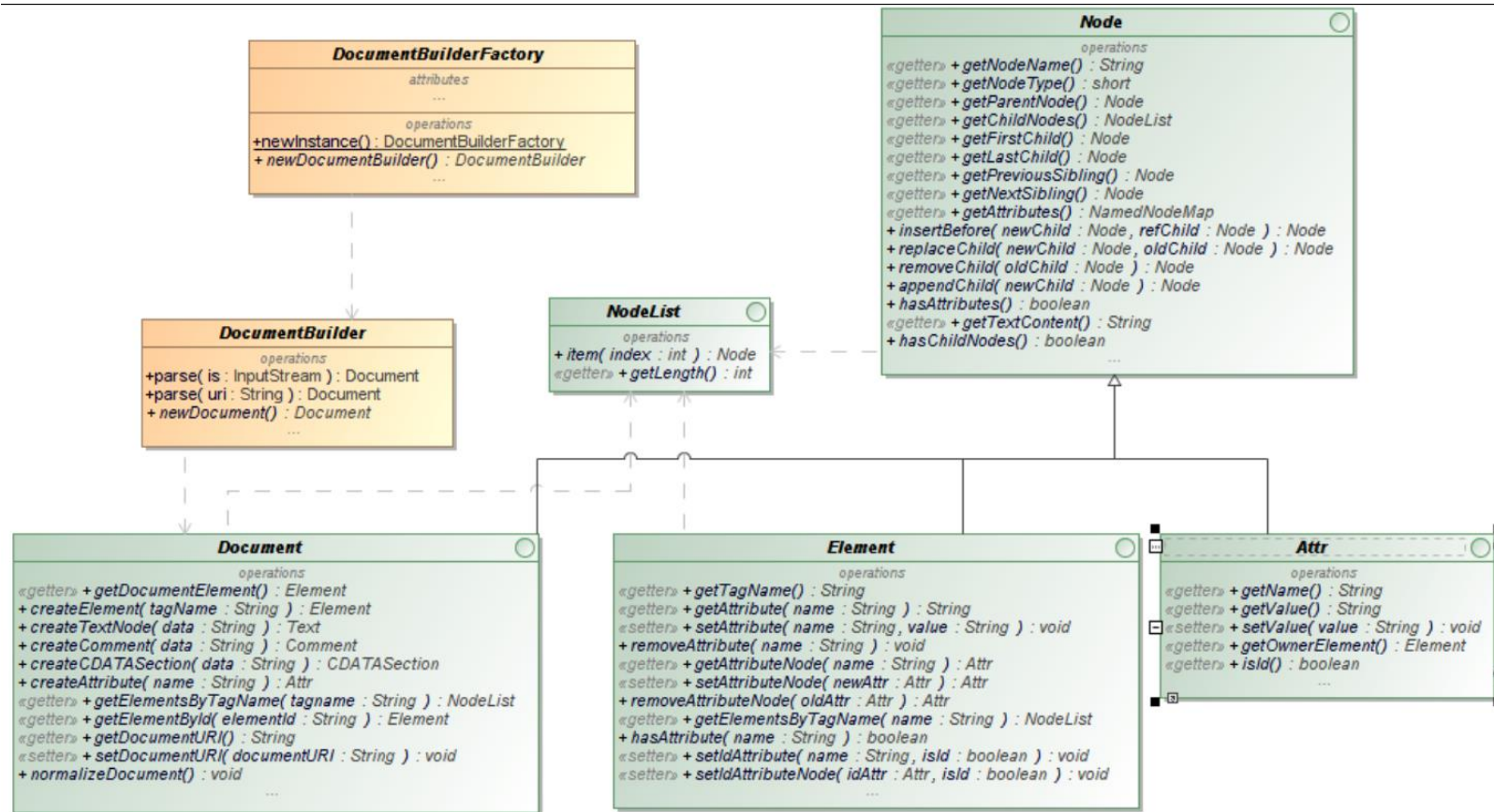
**StAX Parser** – Parsează un document XML într-un mod foarte asemănător cu SAX, dar mult mai eficient...

.....

\_\_\_\_\_

<http://howtodoinjava.com/xml/java-xml-dom-parser-example-tutorial/>

<https://www.mkyong.com/java/how-to-read-xml-file-in-java-dom-parser/>



# Dom parser load document

```
Document loadDocument() {
 try {
 DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
 DocumentBuilder docBuilder = null;
 Document doc=null;
 docBuilder = docFactory.newDocumentBuilder();
 doc= docBuilder.parse(new FileInputStream("Studenti.xml"));
 return doc;
 } catch (ParserConfigurationException e) {
 e.printStackTrace();
 } catch (SAXException e) {
 e.printStackTrace();
 } catch (IOException e) {
 e.printStackTrace();
 }
 return null;
}
```

```
<students>
 <student id="1">
 <firstName value="Popescu"/>
 <lastName value="Stanila"/>
 <emailName value="sasa@sasas"/>
 </student>
</students>
```

# Dom parser loadData()

```
public void loadData() {
 Document document = loadDocument();
 Node root = document.getDocumentElement();
 NodeList nodeList = root.getChildNodes();
 List<Student> students = new ArrayList<>();
 for (int i = 0; i < nodeList.getLength(); i++) {
 Node node = nodeList.item(i);
 if (node.getNodeType() == Node.ELEMENT_NODE) {//node instanceof Element
 Element element = (Element) node;
 Student stud = createStudent(element);
 super.entities.add(stud);
 }
 }
}

private Student createStudent(Element element) {
 String id = element.getAttributeNode("id").getValue();
 String firstName = element.getElementsByTagName("firstName").item(0).getTextContent();
 String lastName = element.getElementsByTagName("lastName").item(0).getTextContent();
 String email = element.getElementsByTagName("email").item(0).getTextContent();
 return new Student(id, firstName, lastName, email);
}
```

# Dom parser writeToFile()

```
public void writeToFile() {
 try {
 DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
 DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
 // root element
 Document doc = docBuilder.newDocument();
 Element rootElement = doc.createElement("students");
 doc.appendChild(rootElement);
 findAll().forEach(x->{
 Element studentElement = doc.createElement("student");
 studentElement.setAttribute("id", x.getId());
 appendStudentElement(doc, "firstName", x.getFirstName(), studentElement);
 appendStudentElement(doc, "lastName", x.getLastName(), studentElement);
 appendStudentElement(doc, "email", x.getEmail(), studentElement);
 rootElement.appendChild(studentElement);
 });
 saveDocument(doc);
 } catch (ParserConfigurationException pce) {
 pce.printStackTrace();
 }
}

private static void appendStudentElement(Document doc,
String tagName, String textNode, Element studentElem)
{
 Element element=doc.createElement(tagName);
 element.appendChild(doc.createTextNode(textNode));
 studentElem.appendChild(element);
}
```



# Dom parser save document

```
void saveDocument(Document doc) {
 // write the content into xml file
 DOMSource source = new DOMSource(doc);
 StreamResult result = new StreamResult(new File(super.fileName));

 TransformerFactory transformerFactory = TransformerFactory.newInstance();
 Transformer transformer = null;
 try {
 transformer = transformerFactory.newTransformer();
 } catch (TransformerConfigurationException e) {
 e.printStackTrace();
 }

 try {
 transformer.transform(source, result);
 } catch (TransformerException e) {
 e.printStackTrace();
 }
 System.out.println("File saved!");
}
```

# StAX Parser

```
public void loadData() {
 try (InputStream input = new FileInputStream(super.fileName)) {
 XMLInputFactory inputFactory = XMLInputFactory.newInstance();
 XMLStreamReader reader = inputFactory.createXMLStreamReader(input);
 readFromXml(reader);
 } catch (IOException | XMLStreamException f) {
 }
}

private void readFromXml(XMLStreamReader reader) throws XMLStreamException {
 Student st = null;
 while (reader.hasNext()) {
 int event = reader.next();
 switch (event) {
 case XMLStreamReader.START_ELEMENT:
 if (reader.getLocalName().equals("student")) {
 //citim pana la Start_Element student
 st = citesteStudent(reader);
 entities.add(st);
 }
 break;
 }
 }
}
```

```
<students>
 <student id="1">
 <firstName value="Popescu"/>
 <lastName value="Stanila"/>
 <emailName value="sasa@sasas"/>
 </student>
</students>
```

# StAX Parser

```
private Student citesteStudent(XMLStreamReader reader) throws XMLStreamException {
 String id = reader.getAttributeValue(null, "id");
 Student s = new Student();
 s.setId(id);
 String currentPropertyValue = "";
 while (reader.hasNext()) {
 int event = reader.next();
 switch (event) {
 case XMLStreamReader.END_ELEMENT:
 if (reader.getLocalName().equals("student")) {
 return s;
 }
 else {
 if (reader.getLocalName().equals("firstName")) {
 s.setFirstName(currentPropertyValue);
 }
 if (reader.getLocalName().equals("lastName")) {
 s.setLastName(currentPropertyValue);
 }
 if (reader.getLocalName().equals("email")) {
 s.setEmail(currentPropertyValue);
 }
 currentPropertyValue = "";
 }
 }
 break;
 }
 case XMLStreamReader.CHARACTERS:
 currentPropertyValue = reader.getText().trim();
 break; } } throw new XMLStreamException("nu s-a citit student"); }
```

```
<students>
 <student id="1">
 <firstName value="Popescu"/>
 <lastName value="Stanila"/>
 <emailName value="sasa@sasas"/>
 </student>
</students>
```

# StAX Parser

@Override

```
public void writeToFile() {
 XMLOutputFactory factory = XMLOutputFactory.newInstance();
 try {
 XMLStreamWriter streamWriter =
 factory.createXMLStreamWriter(new FileOutputStream(super.fileName));
 streamWriter.writeStartElement("students");
 super.entities.forEach(x -> {
 try {
 writeStudentInFile(x, streamWriter);
 } catch (XMLStreamException e) {
 e.printStackTrace();
 }
 });
 streamWriter.writeEndElement();
 } catch (XMLStreamException e) {
 e.printStackTrace();
 } catch (FileNotFoundException e) {
 e.printStackTrace();
 }
}
```

# StAX Parser

```
public void writeStudentInFile(Student x, XMLStreamWriter writer) throws XMLStreamException{
 writer.writeStartElement("student");
 writer.writeAttribute("id",x.getId());
 writer.writeStartElement("firstName");
 writer.writeAttribute("value",x.getFirstName());
 writer.writeEndElement();
 writer.writeStartElement("lastName");
 writer.writeAttribute("value",x.getLastName());
 writer.writeEndElement();
 writer.writeStartElement("email");
 writer.writeAttribute("value",x.getEmail());
 writer.writeEndElement();
 writer.writeEndElement();
}
```