

Metode avansate de programare

Informatică Româna, 2019-2020, Curs 6

- XML
- Reflexia (Reflection)
- GUI – intro (cont in Curs 7)

Ce este un document XML?

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

Structura unui document xml

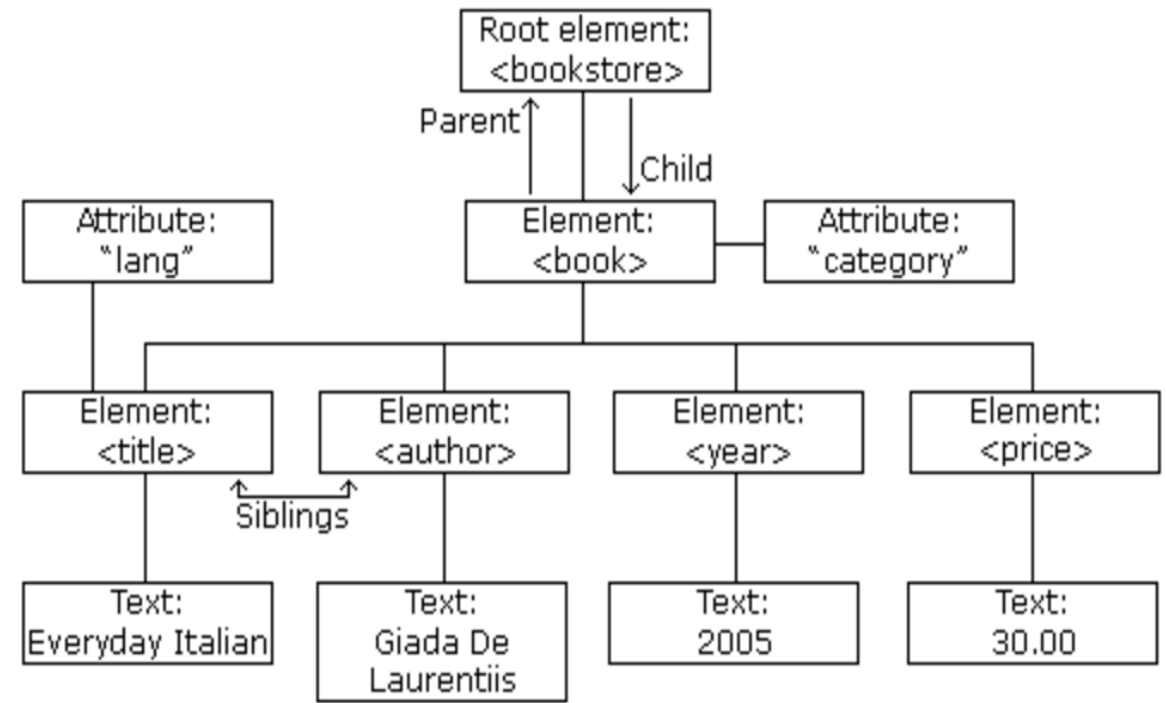
Un fisier XML cuprinde urmatoarele sectiuni:

- **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> si <title> are cate un atribut (category="children", lang="en").

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>
```

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
 - <http://homepage.divms.uiowa.edu/~slonnegr/xml/02.DTDs.pdf>
 - <https://www.w3.org/TR/xmlschema-1/#Annotation>

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

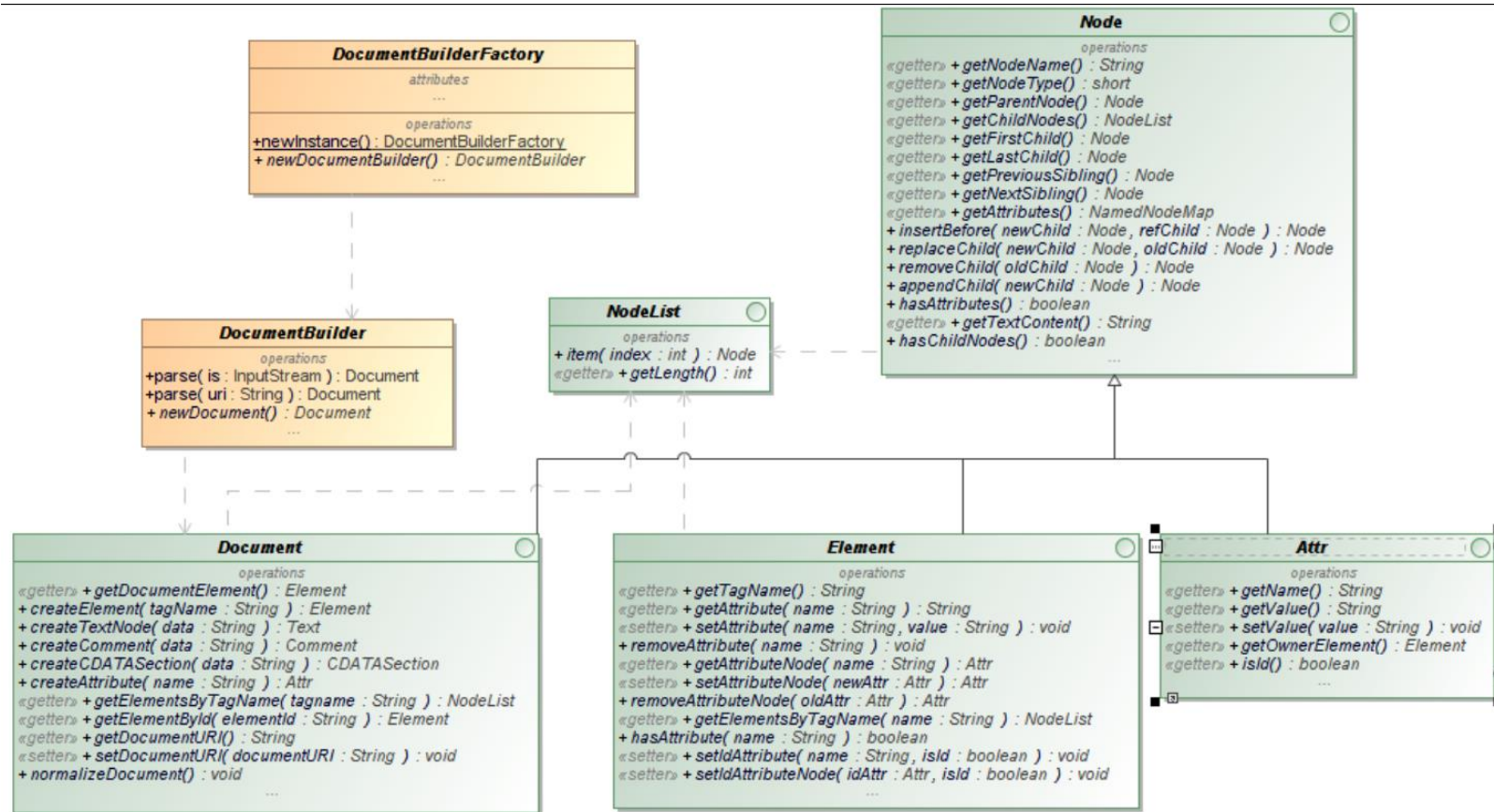
- Parserul SAX este un API bazat pe evenimente. De obicei, un API bazat pe evenimente furnizează interfețe de tipul handler. Există patru interfețe handler: *ContentHandler*, *DTDHandler*, *EntityResolver* și *ErrorHandler*.

StAX Parser – Parsează un document XML într-un mod foarte asemănător cu SAX, dar 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/>



Exemplu curs folosind parserul DOM

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
  <student id="1">
    <nume>Harry Potter</nume>
    <grupa>221</grupa>
  </student>
  <student id="2">
    <nume>Harry Potter 2</nume>
    <grupa>223</grupa>
  </student>
</students>
```

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

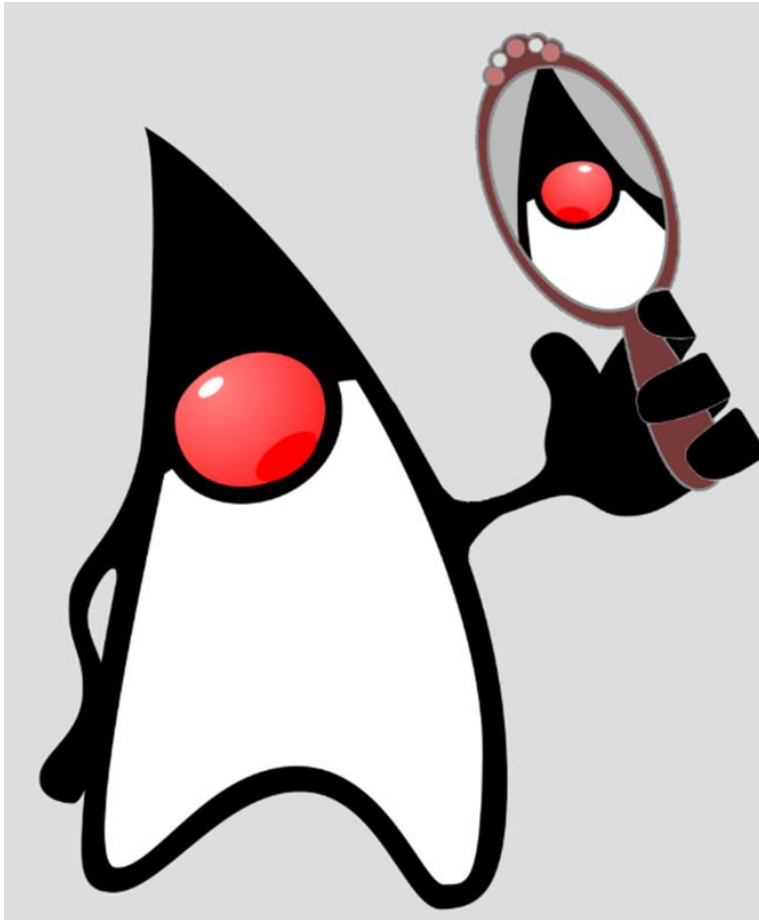
@Override

```
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();
}
```

Reflection



Introspecția =
Capacitatea unui
program de a-și observa,
la execuție, propria
structură;

Încărcarea claselor în memorie

- Un program Java compilat este descris de o mulțime de fișiere cu extensia *.class* corespunzătoare fiecărei clase a programului.
- Aceste clase nu sunt încărcate toate în memorie la pornirea aplicației, ci sunt încărcate pe parcursul execuției acestuia, *atunci când este nevoie de ele*, momentul efectiv în care se realizează acest lucru depinzând de implementarea mașinii virtuale (JVM).
- Încărcarea claselor unei aplicații Java în memorie este realizată prin intermediul unor obiecte, denumite generic *class loader*.

Încarcarea **dinamică** a claselor în memorie

- Se refera la faptul ca nu cunoastem tipul acesteia decat la executia programului, moment in care putem solicita încarcarea sa, specificand numele său complet prin intermediul unui șir de caractere.
- Exista mai multe modalitati:
 - *loadClass* apelata pentru un obiect de tip `ClassLoader`
 - *Class.forName*

Observație

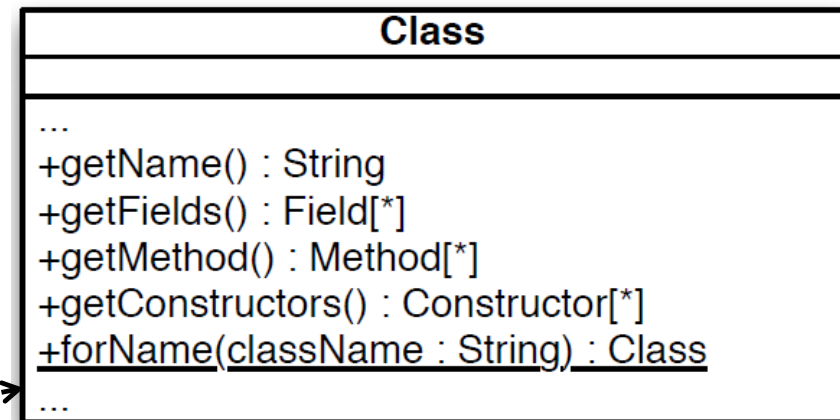
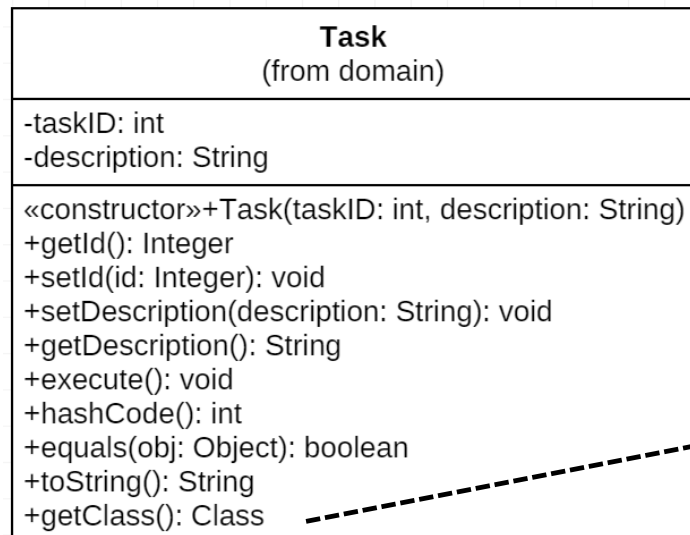
- În programarea orientată pe obiecte noțiunea de clasă și obiect sunt evident diferite
- În mecanismul de introspectie este doar o chestiune de reprezentare și nu se schimbă aceste noțiuni;
 - O clasă din program este reprezentată de un obiect și, ca orice obiect, e definit de o clasă (meta-clasă) în speță, clasa Class.

Încarcarea unei clase în memorie – meta clasa Class

Class.forName

Exemplu: clasa Task este reprezentată la execuția unui program de un obiect, instanță a clasei *Class*

```
Class aCls =Class.forName("domain.Task");  
System.out.println(aCls.getName());
```



- un obiect instanța a clasei *Class* reprezintă o clasă din cadrul programului

Exemplu Reflectarea unei clase

```
public static void reflectClass(Class aClass) {
    String lines = "";
    lines += String.format("Class " + aClass.getName() + " having the following members:\n");

    for (Field aField : aClass.getDeclaredFields()) {
        lines += String.format("\tField name - %s :%s\n", aField.getName(), aField.getType());
    }
    for (Method aMethod : aClass.getDeclaredMethods()) {
        lines += String.format("\tMethod name - %s (): %s\n", aMethod.getName(), aMethod.getReturnType().getName());
        Parameter[] param = aMethod.getParameters();
        for (int i = 0; i < param.length; i++) {
            lines += String.format("\t\tParam %d - %s:%s\n", i + 1, param[i].getName(), param[i].getType());
        }
    }
    for (Constructor aConstructor : aClass.getConstructors()) {
        lines += String.format("\tConstructor name - %s:", aConstructor.getName());
        for (int i = 0; i < aConstructor.getParameters().length; i++) {
            Parameter param = aConstructor.getParameters()[i];
            lines += String.format("\t\tParam - %d: %s :%s\n", i + 1, param.getName(), param.getType());
        }
    }
    System.out.println(lines);
}
```

Task Mirror ☺

Class domain.Task having the following members:

```
Field name - taskID :int
Field name - description :class java.lang.String
Method name - equals (): boolean
    Param 1 - arg0:class java.lang.Object
Method name - toString (): java.lang.String
Method name - hashCode (): int
Method name - execute (): void
Method name - getId (): java.lang.Object
Method name - getId (): java.lang.Integer
Method name - setId (): void
    Param 1 - arg0:class java.lang.Integer
Method name - setId (): void
    Param 1 - arg0:class java.lang.Object
Method name - getDescription (): java.lang.String
Method name - setDescription (): void
    Param 1 - arg0:class java.lang.String
Method name - wait (): void
Method name - wait (): void
    Param 1 - arg0:long
    Param 2 - arg1:int
Method name - wait (): void
    Param 1 - arg0:long
Method name - getClass (): java.lang.Class
Method name - notify (): void
Method name - notifyAll (): void
Constructor name - domain.Task:      Param - 1: arg0 :int
    Param - 2: arg1 :class java.lang.String
```

Exercitii curs

- Instantierea unui obiect de un anumit tip folosind Reflection