

Entrée Sorties en Java

Karima Ennaoui

`karima.ennaoui@insa-cvl.fr`

INSA CVL, 4A STI

Webographie: Lesson: Basic I/O, The Java Tutorials, Oracle Corporation.
Inspiré du cours de Jean-François Lalande

Monday 7th January, 2019

Plan:

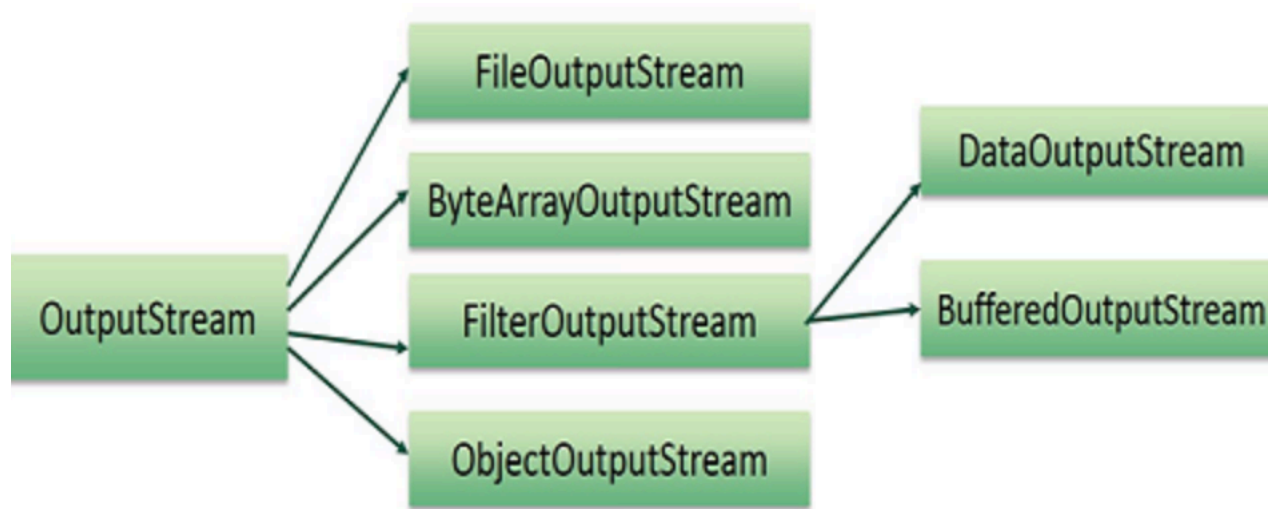
- Streams
- Scanners
- S rialisation
- Parseurs des fichiers XML

Streams

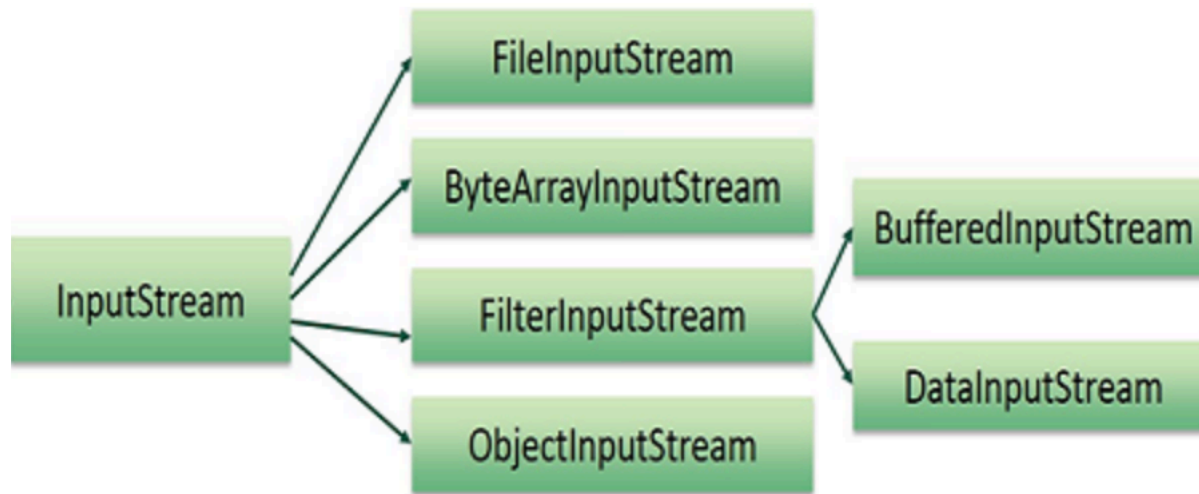
Définition

- ✱ Stream = une séquence de données, à lire ou à écrire.
- ✱ Différents types de sources/destinations: fichiers, supports externes, autres programmes...
- ✱ Différentes représentations de données: bytes, caractères, données primitives, objets...
- ✱ Tous ses classes héritent de la classe ***Reader*** et implémentent l'interface ***Readable*** du package ***java.io***

Classes Java d'output Streams



Classes Java d'input Streams



Examples

- Octet Classes: `FileInputStream`, `FileOutputStream`

```
FileInputStream in = null;
FileOutputStream out = null;
try { in = new FileInputStream("xanadu.txt");
    out = new FileOutputStream("outagain.txt");
    int c;
    while ((c = in.read()) != -1) {
        out.write(c);
    } catch ...
```

Exemples

- Octet **Classes: FileInputStream, FileOutputStream**
- Caractère: Stocké en Unicode et projeté dans le jeu de caractère local.
Classes: FileReader , FileWriter

```
FileReader inputStream = new FileReader("xanadu.txt");  
FileWriter outputStream = new FileWriter("characteroutput.txt");  
// code identique au précédent pour lecture/écriture
```


Exemples

- Octet Classes: `FileInputStream`, `FileOutputStream`
- Caractère: Stocké en Unicode et projeté dans le jeu de caractère local.
Classes: `FileReader` , `FileWriter`
- Ligne: Le flux découpé à chaque *carriage-return* ("r"), ou *line-feed* ("n").
Lecture plus optimale en performance grace au buffer.
Classes: `BufferedReader`, `PrintWriter`

```
BufferedReader inputStream = null;
PrintWriter outputStream = null;

try { inputStream = new BufferedReader(new FileReader("xanadu.txt"));
    outputStream = new PrintWriter(new FileWriter("characteroutput.txt"));

    String l;
    while ((l = inputStream.readLine()) != null) {
        outputStream.println(l);
    }
}
```

Scanners

Définitions

- ✱ Classe ***Scanner*** permet de découper l'input en tokens.
- ✱ Par défaut, un objet scanner considère que les tokens sont séparés par des espaces.

```
import java.io.*;
import java.util.Scanner;

public class ScanXan {
    public static void main(String[] args) throws IOException {

        Scanner s = null;
        try {s = new Scanner(new BufferedReader(new
                                FileReader("xanadu.txt")));

            while (s.hasNext()) {
                System.out.println(s.next());
            }
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

Changement de séparateur du scanner

- ✱ On peut à tout moment changer le séparateur de tokens en utilisant `useDelimiter()` qui prend en entrée une expression régulière.

Exemple: `myScanner.useDelimiter(",\\s*")`.

Scanner par type primitif

- ✱ Les entrées sorties formatées permettent de découper le flux en mots projetés directement dans le type adéquat.

Exemple: `myScanner.hasNextDouble()`, `myScanner.nextDouble()`

Spécifications locales

- ✱ La méthode ***useLocale()*** de scanner permet de préciser à quelle spécification locale les données traitées appartiennent.

Exemple: En traitant des doubles, "32,767" et "8.5" seraient interprétés différemment si je précise:

myScanner.useLocale(Local.US) ou *myScanner.useLocale(Local.FR)*

Lecture et Ecriture de la sortie standard

✱ La plateforme Java supporte trois interactions I/O standards via la classe System:

- System.in implémentant InputStream
- System.out implémentant OutputStream
- System.err implémentant OutputSrteam

Lecture et Ecriture de la sortie standard

✱ La plateforme Java supporte trois interactions I/O standards via la classe System:

- System.in implémentant InputStream
- System.out implémentant OutputStream
- System.err implémentant OutputSrteam

✱ Pour Ecrire:

System.out.println(String), System.out.print(char/boolean....)...

Lecture et Ecriture de la sortie standard

- ✱ La plateforme Java supporte trois interactions I/O standards via la classe System:

- System.in implémentant InputStream
- System.out implémentant OutputStream
- System.err implémentant OutputSrteam

- ✱ Pour Ecrire:

`System.out.println(String), System.out.print(char/boolean....)...`

- ✱ Pour Lire:

`Scanner monScanner=new Scanner(System.in); -> monScanner.nextLine();.....`

Lecture et Ecriture de la sortie standard

✱ Une autre méthode sympa:

```
Console c = System.console();  
if (c == null) {  
    System.err.println("No console.");  
    System.exit(1); }  
char [] p = c.readPassword("Enter your password: ");
```

Lecture et Ecriture de la sortie standard

✱ Une autre méthode sympa:

1. Demander la console

2. Si le programme n'est pas attaché à une console ou qu'il n'est pas en interactivité avec une console.

```
Console c = System.console();  
if (c == null) {  
    System.err.println("No console.");  
    System.exit(1); }  
char [] p = c.readPassword("Enter your password: ");
```

3. L'affichage des caractères est désactivé dans la console.

Sérialisation

Problématique

- Supposant qu'on écrit un programme de gestion des adhérents d'un club.
- La listes de ces adhérents doit persister à la fin du programme.
- Pourtant, La portée des objets est limité au bloc de sa déclaration.
- La solution: sauvegarder dans des fichiers. Mais...

Problématique

- Supposant qu'on écrit un programme de gestion des adhérents d'un club.
- La listes de ces adhérents doit persister à la fin du programme.
- Pourtant, La portée des objets est limité au bloc de sa déclaration.
- La solution: sauvegarder dans des fichiers. Mais...
- Stocker un objet implique stocker ses attributs.
- Le même mécanisme doit être implémenté par toutes les classes en question.
- Gestion des références compliquée.

Concept de la sérialisation

- Solution Java pour la persistance des données.
- Sauvegarder et restaurer des objets dans un seul flux de données.
- Gestion des références en sauvegardant tous les objet référencés une seule fois.
- Existe depuis les débuts de l'API I/O.

Conditions de la sérialisation

- Implémentation de l'interface ***Serializable***:
 - ↪ Interface sans méthodes
 - ↪ Drapeau pour indiquer que les objets de cette classe peuvent être sérialisé (question de sécurité)
- Tous les attributs de la classe doivent également être sérialisable.
- Une super-classe non-serialisable, si elle existe, doit disposer d'un constructeur accessible sans paramètres.

Conditions de la sérialisation

- Pas toutes les classes JDK sont sérialisables: Thread, OutputStream et ses sous-classes, Socket, Image, ...)
- Les attributs *static* ne sont pas sérialisé.
- Le mot clé *transient* sert à indiquer qu'un objet ne doit pas être sérialisé.

La sérialisation d'un objet

- Un appel de la méthode *writeObject* de la classe *ObjectOutputStream*.
- Tous les attributs, sauf si transient ou static, sont sérialisés.

```
import java.io.FileNotFoundException; import java.io.FileOutputStream;
import java.io.IOException; import java.io.ObjectOutputStream;
import java.io.Serializable;
public class PersonneIOMainWrite implements Serializable {
    public static void main(String[] args) {
        PersonneIO p = new PersonneIO();
        p.setNom("JFL");
        PersonneIO p2 = new PersonneIO();
        p2.setAdresse("?");
        FileOutputStream fos;
        try { fos = new FileOutputStream("file.out");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(p);
            oos.writeObject(p2);
            oos.close(); }
        catch (FileNotFoundException e) { e.printStackTrace(); }
        catch (IOException e) { e.printStackTrace(); }
    }}
```

La dé-sérialisation d'un objet

- Un appel de la méthode *readObject* de la classe *ObjectInputStream*.
- Type de retour est *object*. Donc, il faut savoir le type de l'objet restauré.

Possible d'utiliser la méthode *getClass()*.

```
package io;
import java.io.FileInputStream; import java.io.FileNotFoundException;
import java.io.IOException; import java.io.ObjectInputStream;
import java.io.Serializable;
public class PersonneIOMainRead implements Serializable {
    public static void main(String[] args) {
    try { FileInputStream is = new FileInputStream("file.out");
        ObjectInputStream in = new ObjectInputStream(is);
        PersonneIO p = (PersonneIO)in.readObject();
        PersonneIO p2 = (PersonneIO)in.readObject();
        System.out.println(p.nom + " habite " + p.m.adresse);
        System.out.println(p2.nom + " habite " + p2.m.adresse);
        in.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    }}
```

Notion de serialVersionUID

- Un identifiant unique de sérialisation calculé à la compilation: chaque instance sérialisée possède cet identifiant.
- Il est ou bien explicitement dans la classe de cet objet, ou bien la machine Java en détermine un, en fonction des éléments publics et non statiques de la classe.
- Cet ID empêche le chargement d'un objet sérialisé ne correspondant plus à une nouvelle version de la classe correspondante.
- L'exception particulière qui peut être levée est **ClassNotFoundException** dans le cas où la JVM ne trouve pas la définition de la classe à charger.

La sérialisation sélective

- Création de ces deux méthodes dans la classe à sérialiser (en respectant la signature):

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException;
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
```

- Lorsque la machine Java constate qu'une classe Sérialisable comporte ces deux méthodes, alors elle les appelle plutôt que d'utiliser ses mécanismes internes de sérialisation.
- Il est important de noter que ces deux méthodes fonctionnent en tandem, et doivent donc être compatibles l'une avec l'autre.

La sérialisation d'un objet sélective

```
public class Marin implements Serializable {

    private String nom, prenom ;
    private int salaire ;

    // suivent les getters / setters

    // méthode readObject, utilisée pour reconstituer un objet sérialisé
    private void readObject(ObjectInputStream ois)
    throws IOException, ClassNotFoundException {

        // l'ordre de lecture doit être le même que l'ordre d'écriture d'un objet
        this.nom = ois.readUTF() ;
        this.prenom = ois.readUTF() ;
        // le salaire n'est pas relu, vu qu'il n'a pas été écrit
    }

    // méthode writeObject, utilisée lors de la sérialization
    private void writeObject(ObjectOutputStream oos)
    throws IOException {

        // écriture de toute ou partie des champs d'un objet
        oos.writeUTF(nom) ;
        oos.writeUTF(prenom) ;
        // on choisit de ne pas écrire le salaire, qui ne fait
        // pas partie de l'état d'une instance de marin
    }
}
```

La sérialisation des types primitives

➤ Projection directe en utilisant *DataOutputStream*

```
static final double[] prices = { 19.99, 9.99, 15.99, 3.99, 4.99 };
static final int[] units = { 12, 8, 13, 29, 50 };
static final String[] descs = { "Java T-shirt", "Java Mug" };

DataOutputStream out = new DataOutputStream(
    new BufferedOutputStream(new FileOutputStream(dataFile)));
for (int i = 0; i < prices.length; i++) {
    out.writeDouble(prices[i]);
    out.writeInt(units[i]);
    out.writeUTF(descs[i]);
}
```

La dé-sérialisation des types primitives

➤ Projection directe en utilisant *DataInputStream*

```
DataInputStream in = new DataInputStream(  
    new BufferedInputStream(new FileInputStream(dataFile)));  
  
double price; int unit; String desc; double total = 0.0;  
  
try { while (true) {  
    price = in.readDouble();  
    unit = in.readInt();  
    desc = in.readUTF();  
    System.out.format("You ordered %d units of %s at $%.2f%n",  
        unit, desc, price);  
    total += unit * price;  
    }  
} catch (EOFException e) { }
```


Concept de Java Beans

- Java Bean est la brique d'une architecture logicielle dont le but est de simplifier la réutilisation du code sans devoir comprendre son fonctionnement interne.

Exemple: Éléments graphiques de Swing sur NetBeans

Caractéristiques d'un Bean

Pour pouvoir être qualifié de bean, un objet doit:

- être sérialisable pour pouvoir sauvegarder et restaurer son état ;
- disposer d'un constructeur sans argument ou par défaut qui soit publiquement accessible ;
- encapsuler l'accès à ses membres via des méthodes *getters* et *setters* (ce qui forme ce qu'on appelle une **propriété**).
- Suivre des conventions de nommage adoptées pour les getters et setters ainsi que les méthodes de gestion d'évènements, permettant de retrouver aisément les propriétés du *bean* par introspection.

Sérialisation d'un Bean

- En addition aux méthodes classiques, un java bean peut être sérialiser en utilisant les classes *java.bean.XMLEncoder* et *java.bean.XMLDecoder*.
- La classe ne doit pas forcément être déclarée sérialisable.
- Cette méthode ne sauvegarde que les propriétés qui répondent aux normes du bean.
- La sérialisation sous formes de fichiers XML.
- La manière de procéder est similaire à celle de la sérialisation et de désérialisation classique.

Sérialisation d'un Bean

```
XMLEncoder encoder = new XMLEncoder(  
    new BufferedOutputStream(  
        new FileOutputStream("Beanarchive.xml")));  
  
encoder.writeObject(object);  
encoder.close();
```

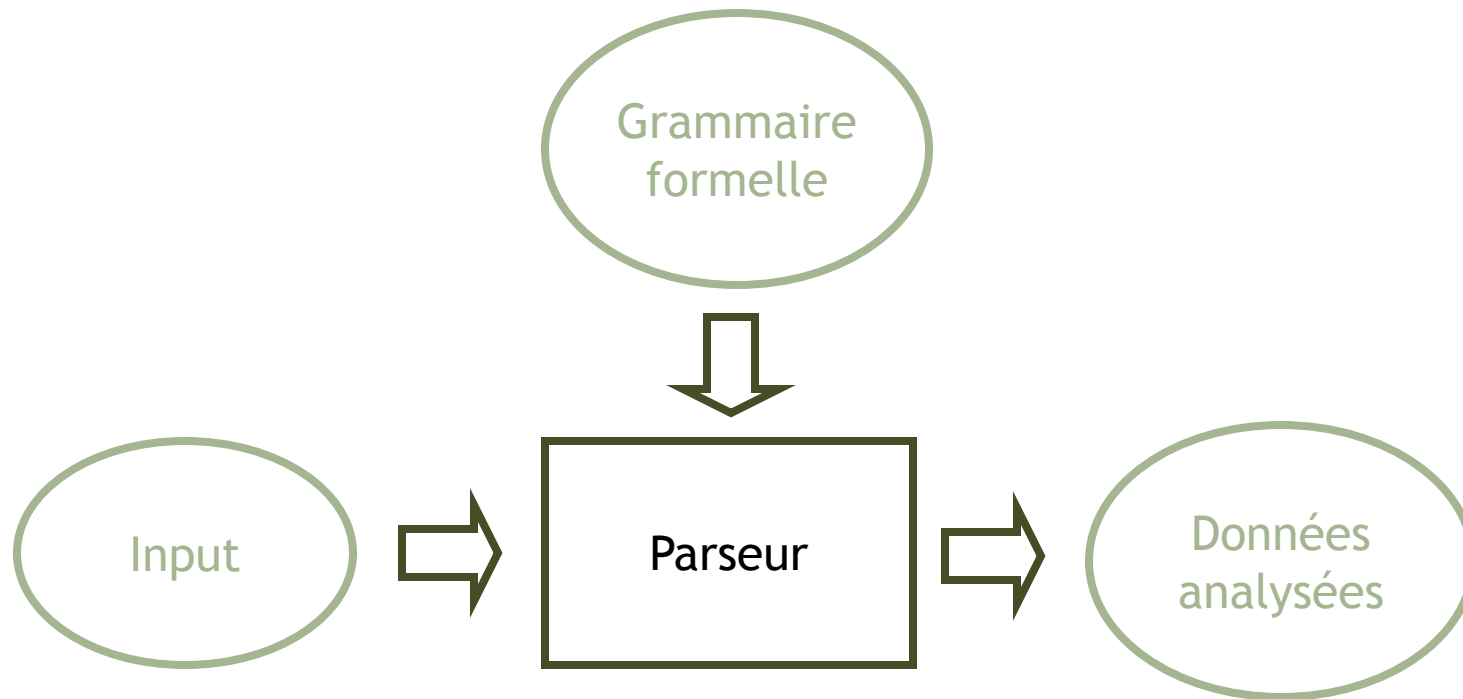
Dé-sérialisation d'un Bean

```
XMLDecoder decoder = new XMLDecoder(  
    new BufferedInputStream(  
        new FileInputStream("Beanarchive.xml")));
```

```
Object object = decoder.readObject();  
decoder.close();
```

Parseur de fichiers XML

Qu'est ce qu'un parseur?



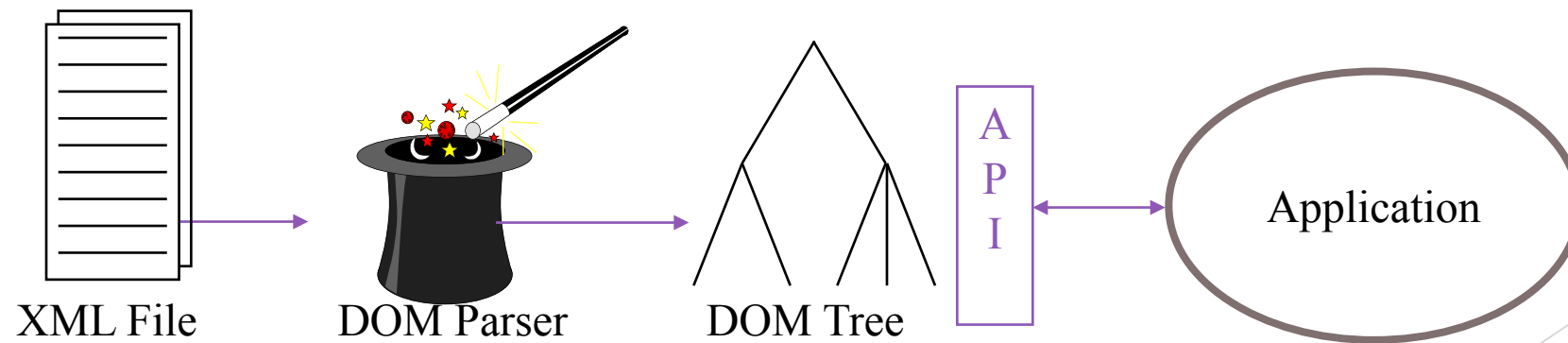
Types de parseurs

Il existe deux grandes familles de parseurs de fichiers XML en Java:

- ▶ Les parseurs évènementiels: *SAX*
- ▶ Les parseurs à parcours d'arbres: *DOM*

DOM

- ▶ DOM = Document Object Model
- ▶ Parseur qui crée un **arbre** d'objet à partir du fichier XML en input.
- ▶ L' API permet la construction, l'accès ainsi que la manipulation la structure et le contenu des fichiers XML.



DOM: Création de l'arbre

- ▶ L'arbre est généré par la méthode **parse()** d'un objet de type **DocumentBuilder**:

```
Document doc=builder.parse("myFile.XML");
```

- ▶ Le **DocumentBuilder** est géré via un design pattern Factory:

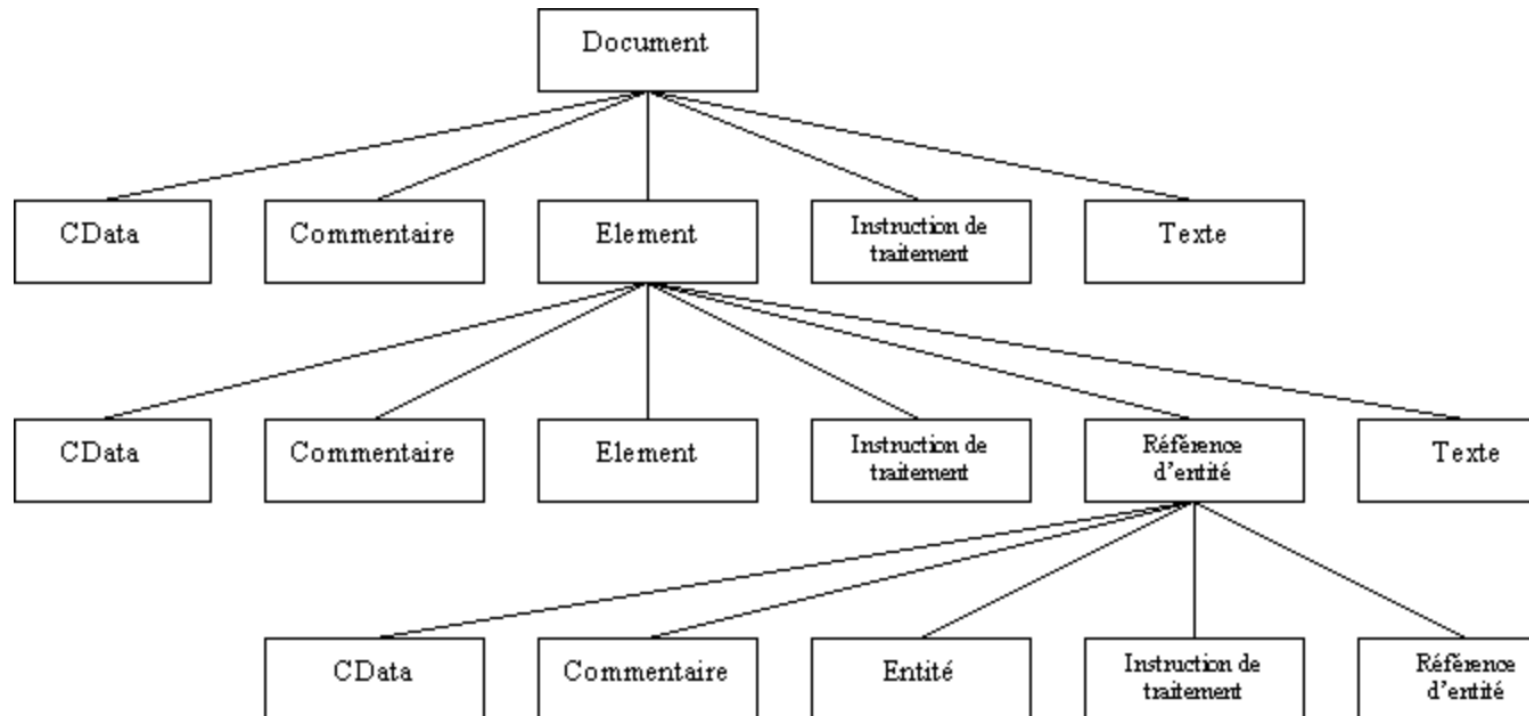
```
DocumentBuilderFactory factory= DocumentBuilderFactory.newInstance();
```

```
DocumentBuilder builder=factory.newDocumentBuilder();
```

- ▶ Le **DocumentBuilderFactory** permet également de paramétrer l'objet **DocumentBuilder** utilisé. Exemple: **factory.setIgnoringComments(false)**

DOM: Manipulation de l'arbre

- Chaque nœud de l'arbre DOM implémente l'interface *node*.
- Il existe plusieurs sous-type de *node* (y inclu Document qui représente la racine)

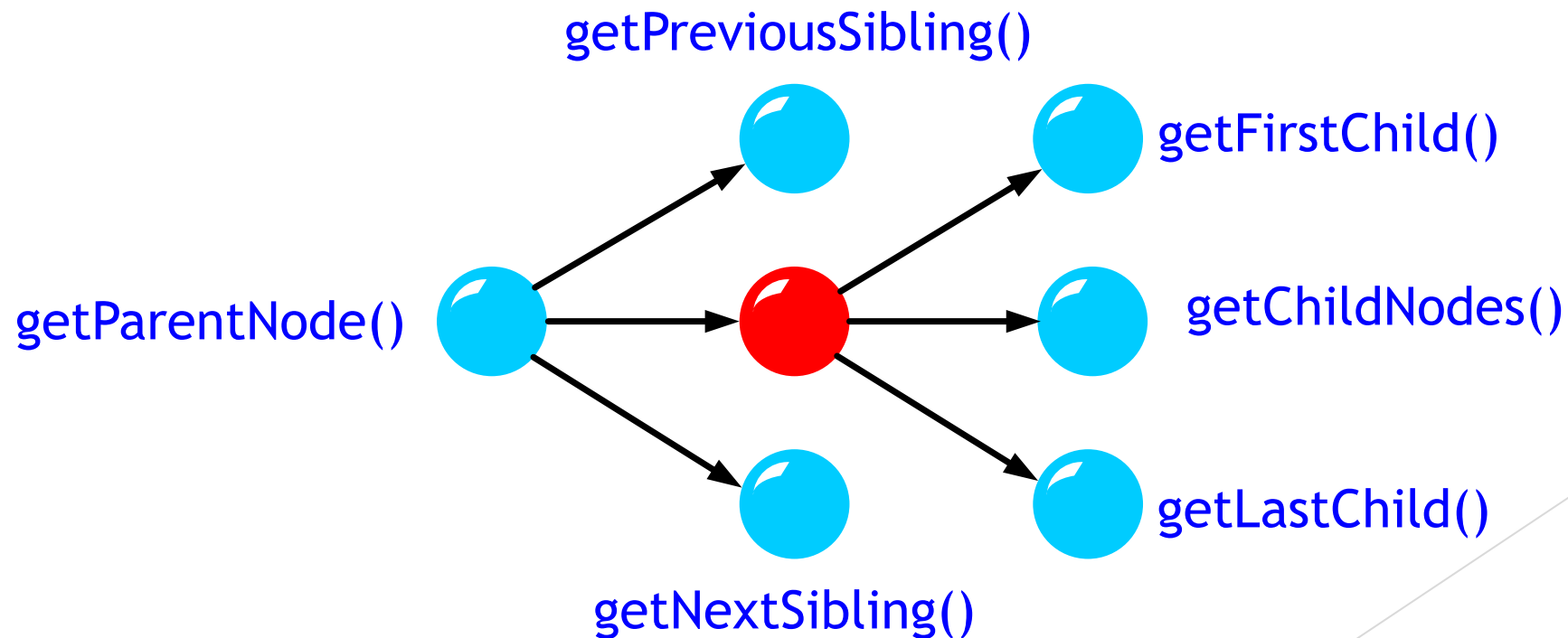


DOM: Manipulation de l'arbre

- ▶ Chaque nœud a ...
 - ▷ Un type
 - ▷ Un nom
 - ▷ Une valeur
 - ▷ Des attribues
- ▶ Le rôle de chacune de ces propriétés dépend du sous-type du nœud.

DOM: Manipulation de l'arbre

- L'interface **Node** implémente des méthodes afin de naviguer l'arbre:




DOM: Manipulation de l'arbre

- ▶ L'interface **Node** implémente des méthodes pour modifier les nœuds de l'arbre:
 - ▷ Pour rajouter des nœuds, on utilise les méthodes: *createElement*, *createAttribute*, *createTextNode*, *createCDATASection* etc...
 - ▷ Pour modifier un noeud, on utilise les méthodes: *appendChild*, *insertBefore*, *removeChild*, *replaceChild*, *setNodeValue*, *cloneNode(boolean deep)* etc...

SAX

- ▶ SAX = Simple API for XML
- ▶ Lecture séquentielle des fichiers XML
- ▶ Le parseur déclenche des *callbacks* lorsque les balises sont rencontrées.
- ▶ Le traitement à faire à chaque «évènement» est géré par le ***handler***.



```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2017">10,248,069</population>
    <city capital="yes"><name>Amman</name></city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```

```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2018">6,348,000</population>
    <city capital="Amman"><name>Amman</name></city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```

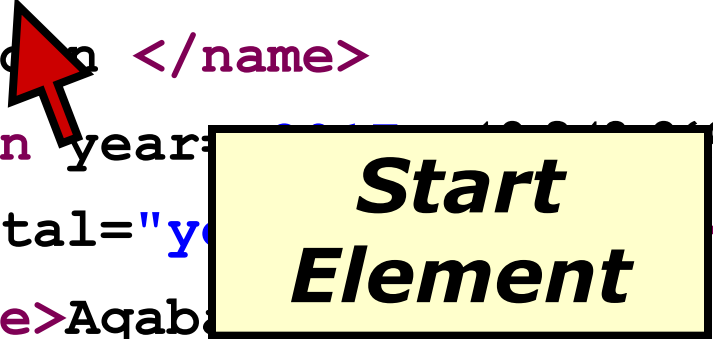
**Start
Document**

```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2018">5,277,000</population>
    <city capital="yes">Amman</name></city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```




**Start
Element**

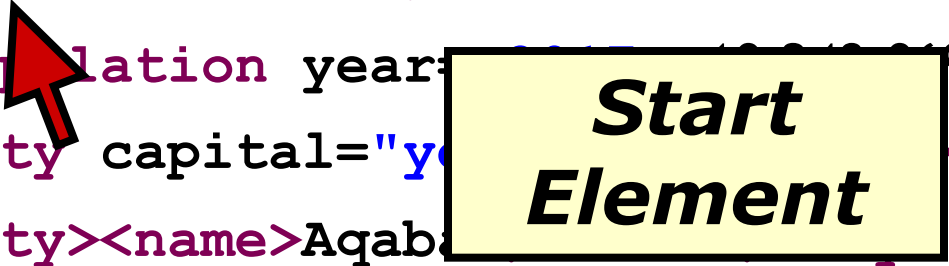
```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2018">6,500,000</population>
    <city capital="yes">Amman</name></city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```



```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2018">5,277,000</population>
    <city capital="yes" name="Amman"></name></city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```

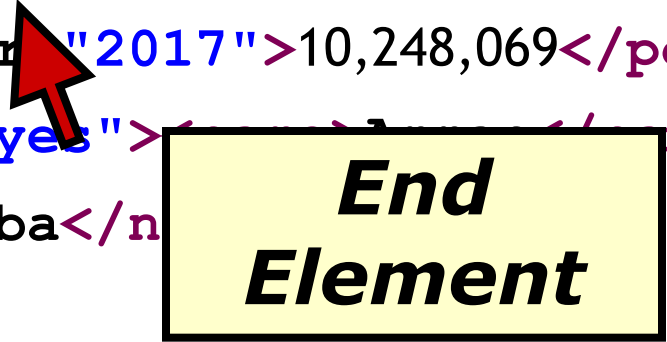


```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2018">5,277,000</population>
    <city capital="yes">Amman</name></city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```


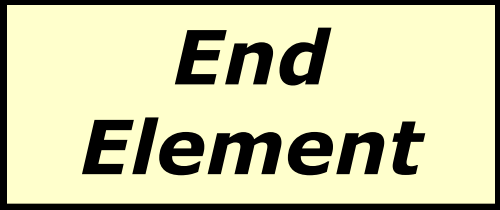


```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2017">10.248.069</population>
    <city capital="<img alt="A red arrow pointing to the word 'Characters' in a yellow box." data-bbox="308 355 340 455"/>Characters</name></city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```


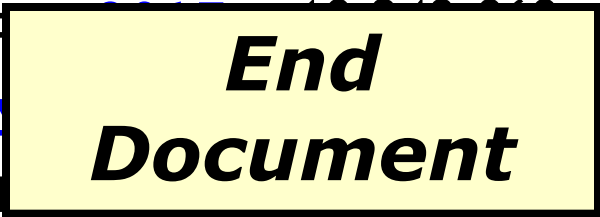
```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2017">10,248,069</population>
    <city capital="yes">Amman</city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```



```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2017">10,248,060</population>
    <city capital="yes">Amman</city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```




```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name> Jordan </name>
    <population year="2018">6,500,000</population>
    <city capital="Amman" name="Amman"></city>
    <city><name>Aqaba</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2018">67,348,000</population>
  </country>
</countries>
```



SAX: Instanciation du parseur

```
import org.xml.sax.*;
```

```
public class ReadWithSax {
```

```
    public static void main(String[] args) throws Exception {
```

```
        SAXParserFactory factory= SAXParserFactory.newInstance();
```

```
        SAXParser parser= factory.newSAXParser();
```

```
        ....
```

SAX: Instanciation du parseur

```
import org.xml.sax.*;
```

```
public class ReadWithSax {
```

```
    public static void main(String[] args) throws Exception {
```

```
        SAXParserFactory factory= SAXParserFactory.newInstance();
```

```
        SAXParser parser= factory.newSAXParser();
```

```
        ....
```

SAX: Instanciation du parseur

```
import org.xml.sax.*;
```

```
public class ReadWithSax {
```

```
    public static void main(String[] args) throws Exception {
```

```
        SAXParserFactory factory= SAXParserFactory.newInstance();
```

```
        SAXParser parser= factory.newSAXParser();
```

```
        ....
```

SAX: Implémentation du handler

- ▶ La classe ***DefaultHandler*** implémente toutes les interfaces Handler, i.e., ***ContentHandler***, ***EntityResolver***, ***DTDHandler***, ***ErrorHandler***
- ▶ Le plus simple est de commencer par une extension de la classe ***DefaultHandler***:

```
public class mySAXHandler extends DefaultHandler{...}
```

- ▶ Les méthodes à implémenter: ***startDocument***, ***endDocument***, ***startElement***, ***endElement***....

SAX: Lecture du fichier XML

```
import org.xml.sax.*;

public class ReadWithSax {

    public static void main(String[] args) throws Exception {

        SAXParserFactory factory= SAXParserFactory.newInstance();

        SAXParser parser= factory.newSAXParser();

        MySaxHandler handler= new MySaxHandler();

        Parser.parse(new File(myFile.xml),handler);

    }
```

SAX: Lecture du fichier XML

```
import org.xml.sax.*;

public class ReadWithSax {

    public static void main(String[] args) throws Exception {

        SAXParserFactory factory= SAXParserFactory.newInstance();

        SAXParser parser= factory.newSAXParser();

        MySaxHandler handler= new MySaxHandler();

        Parser.parse(new File(myFile.xml),handler);

    }
```

SAX: Lecture du fichier XML

- Au moment de la récupération d'une balise, ses attributs et leurs valeurs peuvent être récupérés dans un objet de type **Attributes**:

```
public void startElement(String uri, String localName,  
                        String qName, Attributes attributes)  
    throws SAXException {  
    System.out.println("qname = " + qName);  
    System.out.println("Attribut: " + attributes.getValue(0));  
}
```


SAX Vs .DOM

- ▶ Si ton document est volumineux et que tu n'as besoin que de quelques éléments, choisis SAX
- ▶ Si tu as besoin de changer le document, choisis DOM
- ▶ Si tu dois accéder au fichier plusieurs fois, DOM est ton choix
- ▶ En général, DOM reste plus facile à manipuler que SAX...
- ▶ Mais après tout, le choix dépend aussi de ton application!