

# Techniques de programmation

## Partie 1

1ère Master Ingénieurs Industriels en Informatique (M18)  
Haute Ecole de la Province de Liège (HEPL)

Ludovic Kuty <ludovic.kuty@hepl.be>

2014 – 2015

### Introduction

---

Le travail est divisé en 10 fonctionnalités (ou étapes) numérotées de 1 à 10 indiquées sur la figure 1, page 6. On vous conseille de les réaliser dans l'ordre pour une question de facilité d'implémentation et d'organisation par rapport au cours théorique.

On vous demande d'écrire un programme en Java qui fonctionne exclusivement en ligne de commande. Il sera exécuté à partir d'un shell quel que soit l'OS utilisé. Vous pouvez le développer à l'aide de Netbeans mais son exécution sera testée en ligne de commande.

Vous utiliserez la librairie `Argparse4j`<sup>1</sup> pour analyser les arguments transmis en ligne de commande à votre programme. Prêtez une attention particulière à la manière d'indiquer des arguments vrai/faux, c'est-à-dire dont on détecte la présence ou l'absence.

Le programme Java devra pouvoir automatiquement exécuter toutes les étapes en séquence sans s'arrêter. L'arrêt peut être provoqué en raison d'une erreur de traitement XML, d'arguments obligatoires manquants ou de toute autre erreur grave. On s'attend à disposer d'un message clair.

On doit être en mesure de suivre l'exécution des étapes sur la console. Veillez donc à afficher suffisamment d'informations.

### Mesures des temps d'exécution

---

On vous demande lors de certaines étapes de votre programme de mesurer le temps d'exécution ("T" majuscule sur le schéma). Vous pouvez utiliser `System.nanoTime()` pour ce faire. On vous demande de fournir un affichage en heures, minutes, secondes, millisecondes ou `< 1 ms` si le temps d'exécution est inférieur à une milliseconde. Bien entendu, vous afficherez après chaque étape concernée son temps d'exécution.

### Espaces de noms

---

Tous les documents XML doivent utiliser les espaces de noms. Cela signifie que vous devez pour chaque vocabulaire XML que vous créez inventer un espace de nom et l'utiliser au sein de vos documents.

---

<sup>1</sup><http://argparse4j.sourceforge.net/>

## 1 - Document XML de configuration - XMLConfig

---

Cette fonctionnalité consiste à écrire un document XML de configuration contenant les informations nécessaires pour pouvoir reprendre des films de la base de données **CI** qui est une collection MongoDB appelée **movies.movies**. Une recherche nécessite de pouvoir indiquer au sein du document de manière exclusive :

- Une liste d'identifiants de films
- Un nombre de films à choisir aléatoirement.

On doit pouvoir indiquer plusieurs recherches dans le document XML. Lors de l'implémentation le résultat de l'exécution de votre programme doit correspondre à ce qu'aurait renvoyé une opération d'union ensembliste (ou logique) sur les résultats renvoyés par les recherches.

On peut aussi effectuer une projection des résultats de manière à filtrer certaines propriétés (champs) qui ne nous intéressent pas. Il y a deux méthodes :

- L'inclusion : On doit pouvoir indiquer les propriétés désirées explicitement ou ne rien indiquer et implicitement les avoir toutes.
- L'exclusion : On doit pouvoir indiquer les propriétés non désirées explicitement.

Si on indique les deux en même temps, alors on retire de ce que doit inclure, ce qu'il faut exclure. Notez qu'on doit aussi pouvoir indiquer des sous-propriétés comme **name** dans le tableau **actors**.

Vous aurez besoin de plusieurs documents XML de configuration de manière à effectuer une série de tests que l'on souhaite exhaustifs.

## 2 - DTD pour le document XML de configuration - DTDDConfig

---

Une fois que vous disposez d'un document XML de configuration, vous pouvez écrire le Document Type Definition (DTD) qui permet de le valider. Testez manuellement la validation avec par exemple Netbeans ou Oxygen.

## 3 - Parsing SaX et validation par DTD - SaX

---

Le programme Java doit pouvoir parser un document XML de configuration et le valider. La parsing permet d'extraire les informations dont nous aurons besoin lors de l'étape suivante. Pensez à utiliser des structures de données adéquates (Map, BitSet, Set, List, ...) pour stocker ces informations.

En cas de problème lors de la vérification de la contrainte de forme ou de la validité du document, vous devez arrêter le traitement et afficher un message clair.

L'argument **--config** permet de choisir le document XML de configuration qui va être utilisé. Si on n'indique pas cet argument, la valeur par défaut est **config.xml** dans le répertoire courant.

## 4 - Construction arbre DOM - DOM

---

Utilisez DOM 2 avec l'API JAXP du JDK pour construire un arbre en RAM qui contient les films demandés par l'intermédiaire du document XML de configuration.

Pour sélectionner efficacement des éléments de manière aléatoire tout en gardant des opérations d'ajout et d'effacement rapides, on vous demande d'implémenter et d'utiliser la structure de données décrite ci-dessous<sup>2</sup>. Vous en aurez besoin pour choisir les IDs des films de manière aléatoire en évitant de choisir des IDs précédemment tirés aléatoirement. En effet, si on vous demande de choisir  $N$  films aléatoirement avec  $N$

---

<sup>2</sup>Cette manière de procéder a été suggérée comme réponse sur StackOverflow à l'URL <http://stackoverflow.com/q/5682218/452614>.

qui est le nombre de films présents dans votre base de données, alors votre programme va prendre beaucoup de temps pour terminer sa sélection aléatoire.

Soit une structure de données comprenant une table hash  $H$  et un tableau  $A$ . Les clés de  $H$  sont les éléments de la structure de donnée et les valeurs sont leurs positions dans le tableau. Les éléments sont également présents dans le tableau. L'implémentation de quatre opérations de base est donnée ci-dessous :

- **insert( $v$ )**. On ajoute la valeur  $v$  au tableau  $A$  ce qui lui donne l'index  $i$ . On ajoute l'index dans  $H$  :  $H[v] := i$ .
- **contains( $v$ )**. On renvoie  $H.contains(v)$ .
- **getRandomElement()**. Si  $r$  est un nombre aléatoire compris entre 0 et la taille de  $A - 1$ , il suffit de renvoyer  $A[r]$ .
- **remove( $v$ )**. On remplace l'élément qui contient  $v$  dans  $A$  par le dernier élément de  $A$ . Soit  $d$  le dernier élément de  $A$  à l'index  $m$ . Soit  $i = H[v]$ , l'index dans  $A$  de la valeur à effacer. On fait  $A[i] := d$  et  $H[d] := i$ , on décroît la taille de  $A$  de 1 et on efface la valeur de  $H$ .

Toutes les opérations sont  $O(1)$  sauf l'ajout qui est de type  $O(1)$  amorti (amortized)<sup>3</sup>.

Ensuite, vous utiliserez DOM level 3 avec XDK (Oracle XML Developer Kit) pour construire le même arbre en RAM. Veillez bien entendu à réutiliser certaines parties de votre code. La différence se situe dans l'instanciation de l'implémentation DOM, la sérialisation (étape 5) et la validation (étape 6).

Vous ajouterez un argument `--dom` pour choisir le niveau du DOM, c'est-à-dire 2 ou 3.

En ce qui concerne les propriétés définies dans les documents MongoDB, on vous demande de respecter les points suivants :

- Ne pas considérer les propriétés `adult`, `backdrop_path`, `belongs_to_collection`, `imdb_id`, `popularity` ainsi que les sous-propriétés<sup>4</sup> `directors.credit_id`, `directors.department`, `directors.job`, `actors.credit_id`, `actors.order`. Même si le document XML de configuration en demande l'inclusion.
- On vous demande de "protéger" le contenu des éléments qui contiendront les propriétés `tagline` et `overview` à l'aide de sections CDATA.
- ATTENTION, veillez à ce que ce qui n'existe pas (`null` dans MongoDB) ou qui contient une valeur indiquant en réalité l'absence de valeur (`runtime` de 0) ne soit pas présent dans le document XML.
- Essayez d'ordonner les éléments dans votre document de manière à ce que les informations les plus pertinentes soient présentes au début du document comme par exemple le titre avant le budget.

## 5 - Sérialisation de l'arbre DOM - DOMSer

Sérialisez l'arbre DOM obtenu à l'étape précédente dans un fichier sur disque. L'argument `--output` permet de choisir le chemin qui va être utilisé pour écrire le document XML sur disque. Si on n'indique pas cet argument, la valeur par défaut est `movies.xml` dans le répertoire courant.

Veillez à ce que le titre, le nom des acteurs, ... en bref toutes les données dans lesquelles peuvent se trouver des caractères quelconques soient correctement sérialisés pour éviter une violation de la contrainte de forme (par exemple l'introduction brute du caractère `&` dans le texte).

Testez manuellement la contrainte de forme avec par exemple Netbeans ou Oxygen.

Effectuez la sérialisation pour DOM 2 et DOM 3 (Load and Save feature) selon la valeur de l'argument `--dom`.

<sup>3</sup>Cfr. [http://en.wikipedia.org/wiki/Amortized\\_analysis](http://en.wikipedia.org/wiki/Amortized_analysis)

<sup>4</sup>Les propriétés présentes dans des objets anonymes qui sont dans des tableaux nommés.

## 6 - Validation par DTD et XSD - DOMVal

---

On vous demande d'effectuer une validation par DTD avec SaX si l'argument `--validation` vaut `dtd` en utilisant JAXP. Notez qu'un round-trip (sérialisation et parsing), c'est-à-dire une relecture de l'arbre est nécessaire.

Ensuite construisez un schéma W3C XML Schema (XSD) que vous utiliserez avec DOM 2 ou 3 pour valider le document si l'argument `--validation` vaut `xsd` et que l'argument `--dom` vaut 2 ou 3 respectivement. Pour la validation DOM 2, utilisez JAXP et son package `javax.xml.validation` tandis que pour la validation DOM 3 utilisez un `XSDValidator` du XDK.

La validation par DTD nécessite de sérialiser le document XML. La sérialisation étant déjà réalisée sur le disque dur grâce à l'étape 5, nous allons l'utiliser pour faire la validation par DTD. Par contre la validation par XSD doit avoir lieu en mémoire directement sur l'arbre DOM sans sérialisation.

ATTENTION, veillez à ce que ce qui n'existe pas (`null` dans MongoDB) ou qui contient une valeur indiquant en réalité l'absence de valeur (`runtime` de 0) puisse ne pas être présent dans le document XML. Vous pouvez vous aider du fichier `stats.txt`<sup>5</sup> de SGBD pour écrire le schéma.

En cas de problème lors de la vérification de la contrainte de forme ou de la validité du document, vous devez arrêter le traitement et afficher un message clair.

Assez logiquement, cette étape devrait se trouver avant l'étape 5 mais pour des raisons pédagogiques elle a été déplacée après.

## 7 - Validation en extra avec XPath - DOMVal2

---

Deux contraintes additionnelles ont été définies. Vous allez développer le code nécessaire pour les tester à l'aide de XPath 1.0. Notez que dans la recommandation W3C XML Schema 1.1 il est possible d'utiliser des assertions avec XPath 2.0 pour réaliser des vérifications équivalentes.

- Lorsque la propriété `vote_average` est présente, la propriété `vote_count` doit également l'être.
- Lorsque la propriété `vote_count` est présente, la propriété `vote_average` doit également l'être.

On doit pouvoir réaliser cela avec DOM 2.

## 8 - Transformation XSLT et sérialisation - XSLT

---

L'implémentation XSLT 1.0 utilisée est Saxon 6.5.5<sup>6</sup>.

On vous demande de générer un document XML par transformation XSLT 1.0 contenant la liste des films triés par ordre décroissant sur le `vote_average` puis sur le `vote_count`. Les informations retenues pour chaque film sont : le titre, la date de sortie, les noms des réalisateurs séparés par des virgules, les noms des acteurs séparés par des virgules (max. 3 triés par ordre de `cast_id` croissant), le `vote_average` et le `vote_count`.

L'argument `--xslt` permet de choisir le chemin qui va être utilisé pour écrire le document XML sur disque. Si on n'indique pas cet argument, la valeur par défaut est `movies_xslt.xml` dans le répertoire courant.

Les films sont écrits dans le document sous forme tabulaire, c'est-à-dire une séquence de lignes composées de colonnes. Le vocabulaire XML que vous utiliserez est le suivant :

- L'élément `table` est l'élément contenant les lignes.
- Chaque élément `row` correspond à un film.
- Chaque élément `c` correspond à des données d'un film.

---

<sup>5</sup>[https://cours.khi.be/sghd\\_m1/labo/stats.txt](https://cours.khi.be/sghd_m1/labo/stats.txt)

<sup>6</sup><http://saxon.sourceforge.net/saxon6.5.5/>

- L'élément **caption** correspond au titre du tableau dans lequel on indiquera le nombre de films. Il s'agit du  $x$  dans l'exemple ci-dessous.
- L'élément **head** représente l'en-tête du tableau et contient un élément **c** par colonne.
- La colonne  $N$  représente le numéro d'ordre du film dans le tableau et prend consécutivement les valeurs de 1 à  $x$ .

```
<table>
  <caption>x movies</caption>
  <head>
    <c>N</c>
    <c>Title</c>
    <c>Release date</c>
    <c>Directors</c>
    <c>Actors</c>
    <c>Vote average</c>
    <c>Vote count</c>
  </head>
  ...
  <row>
    <c>42</c>
    <c>Inception</c>
    <c>2010-07-16</c>
    <c>Christopher Nolan</c>
    <c>Leonardo DiCaprio, Ken Watanabe, Joseph Gordon-Levitt</c>
    <c>7.5</c>
    <c>5578</c>
  </row>
  ...
</table>
```

## 9 - Mise en forme par CSS - CSS

---

En partant du document XML résultant de la transformation XSLT, on vous demande d'écrire une feuille de style CSS 2.1 (Cascading Style Sheets) pour réaliser un affichage tabulaire esthétique en utilisant le "CSS table model" ainsi que d'autres propriétés de CSS 2.1 pour contrôler les polices utilisées, les couleurs, les cadres, .... Cfr. <http://www.w3.org/TR/CSS2/tables.html> et plus généralement <http://www.w3.org/TR/CSS2/>.

Contrairement aux tableaux de XHTML, il est nécessaire ici d'indiquer quels éléments de notre document XML jouent quels rôles dans le tableau.

## 10 - Transformation vers XHTML 1.0 - XHTML

---

Adaptez la feuille de style écrite à l'étape 8 pour générer un document XHTML 1.0 permettant d'afficher les films sous forme tabulaire de manière semblable à ce qui a été réalisé aux deux points précédents. Le programme Java doit être capable d'invoquer les deux transformations en séquence en utilisant la même feuille de style.

Validez en Java le document XHTML résultant avec SaX et indiquez si c'est OK ou pas et pourquoi.

L'argument `-xhtml` permet de choisir le chemin qui va être utilisé pour écrire le document XML sur disque. Si on n'indique pas cet argument, la valeur par défaut est `movies.html` dans le répertoire courant.

Vos documents XHTML auront toujours la structure indiquée ci-dessous. L'encodage choisi est UTF-8.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
...
</html>
```

FIG. 1: Système informatique de la partie XML

