SER – Laboratoire 2

HEIG-VD

Adriano Ruberto & Matthieu Villard

2016

**Table des matières**

[1 Introduction 2](#_Toc449685537)

[2 Modification de la Structure du document XML 2](#_Toc449685538)

[3 Grammaire DTD 3](#_Toc449685539)

[4 Génération du document XML 3](#_Toc449685540)

[4.1 ControleurXMLCreation 3](#_Toc449685541)

[4.2 XMLSerializer 4](#_Toc449685542)

[5 Modification de la structure du document JSON 9](#_Toc449685543)

[6 Génération du document JSON 9](#_Toc449685544)

[6.1 ControleurMedia 9](#_Toc449685545)

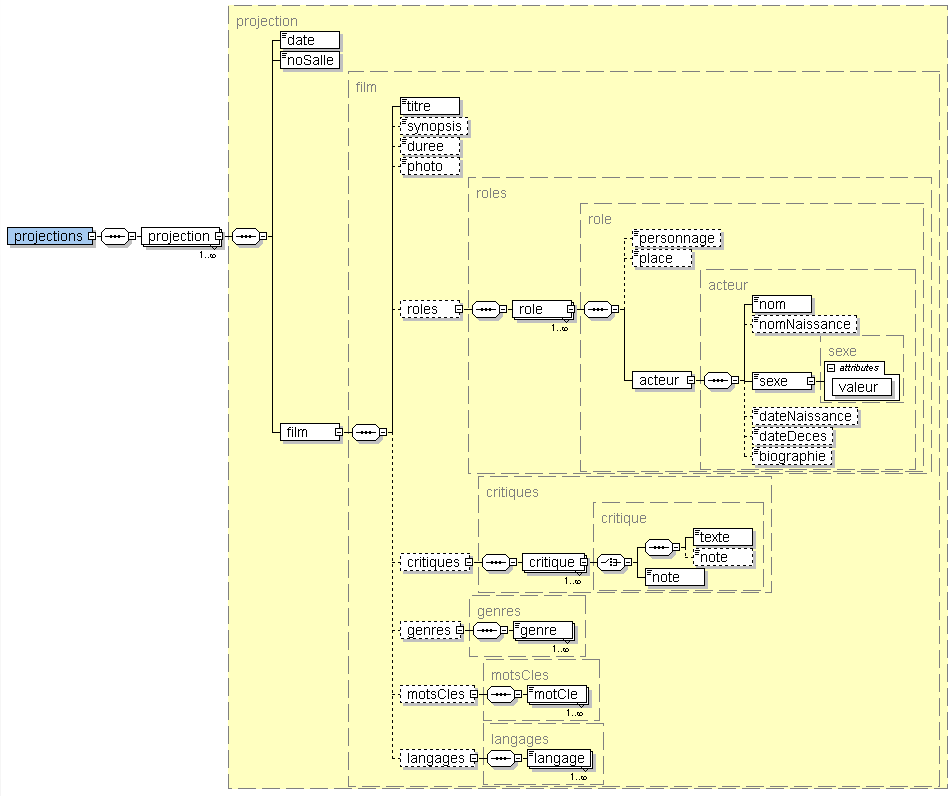
[7 Conclusion 11](#_Toc449685546)

# ****Introduction****

Ce laboratoire a pour but d’implémenter la sérialisation des données en documents XML et JSON. Nous avons donc repris l’application *SER\_PlexAdmin* et l’avons modifiée pour qu’elle permette de sérialiser les données en respectant le format que nous avions déjà défini durant le premier laboratoire. Nous verrons que le format a quelque peu été modifié, notamment pour ajouter certaines contraintes.

Durant ce laboratoire, il nous était demandé de compléter les contrôleurs *ControleurXMLCreation* et *ControleurMedia* et d’en modifier les méthodes *createXML et sendJSONToMedia.* Nous verrons que*,* au lieu de simplement implémenter ces deux méthodes et de décrire tout le processus dans celles-ci, nous avons opté pour l’utilisation de *Serializers*. Ainsi, nous avons créé deux nouvelles classes, à savoir la classe *XMLSerializer* pour générer le document XML et *JSONSerializer* pour générer le document JSON.

# ****Modification de la Structure du document XML****



Nous avons modifié la structure du document XML afin de vérifier les valeurs du sexe de l’acteur. Ainsi le sexe de l’acteur a un attribut valeur qui est un *enum* et ne peut donc contenir que des valeurs valables.

# Grammaire DTD

La figure ci-dessous illustre la nouvelle grammaire DTD contenue dans le fichier *validator.dtd*, située à la racine du projet *SER\_PlexAdmin*.

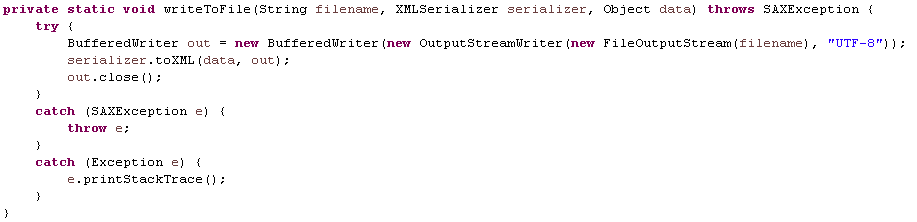


# Génération du document XML

## ControleurXMLCreation



La méthode *createXml* ressemble beaucoup à la méthode *createXStreamXml* déjà existante. Elle récupère les données de la base de données et crée un document XML à partir de celles-ci. La spécificité de cette méthode est d’utiliser un objet XMLSerializer au lien d’un XStream. Cette classe que nous avons implémentée nous-mêmes peut prendre en paramètre le chemin d’un fichier DTD permettant de valider le document XML généré. À noter que les exceptions levées, comme l’illustre la figure ci-dessus permettent de déterminer si le document générer est valide et le cas échéant d’afficher les messages d’erreur dans la fenêtre principale.



La méthode *writeToFile* déjà existante prenait en paramètre un *XStream* et nous avons donc dû la redéfinir. Nous noterons la nécessité de propager les exceptions *SAXException* pour indiquer que le document généré est invalide. Nous pouvons voir que c’est le *XMLSerializer* qui va réellement se charger de générer le document XML.

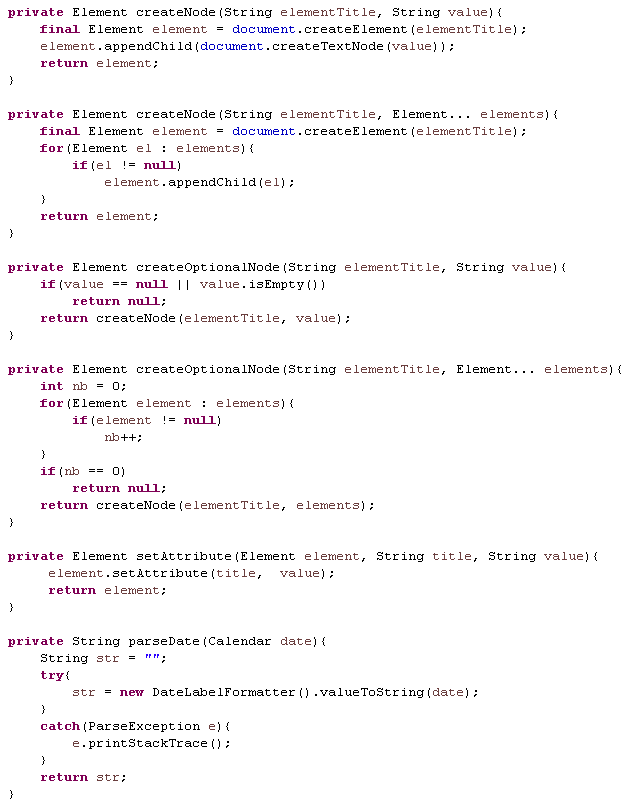
## XMLSerializer



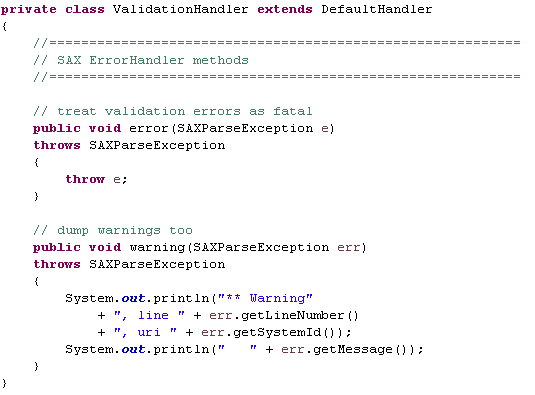
La méthode appelée pour sérialiser les données et les transformer en document XML se nomme *toXML*. Celle-ci utilise la méthode *generateDocument* pour obtenir un objet *Document* qui représente le document XML. Comme le montre la figure ci-dessus, le document peut inclure un schéma DTD. Le document est transformé et écrit puis est validé. À noter que la validation requière une transformation supplémentaire. Il sera intéressant dans les laboratoires à venir d’observer la pertinence de la validation du document XML lors de sa génération.



La méthode la plus importante est sans aucun doute celle chargée de transformer les données en un objet *Document*, à savoir *generateDocument.* Celle-ci, grâce à d’autres méthodes internes, crée les différents nœuds du document XML. Nous avons décidé de faire une différence entre les nœuds que nous jugions optionnels et ceux que nous jugions obligatoires. Ainsi, si un nœud obligatoire n’a pas de données, il sera tout de même ajouté, mais cette balise sera vide. L’appel aux méthodes interne permet d’adopter une approche fonctionnelle qui nous paraissait plus propre, notamment en utilisant des *Stream* pour le parcours des tableaux.



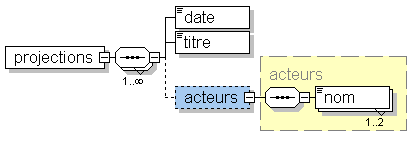
Ci-dessus, vous trouverez un aperçu des différentes méthodes internes qui permettent de générer un document. Vous noterez qu’elles retournent chacune un objet *Element*, ce qui permet de les combiner et ainsi d’avoir une approche très fonctionnelle.



Ceci est une classe interne et contribue à la validation du document XML. Ainsi, les avertissements ne causent pas d’arrêt du programme contrairement aux erreurs. Les erreurs ont pour effet de lever une exception qui pourra être détectée par le programme.

Vous trouverez à la racine du projet *SER\_PlexAdmin*, un document nommé data.xml illustrant le résultat d’une génération de document XML.

# Modification de la structure du document JSON



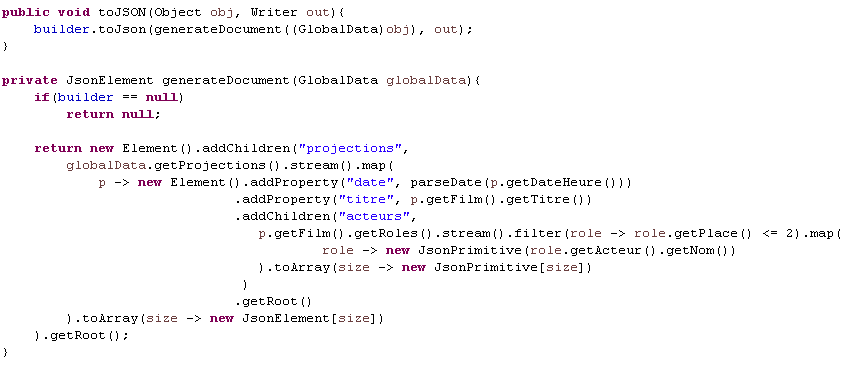
Nous avons modifié la structure du document JSON afin de supprimer certains éléments de la hiérarchie qui nous semblaient superflus. Ainsi, nous n’avons plus qu’une liste de projections et l’élément *projection* a disparu. De plus, les noms des acteurs ayant joué le premier et le deuxième rôle sont enregistrés dans un tableau, ce qui permet de garantir l’évolutivité de notre document.

# Génération du document JSON

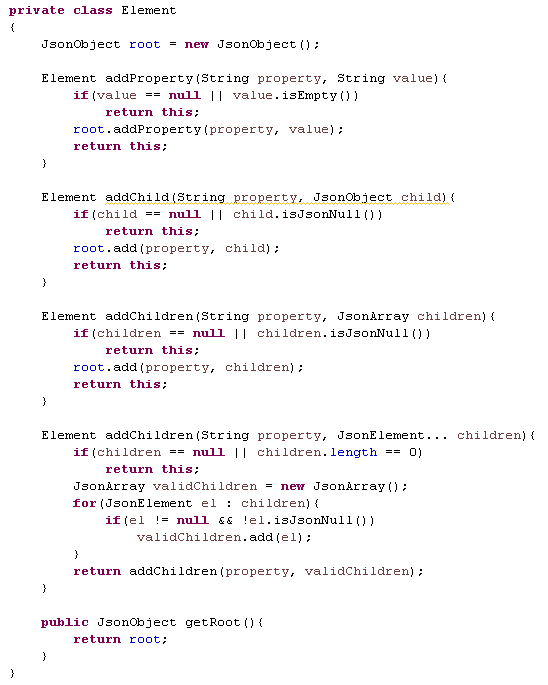
## ControleurMedia



Comme pour la génération du document XML, le contrôleur responsable de la génération du document JSON possède deux méthodes principales. La méthode *sendJSONToMedia,* charge les données et crée un *JSONSerializer*, puis utilise la méthode *writeToFile* pour transformer les données et les écrire sur le flux.



À nouveau, la génération du document est effectuée de manière plutôt fonctionnelle, en utilisant cette fois une classe interne *Element*. L’utilisation de *Stream* nous facilite beaucoup les choses pour le parcours et la transformation de tableaux.



Voici donc la classe interne *Element* utilisée pour générer les objets JSON et y ajouter des propriétés et des enfants. À noter que chaque méthode teste si les données sont non nulles et retournent un objet *Element* afin de pouvoir combiner plusieurs opérations.

Vous trouverez à la racine du projet *SER\_PlexAdmin*, un document nommé data.json illustrant le résultat d’une génération de document JSON.

# Conclusion

Il a été très intéressant de pouvoir transformer des données en documents JSON et XML. Nous avons constaté qu’il n’est pas aisé de transformer des données en document, notamment à cause de données pouvant être absentes. Ainsi, selon notre schéma, certaines données sont obligatoires, alors que selon la base de données, très peu le sont. Ainsi, pour le futur, il faudra veiller à surveiller quelles données sont effectivement présentes et donc valider les documents XML. Il a été très enrichissant de pouvoir utiliser les outils de transformation sans avoir recours à une écriture en dure des documents.