SER - Rapport Labo 2

James Smith et Jérémie Chatillon

19.04.18

Introduction

Dans ce laboratoire, nous avons du implémenter des fonctions java pour transformer des données d'une base de données contenant des projections en *XML* et *JSON*.

Changement XML - DTD

```
<!ELEMENT projections (projection)*>
<!ELEMENT projection (id, date, salle, film)>
<!ELEMENT film (id, titre, synopsis, durée, (critiques*), (genres*), (mots-cles*), (langues*), photo?,
<!ELEMENT acteur (nom, nomNaissance?, biographie, dateNaissance?, dateDeces?)>
<!ELEMENT photo EMPTY>
<!ELEMENT date (jour,mois,annee)>
<!ELEMENT salle (id, noSalle, taille)>

<!ELEMENT critiques (critique*)>
<!ELEMENT genres (genre*)>
<!ELEMENT mots-cles (mot-cle*)>
<!ELEMENT langues (langue*)>
<!ELEMENT roles (role*)>
<!ELEMENT role (id,personnage, place, acteur)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT synopsis (#PCDATA)>
<!ELEMENT durée (#PCDATA)>
<!ELEMENT critique (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT mot-cle (#PCDATA)>
<!ELEMENT langue (#PCDATA)>
<!ELEMENT personnage (#PCDATA)>
<!ELEMENT place (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT nomNaissance (#PCDATA)>
<!ELEMENT jour (#PCDATA)>
<!ELEMENT mois (#PCDATA)>
<!ELEMENT annee (#PCDATA)>
<!ELEMENT noSalle (#PCDATA)>
<!ELEMENT taille (#PCDATA)>
<!ELEMENT biographie (#PCDATA)>
<!ELEMENT dateNaissance (date)>
<!ELEMENT dateDeces (date)>

<!ATTLIST photo url CDATA #REQUIRED>
<!ATTLIST acteur sexe (Masculin|Feminin) #REQUIRED>
```

Nous nous sommes rendu compte que certaines données n'était pas obligatoire et que certain nom que nous avions mis ne correspondait pas avec la BD.

Changement dans la structure JSON

```
{
  "projections": [
    {
```

```
      "date": "17-04-2018",
      "titre": "Casino Royale",
      "premierRole": "Dench, Judi",
      "deuxiemeRole": "Jankovsky, Jaroslav"
    }
  ]
}
```


Nous avons simplifié la structure du JSON car l'ancien structure était trop complexe à réalisé avec *GSON*.

Implémentation de ControleurXMLCreation

```
package controllers;

import models.*;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;
import views.*;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.io.BufferedWriter;
import java.io.StringWriter;
import java.text.DecimalFormat;
import java.util.Calendar;
import java.util.EventListener;
import java.util.List;
import java.util.Set;

import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.jdom2.*;

/*import org.dom4j.*;
import org.dom4j.io.OutputFormat;
import org.dom4j.io.XMLWriter;
*/

import com.thoughtworks.xstream.XStream;

public class ControleurXMLCreation {

        //private ControleurGeneral ctrGeneral;
        private static MainGUI mainGUI;
        private ORMAccess ormAccess;

        private GlobalData globalData;

        public ControleurXMLCreation
          (ControleurGeneral ctrGeneral, MainGUI mainGUI, ORMAccess ormAccess){
              //this.ctrGeneral=ctrGeneral;
              ControleurXMLCreation.mainGUI=mainGUI;
              this.ormAccess=ormAccess;
        }

        public void createXML(){
          new Thread(){
                  public void run(){
                        mainGUI.setAcknoledgeMessage("Creation XML... WAIT");
                        long currentTime = System.currentTimeMillis();
                        try {
```

```
                globalData = ormAccess.GET_GLOBAL_DATA();

                        // Doctype
                DocType doctype = new DocType("cinema","cinema.dtd");

                Element root = new Element("projections");

                // Parcours des projections
                for(Projection projection: globalData.getProjections()){
                      root.addContent(createProjection(projection,root));
                }

                Document document = new Document(root, doctype);

                // Sauvegarde dans le fichier XML
                XMLOutputter outp = new XMLOutputter(Format.getPrettyFormat());
                outp.output(document, new FileOutputStream("cinema.xml"));
                long endTime = System.currentTimeMillis();
                mainGUI.setAcknoledgeMessage("Le fichier XML a été créé en "
                  +timeToSeconds(currentTime,endTime));
              }
              catch (Exception e){
                mainGUI.setErrorMessage("Construction XML impossible"
                  , e.toString());
              }
          }
        }.start();
}

private Element createProjection(Projection projection, Element root){
        Element project = new Element("projection");

        /** ID Projection**/
        Element id = new Element("id");
        id.setText(Long.toString(projection.getId()));
        project.addContent(id);

        /** DATE **/
        Element date = createDate(projection.getDateHeure());
        project.addContent(date);

        /** SALLE **/
        Element salle = createSalle(projection.getSalle());
        project.addContent(salle);

        /** FILM **/
        Element film = createFilm(projection.getFilm());
        project.addContent(film);

        return project;

}

public Element createDate(Calendar dateValue){
        Element date = new Element("date");
```

```
        /** JOUR **/
        Element jour = new Element("jour");

        /** MOIS **/
        Element mois = new Element("mois");
        /** ANNEE **/
        Element annee = new Element("annee");

        int intJour = dateValue.get(Calendar.DAY_OF_MONTH);
        int intMois = dateValue.get(Calendar.MONTH);
        int intAnnee = dateValue.get(Calendar.YEAR);

        jour.setText(Integer.toString(intJour));
        mois.setText(Integer.toString(intMois));
        annee.setText(Integer.toString(intAnnee));

        date.addContent(jour);
        date.addContent(mois);
        date.addContent(annee);
        return date;
}

private Element createSalle(Salle salleValue){
        Element salle = new Element("salle");

        /** ID **/
        Element id = new Element("id");

        /** NO SALLE **/
        Element noSalle = new Element("noSalle");
        /** TAILLE DE LA SALLE **/
        Element taille = new Element("taille");

        id.setText(Long.toString(salleValue.getId()));
        noSalle.setText(salleValue.getNo());
        taille.setText(Integer.toString(salleValue.getTaille()));

        salle.addContent(id);
        salle.addContent(noSalle);
        salle.addContent(taille);

        return salle;
}

private Element createFilm(Film filmValue){
        Element film = new Element("film");

        /** ID **/
        Element id = new Element("id");
        id.setText(Long.toString(filmValue.getId()));
        film.addContent(id);

        /** TITRE **/
        Element titre = new Element("titre");
        titre.setText(filmValue.getTitre());
        film.addContent(titre);
```

```
        /** SYNOPSIS **/
        Element synopsis = new Element("synopsis");
        synopsis.setText(filmValue.getSynopsis());
        film.addContent(synopsis);

        /** DUREE **/
        Element duree = new Element("durée");
        duree.setText(Integer.toString(filmValue.getDuree()));
        film.addContent(duree);

        /** CRITIQUES **/
        if(filmValue.getCritiques() != null) {
                Element critiques = createCritiques(filmValue);
                film.addContent(critiques);
        }

        /** MOTS CLES **/
        if(filmValue.getMotcles() != null) {
                Element motsCles = createMotsCles(filmValue);
                film.addContent(motsCles);
        }

        /** LANGUES **/
        if(filmValue.getLangages() != null) {
                Element langues = createLangues(filmValue);
                film.addContent(langues);
        }

        /** PHOTO **/
        if(filmValue.getPhoto() != null) {
                Element photo = new Element("photo");
                photo.setAttribute("url", filmValue.getPhoto());
                film.addContent(photo);
        }

        /** ROLES **/
        if(filmValue.getRoles() != null) {
                Element roles = createRoles(filmValue);
                film.addContent(roles);
        }

        return film;
}

private Element createCritiques(Film filmValue){
        Element critiques = new Element("critiques");

        for(Critique critiqueValue: filmValue.getCritiques()){
                Element critique = new Element("critique");
                critique.setText(critiqueValue.getTexte());
                critiques.addContent(critique);
        }

        return critiques;
}
```

```
private Element createMotsCles(Film filmValue){
        Element motsCles = new Element("mots-cles");

        for(Motcle motcleValue: filmValue.getMotcles()){
                Element motcle = new Element("mot-cle");
                motcle.setText(motcleValue.getLabel());
                motsCles.addContent(motcle);
        }

        return motsCles;
}

private Element createLangues(Film filmValue){
        Element langues = new Element("langues");

        for(Langage langueValue : filmValue.getLangages()){
                Element langue = new Element("langue");
                langue.setText(langueValue.getLabel());
                langues.addContent(langue);
        }

        return langues;
}

private Element createRoles(Film filmValue){
        Element roles = new Element("roles");

        for(RoleActeur roleValue: filmValue.getRoles()){
          Element role = new Element("role");

          /** ID **/
          Element id = new Element("id");
          /** PERSONNAGE **/
          Element personnage = new Element("personnage");
          /** PLACE DANS LE FILM **/
          Element place = new Element("place");
          /** ACTEUR **/
          Element acteur = createActeur(roleValue);

          id.setText(Long.toString(roleValue.getId()));
          personnage.setText(roleValue.getPersonnage());
          place.setText(Long.toString(roleValue.getPlace()));

          role.addContent(id);
          role.addContent(personnage);
          role.addContent(place);
          role.addContent(acteur);
          role.setAttribute("sexe",roleValue.getActeur().getSexe().toString());

          roles.addContent(role);

        }

        return roles;
}
```

```
private Element createActeur(RoleActeur roleValue){
        Acteur acteurValue = roleValue.getActeur();
        Element acteur = new Element("acteur");

        /** NOM **/
        Element nom = new Element("nom");
        nom.setText(acteurValue.getNom());
        acteur.addContent(nom);

        /** NOM NAISSANCE **/
        if(acteurValue.getNomNaissance() != null) {
                Element nomNaissance = new Element("nomNaissance");
                nomNaissance.setText(acteurValue.getNomNaissance());
                acteur.addContent(nomNaissance);
        }

        /** BIOGRAPHIE **/
        Element biographie = new Element("biographie");
        biographie.setText(acteurValue.getBiographie());
        acteur.addContent(biographie);

        /** DATE NAISSANCE **/
        if(acteurValue.getDateNaissance() != null) {
                Element dateNaissance = new Element("dateNaissance");
                dateNaissance.addContent(createDate(acteurValue.getDateNaissance()));
                acteur.addContent(dateNaissance);
        }

        /** DATE DECES **/
        if(acteurValue.getDateDeces() != null) {
                Element dateDeces = new Element("dateDeces");
                dateDeces.addContent(createDate(acteurValue.getDateDeces()));
                acteur.addContent(dateDeces);
        }

        return acteur;
}

public void createXStreamXML(){
  new Thread(){
        public void run(){
          mainGUI.setAcknoledgeMessage("Creation XML... WAIT");
          long currentTime = System.currentTimeMillis();
          try {
                globalData = ormAccess.GET_GLOBAL_DATA();
                globalDataControle();
          }
          catch (Exception e){
                mainGUI.setErrorMessage("Construction XML impossible", e.toString());
          }

          XStream xstream = new XStream();
          writeToFile("global_data.xml", xstream, globalData);
          System.out.println("Done [" +
            displaySeconds(currentTime, System.currentTimeMillis()) + "]");
```

```
            mainGUI.setAcknoledgeMessage("XML cree en "+
              displaySeconds(currentTime, System.currentTimeMillis()) );
        }
        }.start();
}

private static void writeToFile(String filename, XStream serializer, Object data) {
        try {
                BufferedWriter out = new BufferedWriter(
                  new OutputStreamWriter(new FileOutputStream(filename), "UTF-8"));
                serializer.toXML(data, out);
                out.close();
        } catch (Exception e) {
                e.printStackTrace();
        }
}

private static final DecimalFormat doubleFormat = new DecimalFormat("#.#");
private static final String displaySeconds(long start, long end) {
        long diff = Math.abs(end - start);
        double seconds = ((double) diff) / 1000.0;
        return doubleFormat.format(seconds) + " s";
}

private void globalDataControle(){
        for (Projection p:globalData.getProjections()){
                System.out.println("****************************************");
                System.out.println(p.getFilm().getTitre());
                System.out.println(p.getSalle().getNo());
                System.out.println("Acteurs *********");
                for(RoleActeur role : p.getFilm().getRoles()) {
                        System.out.println(role.getActeur().getNom());
                }
                System.out.println("Genres *********");
                for(Genre genre : p.getFilm().getGenres()) {
                        System.out.println(genre.getLabel());
                }
                System.out.println("Mot-cles *********");
                for(Motcle motcle : p.getFilm().getMotcles()) {
                        System.out.println(motcle.getLabel());
                }
                System.out.println("Langages *********");
                for(Langage langage : p.getFilm().getLangages()) {
                        System.out.println(langage.getLabel());
                }
                System.out.println("Critiques *********");
                for(Critique critique : p.getFilm().getCritiques()) {
                        System.out.println(critique.getNote());
                        System.out.println(critique.getTexte());
                }
        }
}

private String timeToSeconds(long start, long finish){
        long time = Math.abs(finish - start);
        double timeSeconds = (double)time / 1000;
```

```
                return String.valueOf(timeSeconds) + " s";

        }
}
```

On a décidé d'implémenter ca en séparant les différents bloque dans des fonctions pour facilité la lecture.

Nous avons utilisé *JDOM* pour réaliser l'XML.

L'utilisation de la fonction avec JDOM prend: 0.245 s. L'utilisation de la fonction avec XStream prend: 0.3 s.

On peut donc voir que de le faire avec *JDOM* est plus rapide qu'avec *XStream* mais que ca demande beaucoup plus de travail de la part du developpeur et à chaque changement de la structure nécessite des modifications qui peuvent être conséquentes et longue.

Implémentation de ControleurMedia

```
package controllers;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import models.*;
import views.*;

import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.*;

public class ControleurMedia {

        private ControleurGeneral ctrGeneral;
        private static MainGUI mainGUI;
        private ORMAccess ormAccess;

        private GlobalData globalData;

        public ControleurMedia(ControleurGeneral ctrGeneral, MainGUI mainGUI, ORMAccess ormAccess){
                this.ctrGeneral=ctrGeneral;
                ControleurMedia.mainGUI=mainGUI;
                this.ormAccess=ormAccess;
        }

        protected class ProjectionsJSON{
                private List<ProjectionJSON> projections;

                protected class ProjectionJSON{
                        private String date;
                        private String titre;
                        private String premierRole;
                        private String deuxiemeRole;

                        protected ProjectionJSON(Projection projection){
                                // DATE
```

```
                        SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
                        this.date =dateFormat.format(projection.getDateHeure().getTime());
                        // TITRE
                        this.titre = projection.getFilm().getTitre();

                        Set<RoleActeur> roles = projection.getFilm().getRoles();
                        Iterator it = roles.iterator();

                        // PREMIER ROLE
                        premierRole = ((RoleActeur) it.next()).getActeur().getNom();

                        // DEUXIEME ROLE
                        deuxiemeRole = ((RoleActeur) it.next()).getActeur().getNom();
                        System.out.println(date + " " + titre + " " + premierRole +
                          " " + deuxiemeRole);
                }

        }

        protected ProjectionsJSON(){}

        protected ProjectionsJSON(List<ProjectionJSON> projections){
                this.projections = projections;
        }

        protected void setProjections(List<ProjectionJSON> projections){
                this.projections = projections;
        }
}

public void sendJSONToMedia(){
  new Thread(){
        public void run(){
                mainGUI.setAcknoledgeMessage("Envoi JSON ... WAIT");
                long currentTime = System.currentTimeMillis();
                try {
                  globalData = ormAccess.GET_GLOBAL_DATA();
                  //mainGUI.setWarningMessage("Envoi JSON: Fonction non encore implementee");

                  List<ProjectionsJSON.ProjectionJSON> projectionJSONList = new ArrayList<>();

                  for(Projection projection: globalData.getProjections()){
                        projectionJSONList.add(
                          new ProjectionsJSON().new ProjectionJSON(projection));
                  }
                  ProjectionsJSON projectionsJSON = new ProjectionsJSON(projectionJSONList);

                  try (PrintWriter writer = new PrintWriter(new FileWriter("cinema.json"))) {
                        Gson gson = new GsonBuilder().setPrettyPrinting().create();
                        gson.toJson(, writer);
                  }
                  long endTime = System.currentTimeMillis();
                  System.out.println("JSON done in " +
                    timeToSeconds(currentTime, endTime) + "");
                  mainGUI.setAcknoledgeMessage("Le fichier JSON a été créé en "
                    + timeToSeconds(currentTime, endTime));
```

```
                        }
                        catch (Exception e){
                                mainGUI.setErrorMessage("Construction JSON impossible", e.toString());
                        }
                }
          }.start();
        }

        private String timeToSeconds(long start, long finish){
                long time = Math.abs(finish - start);
                double timeSeconds = (double)time / 1000;
                return String.valueOf(timeSeconds) + " s";

        }

}
```

Pour implémenter cette classe, nous avons utilisé *GSON*. C'est très simple de prendre en main et nécessite pas énormement de temps à mettre en place. Mais si on veut faire des structures plus complexe devient plus compliquer à utiliser.

L'utlisation de sendJSONToMedia prends: 0.083 s.

Extrait de l'XML généré

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cinema SYSTEM "cinema.dtd">
<projections>
  <projection>
    <id>2597</id>
    <date>
      <jour>17</jour>
      <mois>3</mois>
      <annee>2018</annee>
    </date>
    <salle>
      <id>1</id>
      <noSalle>Flon 1</noSalle>
      <taille>200</taille>
    </salle>
    <film>
      <id>2053403</id>
      <titre>Casino Royale</titre>
      <synopsis>Casino Royale introduces James Bond before he holds his license to kill.
      But Bond is no less dangerous, and with two professional assassinations in quick succession,
      he is elevated to '00' status. Bond's first 007 mission takes him to Uganda where he is to
      spy on a terrorist, Mollaka. Not everything goes to plan and Bond decides to investigate,
      independently of MI6, in order to track down the rest of the terrorist cell. Following a
      lead to the Bahamas, he encounters Dimitrios and his girlfriend, Solange. He learns that
      Dimitrios is involved with Le Chiffre, banker to the world's terrorist organizations.
      Secret Service intelligence reveals that Le Chiffre is planning to raise money in a
      high-stakes poker game in Montenegro at Le Casino Royale. MI6 assigns 007 to play against
      him, knowing that if Le Chiffre loses, it will destroy his organization. 'M' places Bond
      under the watchful eye of the beguiling Vesper Lynd. At first skeptical of what value Vesper
      can provide, Bond's interest in her deepens as they brave danger together and even torture
```

```
at the hands of Le Chiffre. In Montenegro, Bond allies himself with Mathis MI6's local
field agent, and Felix Leiter who is representing the interests of the CIA. The marathon
game proceeds with dirty tricks and violence, raising the stakes beyond blood money and
reaching a terrifying climax.</synopsis>
<durée>144</durée>
<critiques />
<mots-cles>
  <mot-cle>assassin</mot-cle>
  <mot-cle>corpse</mot-cle>
  <mot-cle>2000s</mot-cle>
  <mot-cle>blood-spatter</mot-cle>
  <mot-cle>snake</mot-cle>
  <mot-cle>double-cross</mot-cle>
  <mot-cle>electrocution</mot-cle>
</mots-cles>
<langues>
  <langue>French</langue>
  <langue>English</langue>
</langues>
<roles>
  <role sexe="MASCULIN">
    <id>2218</id>
    <personnage>Card Players</personnage>
    <place>35</place>
    <acteur>
      <nom>Inzerillo, Jerry</nom>
      <biographie />
    </acteur>
  </role>
  <role sexe="MASCULIN">
    <id>2224</id>
    <personnage>Hot Room Technicians</personnage>
    <place>41</place>
    <acteur>
      <nom>Cox, Simon</nom>
      <biographie />
    </acteur>
  </role>
  <role sexe="MASCULIN">
    <id>2206</id>
    <personnage>Infante</personnage>
    <place>23</place>
    <acteur>
      <nom>Ade</nom>
      <biographie>Born in London, in the early seventies, Ade tried his hand at various trades;
      from selling shoes, tickets for concerts, managing bands, to publishing an
      international horse racing magazine. Ade was known by Guy Ritchie who cast him as
      Tyrone the getaway driver in _Snatch. (2000)_ (qv). He then set off on the European
      press jaunt &amp;amp; traveled to the United States to preview
      the movie, addressing the audience at Harry Knowles' 24hr film festival at The Alamo,
      Texas.  Ade has since been involved with several other film and TV projects; notably
      as Infante in the Bond movie _Casino Royale (2006)_ (qv) and Madonna's directorial
      debut _Filth and Wisdom (2008)_ (qv).</biographie>
    </acteur>
  </role>
</roles>
```

```
    </film>
  </projection>
</projections>
```

Extrait du fichier JSON généré

```
{
  "projections": [
    {
      "date": "17-04-2018",
      "titre": "Casino Royale",
      "premierRole": "Nonyela, Valentine",
      "deuxiemeRole": "Cole, Christina"
    }
  ]
}
```

Conclusion

La sérialisation de donné à beaucoup de méthode possible. Chaqu'une d'elles à ses avantages et inconveniant. Il faut donc à chaque fois ce poser la question sur quelle méthode est la plus approprié au problème.

Si le temps d'exécution n'est pas un problème, XStream sera la plus simple des solutions. Dans le cas d'un problème d'optimisation et que le temps de développement n'est pas restraint, JDOM sera une bonne solution. GSON ne donnant pas le meme type de fichier pourra être utilisé de manière différente. Beaucoup de framework JS utilise du JSON donc GSON sera fort probablement plus utiliser lors de la combinaison avec ce type de technologie.