1. Introduction

Lancez Eclipse. Créez un projet appelé `TP6` dans votre Workspace Java favori. Vérifiez qu'il sépare bien les sources `src` des binaires `bin`. Tous les sources seront mis ensemble, mais seulement l'une des classes (`Test1`, `Test2`...) sera lancée à la fois.

Téléchargez le fichier UtilXML.java. Il faut ajouter ce fichier à votre projet. Il contient des méthodes pour faciliter la lecture et l'écriture de fichiers XML.

2. Sérialisation

Le TP va vous faire programmer ce qu'il faut pour sérialiser et dé-sérialiser des données Java sous forme d'XML. La sérialisation XML d'un objet consiste à écrire ses variables membres sous forme de sous-éléments ou d'attributs, selon ce qui est possible. La dé-sérialisation consiste à instancier un objet à partir de ce qu'on lit dans le document : on reconstruit un objet à partir d'un document XML.

La même chose pourrait être faite avec du JSON ou des données CSV.

Par exemple, soit la classe suivante, avec son constructeur :

```
class Article
{
    private int id;
    private String nom;
    private float prix;

    @SuppressWarnings("unused")
    private Article() {}

    public Article(int id, String nom, float prix)
    {
        this.id = id;
        this.nom = nom;
        this.prix = prix;
    }

    public String toString()
    {
        StringBuilder sb = new StringBuilder();
        sb.append("Article[id="); sb.append(id);
        sb.append(", nom=\"");    sb.append(nom);
        sb.append("\", prix=");   sb.append(prix);
        sb.append("]");
        return sb.toString();
    }
}
```

Copiez-collez ce source dans votre projet Eclipse.

Soit une instance de cette classe créée par le programme suivant, on peut l'afficher avec un appel éventuellement implicite à `toString()` :

```
public class Test1
{
    public static void main(String[] args)
    {
        Article ballon = new Article(1, "ballon", 12.50f);
        System.out.println(ballon.toString());
    }
}
```

Copiez-collez dans Eclipse puis lancez-le. Ça affiche `Article[id=1, nom="ballon", prix=12.5]` dans la console.

Cette méthode `toString()` effectue une sérialisation en mode texte, puisqu'elle écrit toutes les informations nécessaires pour reconstruire l'instance. Il suffirait d'analyser cette chaîne avec une expression régulière. L'idée est de faire la même chose que `toString()`, mais en XML.

2.1. Sérialisation XML

Une sérialisation XML pourrait être la suivante, indentée ou pas :

```
<article id="1">
    <nom>ballon</nom>
    <prix>12.5</prix>
</article>
```

Comme toujours avec XML, il y a plusieurs choix : tout attribut, tout élément ou mixte.

Alors il n'est pas question de faire des affichages *ad-hoc* pour produire le XML. On va utiliser l'API DOM bien proprement. Voici ce que ça donne. Ajoutez cette méthode dans la classe `Article` et complétez-la :

```
public void toXML(Document document, Node parent)
{
    // élément <article>
    Element elArticle = document.createElement("article");
    parent.appendChild(elArticle);

    // attribut <article id="n°">
    elArticle.setAttribute("id", Integer.toString(id));

    // à vous de rajouter ce qu'il faut pour créer les sous-éléments <nom> et <prix>
}
```

Les variables locales de type `Element` sont nommées en `elChapo` pour les distinguer des variables membres du même nom.

Cette fonction est appelée, par exemple dans la fonction `main`, de cette manière :

```
public class Test2
```

```
{
    public static void main(String[] args) throws Exception
    {
        // instance à sérialiser
        Article ballon = new Article(1, "ballon", 12.50f);

        // créer le document XML et sa racine
        Document document = UtilXML.creerDocumentDOM();

        // sérialiser l'article directement au niveau du document
        ballon.toXML(document, document);

        // écrire le document dans un fichier
        UtilXML.ecrireDocumentDOM(document, "article.xml");
    }
}
```

Donc maintenant, la problématique, c'est faire l'inverse : initialiser une ou plusieurs instances à partir d'un document XML.

2.2. Dé-sérialisation XML

On part d'un document XML valide (conforme à la fonction de sérialisation) :

```
<?xml version="1.0" encoding="UTF8"?>
<article id="3"><nom>filet de dingue</nom><prix>29.95</prix></article>
```

Voici un extrait de la fonction principale : 

```
public class Test3
{
    public static void main(String[] args) throws Exception
    {
        // ouverture du fichier XML
        Document document = UtilXML.lireDocumentDOM("article.xml", false);

        // lire et afficher la racine
        Node racine = document.getDocumentElement();
        Article article = Article.fromXML(racine);
        if (article != null) {
            System.out.println(article);
        } else {
            System.err.println("article.xml n'a pas pu être lu");
        }
    }
}
```

La méthode `fromXML(Node node)` est statique dans la classe `Article`. C'est une *fabrique* qui retourne une instance d'`Article` construite à partir du document XML ou `null` s'il est incorrect.

Voici un extrait à placer dans la classe `Article` et à compléter : 

```
public static Article fromXML(Node node) throws Exception
{
    // TODO vérifier que node est un élément <article>, retourner null sinon

    // conversion en Element
    Element elArticle = (Element) node;

    // article à initialiser et retourner
    Article article = new Article();

    // lire l'attribut id
    article.id = Integer.parseInt(elArticle.getAttribute("id"));

    // lire le nom
    Node elNom = elArticle.getFirstChild();
    article.nom = elNom.getTextContent();

    // TODO lire le prix
    Node elPrix = elNom.getNextSibling();
    article.prix = 0.0f;

    // retourner l'instance
    return article;
}
```

Cette manière de programmer la fonction `main` et la méthode `toXML` sans faire aucun test et en attendant exactement les éléments prévus présente plusieurs problèmes potentiels :

- Aucun test n'est fait pour être sûr qu'on est bien sur un élément `<article>`. On pourrait être sur un texte, ça ferait tout planter : `ClassCastException`. La méthode `UtilXML.isElement(node, nom)` pourrait être utile pour cette vérification.
- Aucun test qu'il y a ou non l'attribut `id` ; c'est pas trop grave si on considère que le document est valide.
- Aucun test de l'élément `<nom>` : présent/absent, vide ?
- Aucun test de l'élément `<prix>` : présent/absent, contenant un nombre correct (`NumberFormatException`) ?
- Et si les sous-éléments n'étaient pas dans cet ordre ?

Également, c'est à vous de gérer les exceptions. Avec la manière montrée ci-dessus, c'est dans la méthode `main` que ça plantera s'il y a un problème avec le fichier XML. L'autre manière serait d'intercepter les exceptions dans `fromXML` et de retourner `null` le cas échéant.

Vérifier si ça plante si on indente le fichier xml. Rajoutez les tests et les boucles pour sauter les `Nodes` non attendus, afin de tolérer les variantes, ou même les documents invalides.

3. Sérialisation de plusieurs données liées

On continue avec une autre classe :

```
class Client
{
    private String nom;
    private ArrayList<Article> articles = new ArrayList<Article>();

    @SuppressWarnings("unused")
    public Client() {}

    public Client(String nom)
    {
        this.nom = nom;
    }

    public void add(Article article)
    {
        if (article != null) {
            articles.add(article);
        }
    }

    public String toString()
    {
        StringBuilder sb = new StringBuilder();
        sb.append("Client[nom=");
        sb.append(nom);
        for (Article article: articles) {
            sb.append(", ");
            sb.append(article.toString());
        }
        sb.append("]");
        return sb.toString();
    }
}
```

Voici le programme qui permet de la mettre en œuvre :

```
public class Test4
{
    public static void main(String[] args) throws Exception
    {
        // un client
        Client client = new Client("Euphraste");

        // quelques articles
```

```
        client.add(new Article(1, "ballon", 12.50f));
        client.add(new Article(2, "filet", 24.90f));
        client.add(new Article(3, "raquette", 56.25f));

        // créer le document XML et sa racine
        Document document = UtilXML.creerDocumentDOM();

        // sérialiser le client
        client.toXML(document, document);

        // écriture du document dans un fichier
        UtilXML.ecrireDocumentDOM(document, "client.xml");
    }
}
```

Pour relire les données, on pourrait faire ceci : 

```
public class Test5
{
    public static void main(String[] args) throws Exception
    {
        // ouverture du fichier XML
        Document document = UtilXML.lireDocumentDOM("client.xml", false);

        // lire et afficher la racine
        Node racine = document.getDocumentElement();
        Client client = Client.fromXML(racine);
        if (client != null) {
            System.out.println(client);
        } else {
            System.err.println("client.xml n'a pas pu être lu");
        }
    }
}
```

Écrire les deux méthodes de sérialisation et de dé-sérialisation XML des clients. Choisissez votre propre format d'écriture des données en XML.

4. Sérialisation avec l'API JAXB

Cette API, incluse en standard dans le SDK Java, rend tout le travail beaucoup plus facile. Elle demande seulement de coller des annotations sur les classes et variables membres. Relire le cours. Il y a également des informations sur JAXB sur GitHub et dans ce tutoriel.

4.1. Sérialisation d'une classe

Dupliquez la classe `Article` sous le nom `Livre` et supprimez les méthodes `toXML` et `fromXML`. Il ne reste que les variables membres, les deux constructeurs et la méthode `toString`.

Remplacez les import W3C par ceux de JAXB. Rajoutez les annotations JAXB pour la sérialisation. Elles sont dans le package `javax.xml.bind.annotation`. Choisissez si les variables doivent devenir des attributs ou des éléments.

Voici maintenant comment lancer la sérialisation :

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import java.io.File;

public class Test6
{
    public static void main(String[] args) throws Exception
    {
        Livre livre = new Livre(1, "Le Vagabond des étoiles", 7.50f);

        File file = new File("livre.xml");
        JAXBContext context = JAXBContext.newInstance(Livre.class);
        Marshaller marshaller = context.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_ENCODING,  "UTF-8") ;
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        marshaller.marshal(livre, file);
    }
}
```

La désérialisation est tout aussi facile :

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import java.io.File;

public class Test7
{
    public static void main(String[] args) throws Exception
    {
        File file = new File("livre.xml");
        JAXBContext context = JAXBContext.newInstance(Livre.class);
        Unmarshaller unmarshaller = context.createUnmarshaller();
        Livre livre = (Livre) unmarshaller.unmarshal(file);

        System.out.println(livre.toString());
    }
}
```

4.2. Sérialisation d'une collection

Copiez la classe `Client` et nommez-la `Librairie`. Renommez `articles` en `livres` et supprimez les méthodes `toXML` et `fromXML`.

Comme pour `Livre`, rajoutez les `import` et les annotations afin de pouvoir sérialiser avec JAXB.

Voici maintenant comment lancer la sérialisation : 

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import java.io.File;

public class Test8
{
    public static void main(String[] args) throws Exception
    {
        Librairie librairie = new Librairie("classiques américains");
        librairie.add(new Livre(1, "Le Vagabond des étoiles", 7.50f));
        librairie.add(new Livre(2, "Les Aventures de Tom Sawyer", 6.90f));
        librairie.add(new Livre(3, "Histoires extraordinaires", 7.20f));

        File file = new File("librairie.xml");
        JAXBContext context = JAXBContext.newInstance(Librairie.class);
        Marshaller marshaller = context.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_ENCODING,  "UTF-8") ;
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        marshaller.marshal(librairie, file);
    }
}
```

Si vous regardez le fichier `librairie.xml`, vous verrez que les livres sont sérialisés chacun dans un élément `<livres>`. Pour que ça soit `<livre>`, il faut mettre l'annotation `@XmlElement(name="livre")` devant la liste.

Écrivez la classe `Test9` qui permet de désérialiser `librairie.xml` et l'afficher sur l'écran.

5. Travail à rendre

Vous avez programmé dans un projet `TP6` dans le workspace Java. Il faut seulement rendre son dossier `src` (pas les fichiers XML ni les binaires).

Cliquez droit sur le dossier `src` du projet, choisissez `Compresser...`, cliquez sur `Créer`. Ça va créer une archive `src.tar.gz`. Déposez cette archive sur Moodle, dans la page de cours dédiée.