1. Introduction

Lancez Eclipse. Créez un projet appelé `TP7` dans votre Workspace Java favori. Vérifiez qu'il sépare bien les sources `src` des binaires `bin`.

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

2. Recherche d'information dans un document XML

L'API SAX permet de parcourir un document XML très rapidement et sans occuper beaucoup de mémoire. On va l'utiliser pour extraire des informations de `factbook.xml`. Il faut aussi sa DTD `factbook.dtd`.

Programmer avec l'API SAX demande d'écrire une classe héritant de `DefaultHandler`. C'est un écouteur pour des événements survenant pendant le parcours du document, du type « balise ouvrante rencontrée », « balise fermante rencontrée », « texte rencontré ». Cette classe surcharge généralement plusieurs méthodes, selon les besoins :

- `startDocument()` : appelée quand on commence à traiter le document,
- `endDocument()` : appelée quand le document a été entièrement traité,
- `startElement(..., String qName, Attributes attrs)` : appelée quand on rencontre la balise ouvrante `<qName>` ; on peut accéder à ses attributs,
- `endElement(String qName)` : appelée quand on rencontre la balise fermante `</qName>`,
- `characters(char[] text...)` : appelée quand on rencontre du texte entre deux balises.

On peut aussi surcharger le gestionnaire d'erreur.

2.1. Programme principal

Le squelette de programme est le suivant :

```
public class Main1
{
    public static void main(final String[] args)
    {
        try {

            // ouvrir et traiter le fichier
            UtilXML.lireDocumentSAX("factbook.xml", new FactbookContentHandler1());

        } catch (Exception e) {
            // afficher où ça s'est planté
            e.printStackTrace();
        }
    }
}
```

Tout se passe dans la classe `FactbookContentHandler1`.

2.2. Classe `FactbookContentHandler1`

Voici le squelette de la classe `FactbookContentHandler1` :

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;

public class FactbookContentHandler1 extends DefaultHandler
{
    /**
     * Actions à réaliser lors de la détection d'une balise ouvrante
     */
    @Override
    public void startElement(String nameSpace, String localName, String qName,
            Attributes attrs) throws SAXException
    {
    }

    /**
     * Actions à réaliser lors de la détection de la fin d'un élément
     */
    @Override
    public void endElement(String nameSpace, String localName, String qName)
        throws SAXException
    {
    }

    /** gestionnaire d'erreur **/
    @Override
    public void error(SAXParseException e) {
        System.err.println("Erreur ligne "+e.getLineNumber()
                +" colonne "+e.getColumnNumber());
        System.err.println(e.getMessage());
    }

    /** gestionnaire d'erreur fatale **/
    @Override
    public void fatalError(SAXParseException e) {
        System.err.println("Erreur fatale ligne "+e.getLineNumber()
                +" colonne "+e.getColumnNumber());
        System.err.println(e.getMessage());
    }
}
```

Le but de cette classe est d'afficher certaines informations du fichier factbook. Par exemple, on veut afficher l'altitude du Mont Blanc (*Montblanc* dans le fichier). Cette information se trouve

dans les attributs d'un élément `<mountain name="Montblanc" height="???">`. Donc il n'y a qu'une méthode à programmer, `startElement`. Elle est appelée pour chaque balise ouvrante du document, mais on va lui demander de ne faire quelque chose que si c'est la balise qui nous intéresse :

```
@Override
public void startElement(...) throws SAXException
{
    // afficher l'altitude du Mont Blanc
    if ("mountain".equals(qName)) {
        if ("Montblanc".equals(attrs.getValue("name"))) {
            System.out.println("Le Mont Blanc mesure " +
                attrs.getValue("height") + " m d'altitude");
        }
    }
}
```

Pourquoi la comparaison de l'élément et surtout de l'attribut sont-elle écrites comme ça ? Imaginez qu'on l'écrive ainsi, pour l'attribut :

```
attrs.getValue("name").equals("Montblanc")
```

et qu'il n'y ait pas d'attribut `name` pour l'un des éléments `mountain` du fichier, pas forcément celui qu'on cherche mais n'importe lequel autre. On aurait une `NullPointerException` sur `null.equals("Montblanc")` au lieu de `"Montblanc".equals(null)` qui est testé sans erreur à faux.

Retenir que les comparaisons d'une chaîne possiblement nulle avec une constante doivent toujours s'écrire ainsi :

```
"constante".equals(variable_nullable)
```

2.3. Travail à faire

Pour chacun des points suivants, on garde le même programme, il suffit d'ajouter des instructions dans la méthode `startElement`.

- Afficher la population de la Bretagne : élément `<province name="Bretagne" population="???">`. NB: on fait l'hypothèse qu'il n'y a qu'une seule province appelée *Bretagne*. Ce serait nettement plus complexe de vérifier le pays.
- Afficher les noms des fleuves dont la longueur est au moins 6000 km : élément `<river length="N" name="???">` avec $N \geq 6000$. Pensez à convertir les chaînes en `int`.
- Afficher le nom de la plus grande île du fichier. C'est un peu plus compliqué : vous allez parcourir tout le fichier à la recherche des éléments `<island name="???" area="???">`. Il faut ignorer l'île lorsque sa surface est inconnue. Mémorisez la taille et le nom de la plus grande vue jusqu'ici – initialisez-les dans la méthode `startDocument()` ci-dessous. À chaque île qui a une surface, vous regardez si cette surface est plus grande que la meilleure jusqu'ici et vous mettez à jour si c'est le cas. Vous ne saurez qu'à la fin du parcours, donc dans la méthode `endDocument()`, laquelle est la plus grande de toutes.

```java
/**
 * Actions à réaliser au début du document
 */
@Override
public void startDocument() throws SAXException
{
    // initialiser le nom et la surface de la plus grande île
}

/**
 * Actions à réaliser à la fin du document
 */
@Override
public void endDocument() throws SAXException
{
    // afficher le nom et la surface de la plus grande île
}
```

- Afficher le nom de la plus ancienne organisation. On trouve l'information dans l'attribut `established` des éléments `<organization name="???"`. C'est le même principe : on doit mémoriser quelle est la plus ancienne organisation rencontrée jusqu'à présent, nom et date, et c'est seulement à la fin du fichier qu'on affiche l'information. La difficulté vient du fait que la date est au format `"jj mm aaaa"`. Il est donc très compliqué de les comparer. Voici comment faire. Le principe est de transformer la date en une sorte de nombre de jours : $(aaaa * 12 + mm) * 31 + jj$ (une année = 12 mois de 31 jours). Le nombre obtenu rend les dates comparables entre elles. Voici maintenant le détail :

```java
String established = attrs.getValue("established");
if (established != null) {
    // séparer en 3 mots JJ MM AAAA
    String[] mots = established.split(" ");
    if (mots.length == 3) {
        try {
            int jj = Integer.parseInt(mots[0]);
            int mm = Integer.parseInt(mots[1]);
            int aaaa = Integer.parseInt(mots[2]);
            int date = (aaaa*12 + mm)*31 + jj;
            ...
        } catch (NumberFormatException ignored) {}
    }
}
```

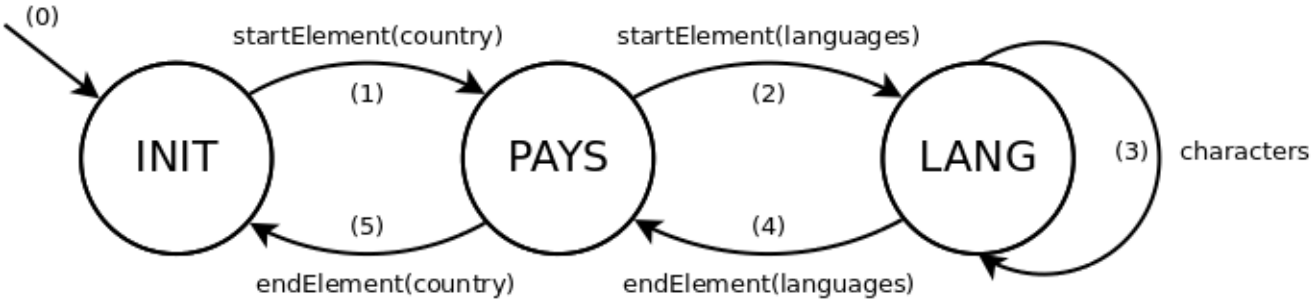
3. Extraction d'informations structurées

Créez une nouvelle classe appelée `FactbookContentHandler2` en reprenant le squelette du début. Et un autre lanceur, `Main2` similaire à `Main1`.

Jusque là, les informations étaient dans le même élément, facile à extraire. Maintenant, on voudrait par exemple afficher les langues parlées dans différents pays. Ce sont des textes contenus dans des sous-éléments `<languages>` de `<country>`. Le problème, c'est de n'afficher que les textes et sous-éléments qu'on veut.

3.1. Automate à états

Au lieu de bricoler, on va programmer proprement une solution qui est extensible. On va définir un automate à états. Il change d'état à chaque balise ouvrante ou fermante qui nous intéresse. Voici par exemple celui qu'on va utiliser pour commencer :



Des traitements sont à faire lors des transitions :

0. C'est l'initialisation de la machine. On indique l'état initial.
1. Si on est devant une balise ouvrante `<country>`, alors on affiche l'attribut `name` et on passe dans l'état `PAYS`,
2. Si on est devant une balise ouvrante `<languages>` alors on passe dans l'état `LANG`,
3. On mémorise le texte (en concaténant les morceaux),
4. Si on est devant une balise fermante `<languages>` alors on affiche le texte mémorisé et on repasse dans l'état `PAYS`,
5. Si on est devant une balise fermante `<country>`, alors on revient dans l'état initial.

Voici comment ça se programme. Il faut un type énuméré et une variable pour représenter les états. Il faut également un `StringBuilder` pour stocker les textes rencontrés.

```
public class FactbookContentHandler2 extends DefaultHandler
{
    // gestion des états de l'automate
    private enum Etat {INIT, PAYS, LANG};
    private Etat EtatCourant;

    // texte en cours de lecture, utiliser texte.toString() pour sa valeur
    private StringBuilder texte = new StringBuilder();

    // nom du pays courant, null s'il a déjà été affiché
    private String pays = null;
```

Un `StringBuilder` est une sorte de chaîne éditable. En Java, les instances de `String` ne sont pas modifiables. Quand on veut concaténer quelque chose ou changer des caractères dans une chaîne, il faut entièrement la recréer. C'est très inefficace. Les `StringBuilder` et `StringBuffer`, au contraire, sont éditables. Par exemple :

```
StringBuilder sb = new StringBuilder();
sb.append("pi = ");
sb.append(3.1415);
return sb.toString();
```

La différence entre `StringBuilder` et `StringBuffer`, c'est que les seconds sont à réserver aux programmes multitâches (*threads*).

On a besoin d'un `StringBuilder` parce que les textes peuvent être fournis en plusieurs morceaux lors de la lecture des fichiers XML. Dans l'automate, le buffer sera vidé à chaque balise ouvrante ou fermante rencontrée.

Voici maintenant l'écouteur pour le début du document, transition n°0 du diagramme. Il n'y a pas d'écouteur pour la fin du document.

```
    /**
     * Actions à réaliser au début du document.
     */
    @Override
    public void startDocument() throws SAXException
    {
        // état initial
        EtatCourant = Etat.INIT;
    }
```

On voudrait faire un affichage assez propre des langues et des pays : un pays par ligne et ses langues séparées par une virgule. C'est moins simple que ça en a l'air. En effet, certains pays n'ont aucune langue, il ne faut pas les afficher du tout. D'autre part, il ne faut pas mettre de virgule pour la première langue qu'on affiche.

C'est le rôle de la variable `pays` de mémoriser le nom du pays. On l'initialise quand on rencontre la balise ouvrante `<country>`. Ensuite, quand on rencontre la balise fermante `</languages>`, on affiche le nom du pays et le nom de la langue (dans `texte`), puis, c'est une astuce, on met le nom du pays à `null`, de façon à ne pas le réafficher pour les autres langues de ce pays.

On arrive au plus complexe, le traitement des balises ouvrantes. Il faut comprendre que c'est la même fonction, `startElement` qui est appelée pour toutes les balises. C'est l'état de l'automate qui sert à distinguer les traitements.

```
    /**
     * Actions à réaliser lors de la détection d'une balise ouvrante
     */
    @Override
    public void startElement(String nameSpace, String localName, String qName,
            Attributes attrs) throws SAXException
    {
        // selon l'état courant
        switch (EtatCourant) {

            case INIT:
```

```
                if ("country".equals(qName)) {
                    // mémoriser le nom du pays
                    pays = attrs.getValue("name");
                    // état suivant
                    EtatCourant = Etat.PAYS;
                }
                break;

            case PAYS:
                if ("languages".equals(qName)) {
                    // état suivant
                    EtatCourant = Etat.LANG;
                }
                break;

            default:
                // on ne fait rien, on ignore cette balise
        }

        // vider le StringBuilder à chaque balise
        texte.setLength(0);
    }
```

La deuxième fonction gère la rencontre d'une balise fermante :

```
    /**
     * Actions à réaliser lors de la détection de la fin d'un élément
     */
    @Override
    public void endElement(String nameSpace, String localName, String qName)
        throws SAXException
    {
        switch (EtatCourant) {

            case LANG:
                if ("languages".equals(qName)) {
                    // si le pays n'a pas déjà été affiché, le faire maintenant
                    if (pays != null) {
                        System.out.print(pays + " : ");
                        // noter qu'il a été affiché, voir le else
                        pays = null;
                    } else {
                        // le pays a déjà été affiché, donc n'afficher qu'une virgule
                        System.out.print(", ");
                    }
                    // afficher la langue
                    System.out.print(texte.toString());
                    // remonter
```

```
                    EtatCourant = Etat.PAYS;
                }
                break;

            case PAYS:
                if ("country".equals(qName)) {
                    // sauter à la ligne si le pays a été affiché
                    if (pays == null) System.out.println();
                    EtatCourant = Etat.INIT;
                }
                break;

            default:
                // on ne fait rien, on ignore cette balise
        }

        // effacer le texte à chaque balise
        texte.setLength(0);
    }
```

Le troisième événement est la rencontre d'un texte quand on est dans l'état `LANG`, c'est à dire entre les balises `<languages>` et `</languages>`. On se contente de l'ajouter au `StringBuilder`. C'est seulement dans `endElement` qu'on l'utilise.

```
    /**
     * Actions à réaliser sur les textes
     */
    @Override
    public void characters(char[] text, int debut, int lng)
    {
        // si on est entre <languages> et </languages>, on stocke le texte
        if (EtatCourant == Etat.LANG) {
            texte.append(new String(text, debut, lng));
        }
    }
```

3.2. Affichage des noms et populations des villes françaises

Commencez par recopier les classes `FactbookContentHandler2` en `FactbookContentHandler3`, et `Main2` en `Main3` en changeant la méthode main pour lancer la bonne classe.

Vous allez maintenant programmer ce qu'il faut pour afficher les noms des villes des provinces françaises et leur population, par province. Le résultat attendu est :

```
Alsace :
    Strasbourg  252338
    Mulhouse    108357
...
```

```
Bretagne :
    Rennes      197536
    Brest       147956
...
```

Les données sont plus difficiles à attraper. Voici leur structure :

```
<country name="France" ...>
    <province name="???" ...>
        <city...>
            <name>???</name>
            <population>???</population>
        </city>
    </province>
</country>
```

Vous allez devoir définir un automate un peu plus complexe. Il devra détecter l'entrée dans l'élément `<country name="France">` : il faut tester à la fois le nom de l'élément et s'il a un attribut name égal à *France*, puis l'entrée dans les sous-éléments `<province>` pour afficher leur nom, puis les sous-éléments `<city>`, puis `<name>` d'un côté et `<population>` de l'autre.

Remarquez qu'il y aurait un problème si les éléments `<population>` étaient avant les éléments `<name>` ou s'il y avait plusieurs populations pour la même ville. Il faudrait alors stocker les valeurs et ne les afficher qu'à la sortie de `<city>`.

Pour finir, faire afficher la somme des populations des villes rencontrées, à la fin du traitement. Remarquez que cette population totale (8 833 289) est nettement inférieure à la population du pays, puisque seules les grandes villes sont comptées.

4. Travail à rendre

Ouvrez le workspace dans le navigateur de fichiers. Descendez à l'intérieur du répertoire `TP7`. Cliquez droit sur le dossier `src`, choisissez `Compresser...`, cliquez sur `Créer`. Ça va créer une archive `src.tar.gz` ou `src.zip`. Déposez cette archive sur Moodle, dans la page de cours dédiée L4IN121T Formats et traitements de données internet.