**DOM PARSER**: utiliser par exemple pour faire le mapping objet XML.

Diffèrent package utiliser sont : javax.xml.parsers et org.w3c.dom :

Pour charger un document XML en utilisant java dom parser il faut passer par désigne patterns factory :

DocumentBuilderFactory factory = DocumentBuilderFactory.newDefaultInstanse() ;

DocumentBuilder documentBuilder = factory.newDocumentBuilder() ;

Document document = parse(‘’emplacement du fichier ‘’) ;

Objet document se compose de plusieurs objets Node de diffèrent type :

* 1: If node is an element node.
* 2: If node is an attribute node.
* 3: If node is a text node.
* 8: If node is a comment node.

D’autre objet :

**NodeList**: est un objet qui continent un ensemble des nœuds, getChildNode() ou getElementByTagName() retourne un objet de type NodeList.

**NameNodeMap** : Objet qui contient une map des nœuds, la clé est une chaine de caractère et la valeur est un nœud, la méthode getAttributes retourne un objet NamedNodeMap.

**Différentes fonctions de la classe Node :**

* **getDocumentElement()** : appeler sur l’objet **Document** pour retourner la racine du document.
* **getNodeName()** : retourne le nom de l’élément XML, en cas du nœud text elle retourne #text, en cas du commentaire elle retourne #comment.
* **getNodeType() :** retourne le type du nœud .
* **getNodeValue() :** retourne la valeur du nœud, le cas d’attribut il retourne sa valeur, et le cas du nœud texte elle retourne le texte intérieur.
* **getChildNodes() :** retourne un objet de type NodeList qui contient toutes les nœuds fils du noud questionné.
* **getElementByTagName(‘’tagName’’) :** utiliser sur un objet **Document** pour retourner une **NodeList** des éléments qui porte le tagName passer en paramètres.
* **item(indice i)** : utiliser sur un objet de type **NodeList** pour retourner objet **Node** de l’indice i ;
* **getAttributes()** : renvoie un objet **NamedNodeMap** qui contient tous les attributs d’un objet **Element.**
* **getNamedItem(‘’attrName’’) :** renvoie un objet **Node** de type **attribute** qui porte le nom du même attribut passé en paramètre.
* **getParentNode()** : retourne le nœud parent de l’objet **Node** appelant.
* **getNextSibling() :** retourne le nœud suivant de même niveau de l’objet **Node** appelant.
* **getPreviousSibling() :** retourne le nœud suivant de même niveau de l’objet **Node** appelant.
* **appendChild( Node n)** : permet d’ajouter le nœud n au fils de l’objet **Node** appelant.
* **removeChild(Node n) :** utiliser sur un objet **Node** pour supprimer son nœud fils qui est passé en paramètre.
* **replace(Node newChild,Node oldChild)**: permet de remplacer un nœud fils par un autre.
* **setAttribute(String name, String value)**: permet d’ajouter ou modifier un attribut, utiliser que sur un objet de type **Element.**
* **setTextContent(String text) :** utiliser sur un objet **Element** pour modifier son contenu.
* **getTextContent()** : récupère le contenu textuelle de l’objet **Element** appelant.
* **createElement(‘elementName’) :** utiliser sur l’objet **Document** pour crée un Objet **Element** qui porte le nom passer en paramètre

**Exemple d’utilisation :**

**public** **class** RacineDao {

**private** Document document;//c'est le document XML ouvert au niveau de la mémoire

**private** Transformer transformer;//Utiliser pour rendre les modification permanente dans le document XML

**private** String path = "resources/racine.xml";//Chemin du fichier

**private** StreamResult file;//Utiliser par transformer pour pointer sur le fichier physique

**private** DOMSource source;//Utiliser par transformer pour pointer sur le document au niveau de la mémoire

**public** RacineDao() {

**try** {

//Instanciation du document à partir du chemin relatif

document = DocumentBuilderFactory.*newDefaultInstance*().newDocumentBuilder().parse(path);

//Instanciation du transformer

transformer = TransformerFactory.*newDefaultInstance*().newTransformer();

//Pointer sur le fichier physique

file = **new** StreamResult(path);

//Pointer sur le document au niveau de la memoire

source = **new** DOMSource(document);

} **catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

/\*l'idée ici de trouver tous les éléments de type dossier qui ont l'attribut id égale au

\* paramètre id et de le supprimer, pour ça on récupère tous les éléments dossier par

\* getElementByTagName("dossier") qui est appliquer que sur l'objet document, et ensuite on

\* parcours cette liste élément par élément, et que pour chaque élément on obtient la liste de

\* ses attributs, et on fait appel à l'attribut id et on vérifier si son contenu égale

\* au paramètre id, si oui on fait appelle au nœud père de l'élément courant et on lui applique

\* removeChild() avec l'élément courant comme paramètre\*/

**public** **void** removeFolder(String id) {

NodeList folders = document.getElementsByTagName("dossier");

**for**(**int** i = 0; i < folders.getLength();i++) {

NamedNodeMap attributes = folders.item(i).getAttributes();

**if**(attributes.getNamedItem("id").getNodeValue().equals(id)) {

System.***out***.println("element found");

Node father = folders.item(i).getParentNode();

father.removeChild(folders.item(i));

}

}

save();

System.***out***.println("done");

}

/\*Pour ajouter nouveau dossier il faut créer un élément dossier avec createElement

\* qui est applicable que sur l'objet document, ensuite on ajoute les attributs nessecaire par

\* setAttribute(), puis on récupère la racine avec getDocumentElement() qui est applicable aussi que

\* sur l'objet document, et finalement on fait appelé à la méthode appendChild pour ajouter le

\* nouveau élément dossier a la racine \*/

**public** **void** addFolder(Folder f) {

Node racine = document.getDocumentElement();

Element folderNode = document.createElement("dossier");

folderNode.setAttribute("id", f.getId());

folderNode.setAttribute("nom", f.getNom());

folderNode.setAttribute("dateDeCreation", f.getDateCreation());

racine.appendChild(folderNode);

save();

}

/\*Méthode save et tout simplement fait appelle à la méthode transform de transformer

\* on lui passant le fichier physique et le document qui est dans la mémoire\*/

**private** **void** save() {

**try** {

transformer.transform(source, file);

} **catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

**public** **static** **void** main(String[] args) {

RacineDao rDao = **new** RacineDao();

rDao.addFolder(**new** Folder("d11", "Games", "2018-11-01"));

}

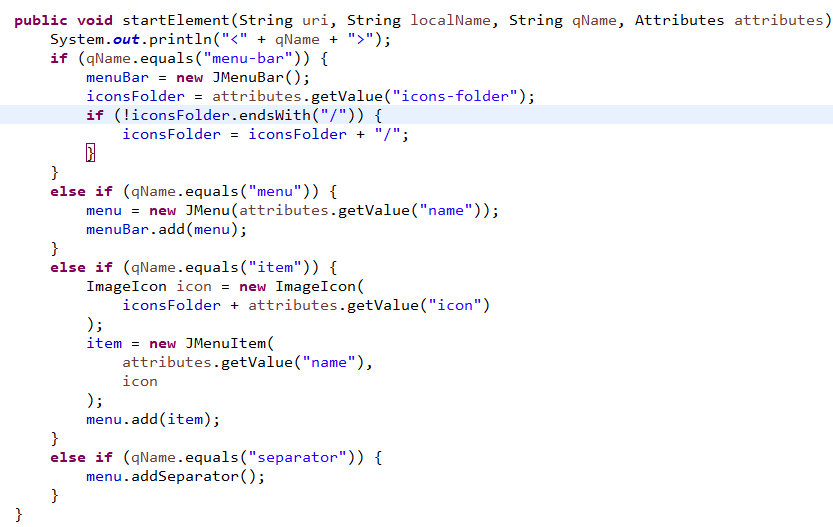
}

***Instancier un document XML au niveau de la mémoire et opération d’ajoute et de suppression***

**SAX PARSER** : pour travailler avec sax parser il faut tout d’abord que votre class hérite de la classe DefaultHandler, après dans le constructeur de la classe qui a comme paramètre une string, il faut passer par 2 objets pour que vous pouvez enfin parser le document XML :

* SAXParserFactory factory = SAXParserFactory.newDefaultInstance();
* SAXParser parser = factory.newSAXParser();
* parser.parse(source, this); 🡪 Objet this c’est l’objet De la classe DefaultHandler (heritage).

Lorsqu’on hérite de la classe DefaultHandler et pour extraire les donnée XML, on travaille on majorité avec la redéfinition de la méthode startElement, qui point chaque fois sur un élément, pour associer le traitement convenable de chaque élément, il faut passer par des if else imbriqué sur le qNam qui contient le nom de l’élément encours, exemple :



attribute.getValue(‘’nomAttribut’’) 🡪 permet de récupérer la valeur de l’attribut passer en paramètre de l’élément encours.

La méthode characters() permet de récupéré le texte qui est entre les éléments, à chaque passage d’un élément á un autre elle change de contenu, elle est appelé plusieurs fois dans le même fichier, exemple :

