TD Algorithmique Avancée : Programmation JAVA/XML

Tutoriel n° 1 : Découverte de l'API SAX (Simple API for XML : org.xml.sax)

SAX (**S**imple **A**PI for **X**ML) est une API **de type événementielle et incrémentale**.
Elle génère différents types d'événements lorsqu'elle parcourt le fichier XML.
L'utilisateur peut alors recevoir ces événements et adopter le comportement approprié.
SAX ressemble à la programmation événementielle des Interfaces Graphiques comme **Swing**.

SAX est une API écrite par David Megginson (http://www.megginson.com/SAX).
SAX *n'est pas une API du W3C* mais elle est largement utilisée (**développement communautaire**)
Le site officiel de SAX est : http://www.saxproject.org/about.html (ou http://sax.sourceforge.net/)

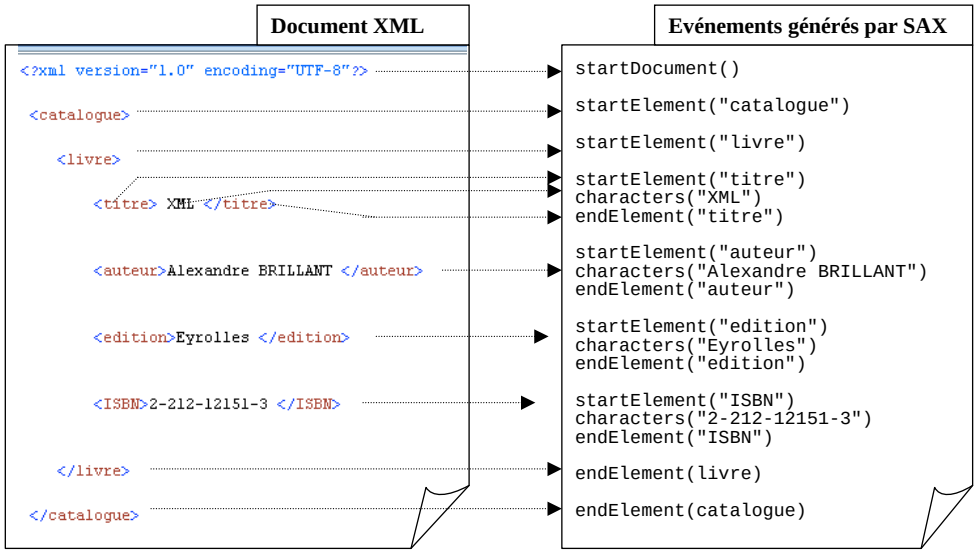
➧ **SAX** (**S**imple **A**PI for **X**ML) est une API **incrémentale.**
En effet, SAX parcourt le document XML ***une seule fois de manière séquentielle***.

➧ **SAX** (**S**imple **A**PI for **X**ML) est une API **de type événementielle**.

…mais ***quels sont les "événements" déclenchés par le parser SAX à la lecture d'un fichier XML***?
Par exemple, une balise ouvrante sera à l'origine d'un événement, une balise fermante sera à l'origine d'un autre…
Les différents types d'événement sont donc liés :

- au début et à la fin du document
- au début et à la fin d'éléments
- à la présence d'attributs, de texte, …

Prenons par exemple le document XML ci-dessous et notons à droite dans l'ordre les principaux événements générés par le **parser SAX** lors de la lecture de ce document :

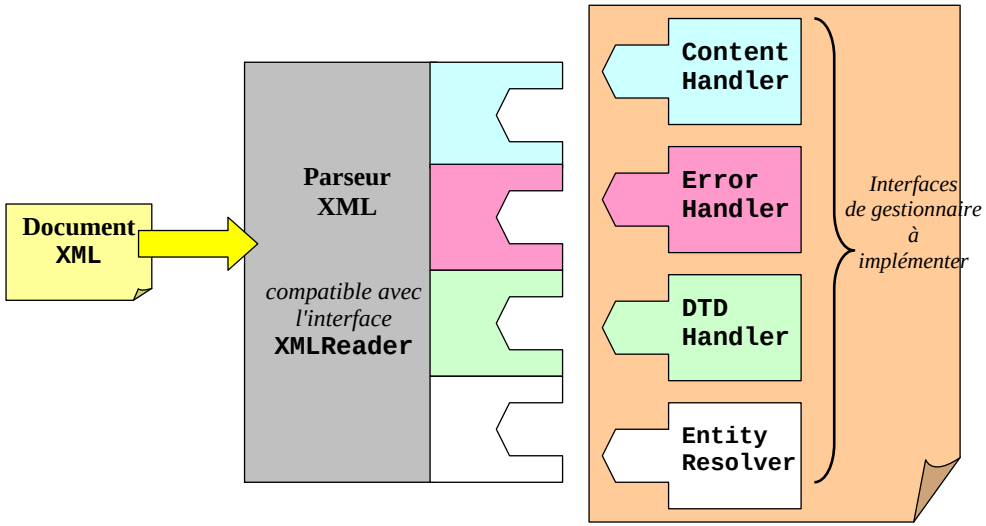


Le **parser SAX** lit le document et déclenche des événements au fur et à mesure.
Pour chaque événement une méthode (`startDocument, startElement, characters, endElement,…`) est invoquée automatiquement par le parser SAX.
Dans cet exemple, nous venons de voir que les événements sont générés par un parser SAX : ils sont donc liés au contenu du fichier XML : c'est donc un ***gestionnaire de contenu*** (**`ContentHandler`**) qui aura la charge de traiter ces événements en implémentant les méthodes invoquées (`startDocument, startElement, characters, endElement,…`)

➢ 1.Fonctionnement de SAX (Simple API for XML)

Le principe de fonctionnement de **SAX** est le suivant :

- un **parser** va parcourir le fichier XML de manière séquentielle. Au cours de cette lecture du fichier, le parser va déclencher des événements.
- Pour que l'application puisse réagir aux **événements SAX envoyés par le parser**, il est nécessaire d'enregistrer des **gestionnaires** auprès du parser qui contiendront les traitements à appliquer
  Sur le même principe que *Swing* et ses *interfaces d'écoutes* (`XXXListener`), **SAX** fournit des ***interfaces de gestionnaire*** (`XXXHandler`)qu'il suffit d'implémenter pour traiter les événements. Ainsi à chaque événement est associé un gestionnaire (**`handler`**) qui est exécuté lorsque l'événement survient.



SAX fournit donc un ensemble d'interfaces disponibles dans le **paquetage** **`org.xml.sax`**.
Les interfaces fondamentales de ce paquetage sont :

`org.xml.sax.XMLReader`	**l'interface fondamentale d'accès à un parser**
`org.xml.sax.ContentHandler`	**le gestionnaire de contenu** qui listent les méthodes invoquées suite à des événements XML. C'est ce gestionnaire qui permet le véritable travail d'analyse du fichier XML.
`org.xml.sax.ErrorHandler`	**le gestionnaire d'erreurs** qui listent les méthodes invoquées suite à des erreurs.
`org.xml.sax.DTDHandler`	**le gestionnaire de DTD** qui listent les méthodes liées à l'analyse de la DTD par le parser. (notification des déclarations d'entités)
`org.xml.sax.EntityResolver`	utilisée pour le **traitement des entités** en indiquant au parser comment trouver certaines ressources.

Voir javadoc

SAX s'organise en réalité en 3 paquetages. Ces paquetages sont fournis par défaut dans la version standard du JDK (pas de **`jar`** à ajouter pour le moment, et la documention de ces paquetages est disponible en ligne sur le site de sun comme toute API standard : http://download.oracle.com/javase/6/docs/api/) :

`org.xml.sax`	où se trouvent les classes fondamentales, c'est-à-dire le cœur de l'API SAX. Il s'agit essentiellement d'interfaces dont une partie est présentée ci-dessus
`org.xml.sax.ext`.	où se trouvent les classes étendues de SAX2 (optionnelles)
`org.xml.sax.helpers`	où se trouvent les classes utilitaires comme **`XMLReaderFactory, DefaultHandler`**

➢ 2. le parser SAX : interface `org.xml.sax.XMLReader`

Le **parserSAX** de l'application sera une instance d'une classe implémentant l'interface **`org.xml.sax.XMLReader`**

Rappelons que **SAX** ne fournit dans le **paquetage** **`org.xml.sax`** que des interfaces (c-a-d que des définitions). Dans le JDK par défaut, il n'y a donc pas d'implémentation utilisable pour SAX.

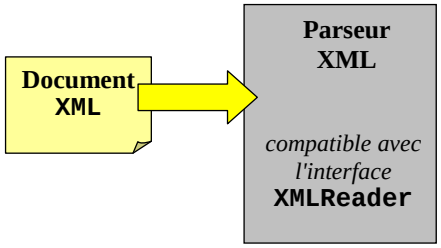
L'implémentation est laissée à différents processeurs XML (appelés aussi éditeurs XML) qui fournissent un parseur **SAX** implémenté (c-a-d instanciable et utilisable) respectant au minimum le contrat SAX défini par les interfaces du paquetages **`org.xml.sax`**. Par exemple, nous pouvons citer :

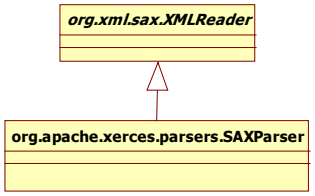
- **`Xerces`** développé par **Apache** (disponible sous http://xerces.apache.org/)
- **`MSXML`** développé par **Microsoft**

Il existe d'autres parseurs que l'on peut télécharger sur le web.
Tous ces parseurs apportent pour la plupart les mêmes fonctionnalités. Ils se distinguent en terme de rapidité, facilité d'utilisation, etc …

Dans le cadre de ce tutoriel, nous utiliserons le parser **`Xerces`** (http://xerces.apache.org/xerces2-j/).

➢ 2. 1 : Etape n°1 : Instanciation d'un parser SAX





Le code nécessaire à l'instanciation d'un parseur est spécifique pour chaque processeur XML.
Dans le cadre du processeur XML **`Xerces`**, c'est la classe **`org.apache.xerces.parsers.SAXParser`** qui implémente l'interface `org.xml.sax.XMLReader` : obtenir un parser SAX pour notre application reviendra donc à obtenir une instance de la classe **`org.apache.xerces.parsers.SAXParser`**

→ Pour instancier un parser SAX, la première méthode qui nous vient à l'esprit est d'appeler directement le constructeur de parser proposé par l'éditeur choisi. Dans le cas de **Xerces**, cela revient à écrire :

```
XMLReader saxParser = new org.apache.xerces.parsers.SAXParser();
```

Cette instruction, bien que très simple, est dans sa syntaxe fortement liée au processeur XML choisi (**`xerces`** dans notre cas). Il est nécessaire d'importer spécifiquement l'implémentation du parser choisi.

→ Afin d'éviter d'instancier directement un parser particulier, une autre méthode consiste à utiliser une **factory**.
La classe de l'analyseur apparaît alors en tant que paramètre afin de favoriser une meilleure portabilité du code.
C'est bien sûr cette méthode que nous adopterons par la suite, d'autant plus que dans **SAX** tout est déjà prévu pour.

En effet, la classe **`XMLReaderFactory`** est proposée par la distribution officielle de SAX dans le paquetage **`org.xml.sax.helpers`**. Cette classe permet de créer un **`XMLReader`** avec la méthode **`createXMLReader`** qui est susceptible de déclencher une **`SAXException`**.

***Travail à faire :* Etape n°1 : Instanciation du parser XML SAX.**

➢ Sous **Eclipse** ouvrir votre espace de travail et créer un **nouveau projet** que vous appelerez **`ProgrammationXML`**.

➢ Dans ce projet, importer le fichier **`TestSAX.java`** disponible sur la zone libre.
L'ouvrir. Vous retrouvez le code suivant :

```
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;


public class TestSAX {

   // Constructeur
   public TestSAX(){
      // Etape n°1: Instanciation d'un parseurSAX à l'aide d'une factory
      try {
      XMLReader saxParser =
XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
      } catch (SAXException e) {
         System.out.println("Erreur d'analyse: "+ e.getMessage());
      }
   }


   //Test
   public static void main(String[] args) {
         TestSAX monPremierSAX = new TestSAX();

         System.out.println("Fin du Test !!! ");
   }
```

➢ **Exécuter ce code.**
Une exception `java.lang.ClassNotFoundException: org.apache.xerces.parsers.SAXParser` se déclenche, c'est qu'il vous faut ajouter au **`classpath`** de l'application le **`jar`** du processeur XML **`xerces`**.

En effet, profitons-en pour rappeler que les paquetages **`org.xml.sax, org.xml.sax.ext, org.xml.sax.helpers`** sont présents dans la version standard de java (pas de **`jar`** à ajouter)

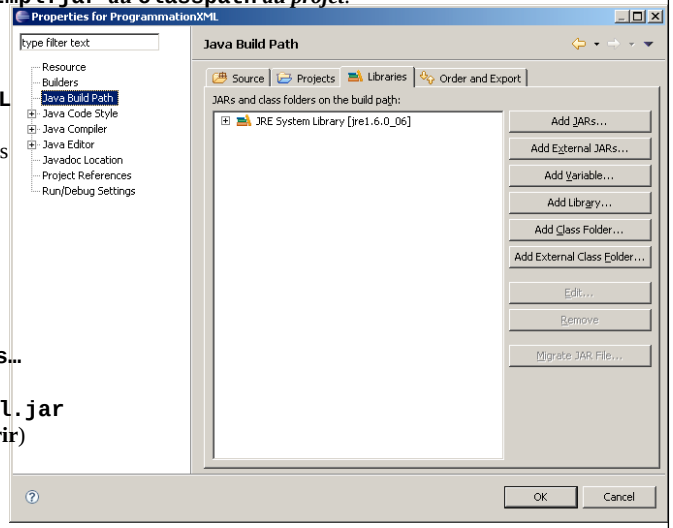
Mais dès lors que nous faisons un ***choix*** sur un ***processeur XML***, il est nécessaire pour l'utiliser d'ajouter son fichier **`.jar`** dans le **`classpath`**.
Pour pouvoir utiliser **`Xerces`** dans notre projet, il est nécessaire d'inclure le fichier **`xercesImpl.jar`** dans le **`classpath`.**

Ici nous utilisons Xerces, nous allons donc devoir ajouter le fichier **`xercesImpl.jar`** disponible sur la zone libre à notre **`classpath`**.

Où trouver le fichier `xercesImpl.jar` depuis votre ordinateur personnel ?

- Ce fichier est bien sûr disponible sur la zone libre.
- Depuis votre ordinateur personnel, vous le récupérer sur le site d'Apache consacré au parser **`Xerces`** pour Java (http://xerces.apache.org/xerces2-j/). Aller dans la partie **`Download`** et télécharger le fichier de type **`Xerces-J-bin.x.x.x.tar.gz`** qui correspond au **`Latest binary release`** (c-a-d à la dernière version binaire). Enregistrer ce fichier, puis décompacter-le. Vous pouvez alors récupérer alors le fichier **`xercesImpl.jar`**.

Travail à faire : *Ajouter* **`xercesImpl.jar`** *au* **`classpath`** *du projet.*



↳ Sous Eclipse, placez-vous dans la vue **Package** sur le projet **`programmationXML`**

Effectuer un clic droit avec la souris et choisir l'option **`Properties`**
La fenêtre ci-contre doit s'ouvrir.

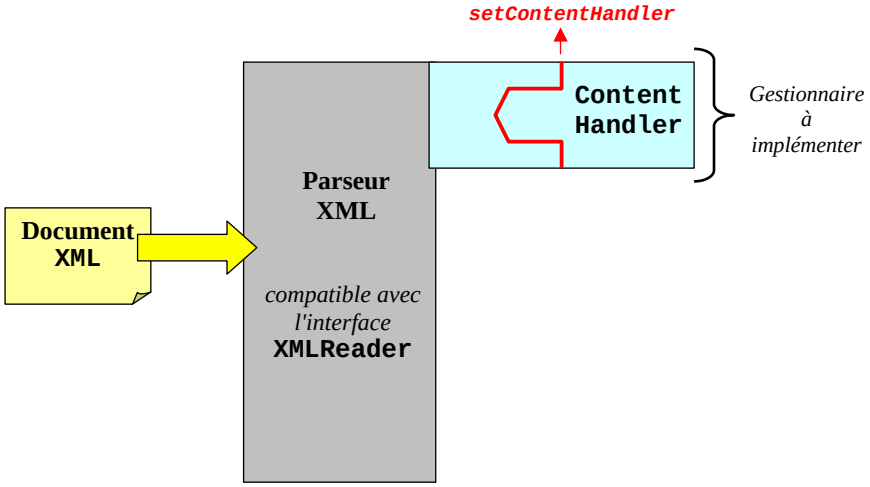
↳ Sélectionnez à gauche
`Java Build Path`

↳ Puis, cliquez à droite sur le bouton **`AddExternalJARs…`**

Rechercher le fichier **`xercesImpl.jar`** et sélectionner-le (cliquer sur **Ouvrir**)
Validez en cliquant sur **`OK`**.

↳ Exécuter à nouveau le code du fichier **`TestSAX.java`.** Aucune erreur ne doit apparaître cette fois-ci dans la console.

➢ 2. 2 : Etape n°2 : Enregistrement d'un gestionnaire de contenu auprès du parseur SAX choisi.



L'enregistrement d'un gestionnaire se fait auprès du parseur.
La méthode **`setContentHandler`** permet d'enregistrer un ***gestionnaire de contenu***.

```
void setContentHandler(ContentHandler handler)
    Allow an application to register a content event handler.
```

Cette méthode attend en paramètre qu'on indique une **classe concrète** de type `ContentHandler` qui s'occupera réellement de la gestion des événements. Cette classe, que nous devons écrire, doit implémenter l'interface `ContentHandler` proposée par SAX (polymorphisme).
Dans notre exemple, nous appelerons cette classe **`MonContentHandler.`**

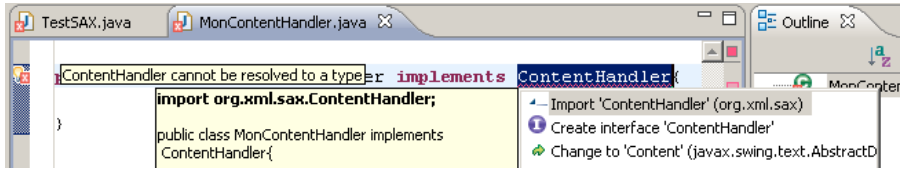
Travail à faire : **Etape 2 : Enregistrement du gestionnaire de contenu auprès du `parser` SAX choisi.**

↳ Dans le constructeur de la classe **`TestSAX`**, rajouter dans le bloc **`try`** en dessous du code déjà écrit l'instruction suivante :

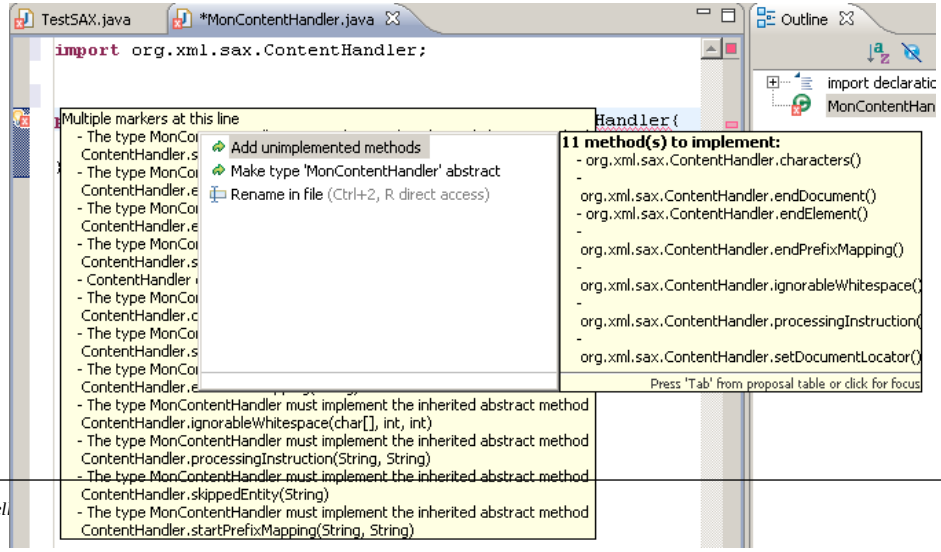
```
try {
        …
        //Enregistrement d'un gestionnaire de contenu concret
        saxParser.setContentHandler(new MonContentHandler());
    }
```

↳ Vous devez donc créer dans le projet **`ProgrammationXML`**, une nouvelle classe **`MonContentHandler`** qui implémente **`ContentHandler`**.

Afin de supprimer, les erreurs de compilation, aidez-vous de la "croix rouge" d'Eclipse :
- un premier clic vous permettra de rajouter : `import org.xml.sax.ContentHandler;`



- un second clic vous permettra de transformer la classe **`MonContentHandler`** en une classe concrète en obtenant une redéfinition (vide pour le moment) de toutes les méthodes de l'interface **`ContentHandler`** .



Pour l'instant, on ne rajoute aucun code dans la classe **MonContentHandler** .
Contentez-vous de sauver les fichiers **TestSax** et **MonContentHandler**, et d'exécuter **TestSax**.
Vous devez voir s'afficher le message : **Fin du Test !!!**

➢ 2. 3 : Etape n°3 : Création du gestionnaire de contenu par implémentation de l'interface ContentHandler

La classe **MonContentHandler** est la classe principale de notre application.
Elle joue le rôle de gestionnaire de contenu en traitant les événements SAX : les traitements devront être implémentés dans les méthodes issues de l'interface **ContentHandler.**

Afin de rendre la classe **MonContentHandler** concrète, l'aide d'Eclipse nous a permis de rajouter les 11 méthodes (vides pour l'instant) définies de l'interface **ContentHandler**.
Il faut maintenant implémenter ces méthodes pour donner un comportement à notre gestionnaire de contenu.

Vous pouvez consulter rapidement **l'annexe 1** pour avoir un aperçu des 11 méthodes déclarées dans **ContentHandler.**

Pour commencer, nous nous intéresserons aux 4 méthodes suivantes :

↳ Tout d'abord, aux 2 méthodes qui se déclenchent respectivement au début et à la fin d'un document XML.

`void startDocument() throws SAXException`	Receive notification of the beginning of a document.
`void endDocument() throws SAXException`	Receive notification of the end of a document

↳ Ensuite, aux 2 méthodes qui se déclenchent respectivement à l'ouverture et à la fermeture d'un élément.

`void startElement(String uri, String localName, String qName, Attributes atts) throws SAXException`	Receive notification of the beginning of an element.
`void endElement(String uri, String localName, String qName) throws SAXException`	Receive notification of the end of an element.

Avec comme paramètres :

- **uri** qui représente l'URI **(U**niform **R**essource **I**dentifiers**)** de l'espace de nom. Les espaces de noms sont une recommandation W3C depuis janvier 1999. Ils permettent de qualifier de manière unique des éléments et des attributs définis dans un document XML.

- **localName** qui représente le nom local de la balise (sans préfixe)
- **qName** qui représente le nom qualifié de la balise (avec préfixe) de la forme **uri:localName**

On se contentera d'afficher le paramètre **localName.**

Remarque sur le nom local (ou nom simple) et le nom qualifié :
Après la définition du "namespace" (espace de nom) suivant :
xmlns:catalog="http://www.mon.site/monCatalog/catalog.dtd
Le préfixe « **catalog** » peut-être indiqué pour préciser d'ou provient l'élément : `<catalog:livre>`
S'il existe un autre « namespace » comme « **commande** » , la commande de livres : `<commande:livre>`

- **atts** qui représente la collection des attributs de l'élément.
  Remarque: Comme indiqué dans l'annexe 2, l'interface **Attributes** propose plusieurs méthodes pour obtenir des informations sur la liste d'attributs. Par exemple, nous pouvons citer :
  - `getLength()` qui renvoie le nombre d'attributs
  - `getQName(int i)`qui renvoie le nom de l'attribut numero i
  - `getValue(int i)` qui renvoie la valeur de l'attribut numero i
  - `getValue(String name)` qui renvoie la valeur de l'attribut name
  - `getType(int i)` qui renvoie le type de l'attribut

***Travail à faire :* Etape n°3 : Implémentation du gestionnaire de contenu MonContentHandler**

↳ Dans le fichier **MonContentHandler.java**, implémenter comme indiqué ci-dessous les méthodes **startDocument, endDocument, startElement** et **endElement.**
Ce code est disponible sur la zone libre dans le fichier **DebutImplementationCH.txt**

```
// Evenement genere au debut du document
 public void startDocument() throws SAXException {
      System.out.println("----------------------");
       System.out.println("Debut de l'analyse du document XML");
       System.out.println("----------------------");
 }

 // Evenement genere en fin de document
 public void endDocument() throws SAXException {
       System.out.println("----------------------");
        System.out.println("Fin de l'analyse du document" );
        System.out.println("----------------------");
 }

 //Evenement genere a chaque fois que l'analyseur rencontre une balise xml ouvrante
 public void startElement(String uri, String localName, String name,
        Attributes atts) throws SAXException {

         System.out.println();
         System.out.println("----------------------");
          System.out.println(" Ouverture de la balise " + localName );

          // s'il existe un espace de nom
          // cela ne sera pas le cas dans nos exemples : testSAXi.xml
          if  ( ! "".equals(uri)){
            System.out.println("--> appartenant a espace de nommage : "  + uri);
            System.out.println("--> nom qualifié de la balise : " + name);
            }

          // A propos des attributs de la balise
          int nbAttributs = atts.getLength();
    if (nbAttributs ==0) System.out.println("--> pas d'attributs pour cette balise");
          else
            {
            System.out.println("--> Détail des attributs de cette balise : ");
            //parcours de la liste des attributs
            for (int index = 0; index < nbAttributs; index++) {
                  System.out.println("     -> " +  atts.getLocalName(index) + " = " +
atts.getValue(index));
                }
            }
 }

 //Evenement généré a chaque fermeture d'une balise.
  public void endElement (String uri, String localName, String name)
 throws SAXException {

      System.out.println();
      System.out.println("----------------------");
       System.out.println(" Fermeture de la balise " + localName);

 }
```

↳ Enregistrer le fichier **MonContentHandler.java** et exécuter le fichier **TestSAX.java**
Dans la console, vous ne constatez rien de plus que le message habituel : **Fin du Test !!!**
En effet, il manque l'étape qui consiste à lancer le début de lecture du document XML par le parser …

➢ 2. 4 : Etape n°4 : Lier le parser à un fichier XML pour lancer la lecture …

Jusqu'à présent dans le constructeur du fichier **TestSAX.java**, nous avons :
- instancier un **parser** à l'aide d'une factory
- enregistrer un gestionnaire de contenu auprès du parser

… Il ne nous reste plus qu'à demander au parser de démarrer la lecture du document XML…
La méthode **parse** permet de lier le parser à un document XML.
Le début de lecture du document XML est déclenché dès l'appel à la méthode **parse**.
C'est donc cette méthode qui va permettre à SAX de déclencher les événements qui seront ensuite traités par le gestionnaire de contenu. **La méthode parse est donc indispensable pour visualiser le fonctionnement de notre gestionnaire de contenu !**

***Travail à faire :* Etape n°4 : Déclencher la lecture du document XML par le parser**

↳ Sous Eclipse, placez-vous dans la vue **Package** sur le projet **programmationXML**
Effectuer un clic droit avec la souris et choisir l'option **Import,** puis sélectionner **File System** et sélectionner les fichiers **testSAX1.xml** et **testSAX2.xml** disponibles sur la zone libre afin de l'importer-les dans votre projet.

↳ Dans le constructeur de la classe **TestSAX**, rajouter dans le bloc **try** en dessous du code déjà écrit l'instruction suivante :

```
try {
     //… à la suite du code déjà écrit …

    //Lancement de  l'analyse de la source XML par le parser
    saxParser.parse("testSAX1.xml");
    }
//… après  le catch SAXException, ne pas oublier d'attraper les IOExceppions
catch (IOException e) {
    System.out.println("Erreur de lecture: "+ e.getMessage());
    }
```

↳ Enregistrer et exécuter le fichier **TestSAX.java**.

↳ Changer le fichier parsé `testSAX1.xml` par le `testSAX2.xml` . Compiler et exécuter pour visualiser le traitement des attributs de la balise **livre**.

Remarque : A propos de la surcharge de la méthode **parse** de l'interface **org.xml.sax.XMLReader**

void **parse**(InputSource input)
Parse an XML document.

↳ Pour lier un parser à un fichier XML, la méthode **parse** la plus générale de la classe XMLReader.avec un `InputStream` comme paramètre.
Un **org.xml.sax.InputSource** est un objet analogue à un **java.io.InputStream** mais plus général, puisqu'il encapsule un flux de lecture sur : – un flux d'octets (bytes)
– un flux de caractères
– une connexion HTTP
– un fichier XML local

Exemple d'utilisation:
```
InputSource source= new InputSource("testSAX2.xml");
saxParser.parse(source);
```

void **parse**(String systemId)
Parse an XML document from a system identifier (URI).

↳ Une seconde méthode **parse** est disponible dans la classe **XMLReader**. C'est celle que nous avons utilisée pour notre exemple. C'est un raccourci de la première en passant directement l'**uri** de la source. L'appel de cette méthode revient à faire :

```
parse(new InputSource(systemId));
```


↳ 2. 5. : Etape n°5 : Valider le document XML par un schéma.

Pour valider le document **XML**, il faut activer une **feature** du parser.
En ce qui concene le parser Xerces, la liste des features est consultable sur : http://xerces.apache.org/xerces-j/features.html
Pour valider un schèma, il faut donc utiliser une instruction du type :
```
saxParser.setFeature("http://apache.org/xml/features/validation/schema", true);
```

Pour choisir le nom du schéma, il faut utiliser la méthode `setProperty`.
Pour le parser Xerces, la liste des propriétés est consultable sur:
http://xerces.apache.org/xerces-j/properties.html
Pour utiliser un schéma XML sans namespace, il faudra écrire une instruction du type :

org.xml.sax
Interface XMLReader

void **setFeature**(java.lang.String name, boolean value)
Set the value of a feature flag.

void **setProperty**(java.lang.String name, java.lang.Object value)
Set the value of a property.

```
saxParser.setProperty("http://apache.org/xml/properties/schema/external-
                noNamespaceSchemaLocation",URLDuSchemaEnString);
```

***Travail à faire :* Etape n°5 : Valider un document XML par un schéma**

↳ Importer dans votre projet **programmationXML**, les fichiers **testSAX3.xml** et **catalogue.xsd** disponibles sur la zone libre.

↳ Dans le constructeur de la classe **TestSAX**, rajouter dans le bloc **try** en dessous du code déjà écrit l'instruction suivante et exécuter…

```
try {
     XMLReader saxParser =
XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
     saxParser.setFeature("http://apache.org/xml/features/validation/schema", true);
     URL xsd = new File("catalogue.xsd").toURI().toURL();
     saxParser.setProperty("http://apache.org/xml/properties/schema/external-
noNamespaceSchemaLocation",xsd.toString());

//Enregistrement d'un gestionnaire
//…code déjà écrit…
}
```

➢ 2. 6 : Etape n°6 : Compléter l'implémentation du gestionnaire de contenu.

↳ Etape n°6 : Comment récupérer le texte ?

La méthode **characters** de l'interface **ContentHandler** se déclenche chaque fois que l'analyseur rencontre des caractères entre deux balises.

void **characters**(char[] ch, int start, int length)
Receive notification of character data.

Les caractères sont stockés dans la chaine **ch** à la borne inférieure (**start**) et ils sont au nombre de **length.**
Il faut donc construire une chaîne en tenant compte de ces bornes pour obtenir les données à afficher :

```
new String(ch,start,length)
```

***Travail à faire :* Etape n°6 : Récupérer le texte des balises**

↳ Dans le fichier **MonContentHandler.java**, surcharger maintenant la **characters.**

```
// Evenement genere à la rencontre de texte entre deux balises
public void characters(char[] ch, int start, int length)
       throws SAXException {
       String content = new String (ch,start,length);
       System.out.println("--> caractères rencontrés : " + content);
}
```

↳ Enregistrer et exécuter le fichier **TestSAX.java**.

En examinant le résultat obtenu sur la console, vous pouvez constater que tous les caractères blancs (même ceux séparant les balises) sont perçus comme faisant partie d'un texte.

Pour épurer le texte envoyé par le parseur, vous pouvez modifier le code de la méthode characters de la manière suivante :

```
String content = new String (ch,start,length);
String text = content.trim();
if (text.length()!=0) System.out.println("--> caractères rencontrés : " + content);
```

➢ 2. 6 : Etape n°7 : Ajouter un gestionnaire d'erreur.

Remarque : Pour l'instant dans notre exemple, nous ne nous intéressons qu'au gestionnaire de contenu, mais comme l'indique la *javadoc* il existe des méthodes équivalentes à **`setContentHandler`** pour chaque handler (gestionnaire).

void	**setDTDHandler**(DTDHandler handler) Allow an application to register a DTD event handler.
void	**setEntityResolver**(EntityResolver resolver) Allow an application to register an entity resolver.
void	**setErrorHandler**(ErrorHandler handler) Allow an application to register an error event handler.

Le gestionnaire d'Erreur : `ErrorHandler`

Le gestionnaire d'erreur (**`ErrorHandler`**) permet de gérer les erreurs rencontrées lors du parsing.
Il permet, par exemple, d'intercepter des erreurs dues à un document XML mal formé . Il propose 3 méthodes qui définissent 3 niveaux d'erreurs :

org.xml.sax

Interface ErrorHandler

Method Summary

void	**error**(SAXParseException exception) Receive notification of a recoverable error.
void	**fatalError**(SAXParseException exception) Receive notification of a non-recoverable error.
void	**warning**(SAXParseException exception) Receive notification of a warning.

- **Erreur fatale (`fatalError`) :**

Il s'agit d'une ***erreur de syntaxe*** qui ne permet plus au parseur de continuer son travail (par exemple si le fichier n'est pas lisible)

- **Erreur (`error`) :**

Il s'agit d'une ***erreur de validation*** liée à un schéma ou une DTD, ce qui empêche le parseur de continuer l'analyse du document (par exemple si la spécification XML n'est pas respectée et le document n'est pas bien formé).

- **Avertissement (`warning`) :**

Il ne s'agit pas vraiment d'une erreur, mais de l'indication d'une incohérence comme par exemple la présence de définitions inutilisées dans une DTD (élément indépendant…)

Ces 3 méthodes **`fatalError`** , **`error`** et **`warning`** prennent en paramètre un objet de type **`SAXParseException`** qui contient les informations sur la nature et la localisation des erreurs. Le gestionnaire d'erreur, nous permet ainsi de ***récupérer les erreurs de parsing et de les traiter de façon personnalisée***.

Travail à faire : ***Etape n°7 : Mise en place d'un gestionnaire d'erreurs***

Etape n°7.1 : Personnalisation des messages d'erreurs.
Pour personnaliser les messages d'erreurs, il faut créer une nouvelle classe **`MonErrorHandler.java`**
Cette classe doit implémenter l'interface **`ErrorHandler`** et redéfinir les 3 méthodes : **`fatalError, error`** et **`warning`**.
Vous pouvez récupérer la classe **`MonErrorHandler`** sur la zone libre et l'importer à votre projet.

Etape n°7.2 : Enregistrement du gestionnaire d'erreur auprès du parser.
Pour enregistrer le gestionnaire d'erreurs auprès du parseur, il est nécessaire d'utiliser la méthode **`setErrorHandler.`**
Dans votre fichier **`TestSAX.java`**, juste en dessous de l'enregistrement du gestionnaire de contenu auprès du parseur, rajouter l'enregistrement du gestionnaire d'erreur :

```
saxParser.setErrorHandler(new MonErrorHandler());
```

Compiler et exécuter. Pour voir l'effet de votre gestionnaire d'erreur, falsifier votre document test XML, par exemple en enlevant la balise fermante `<\livre>` ou en rajoutant `aaa` en début de fichier…

Pour information, il existe deux autres gestionnaires que nous n'implémenterons pas dans ce tutoriel :

→ Le **`DTDHandler`** qui est un gestionnaire qui permet de récupérer les déclarations d'entités dans la DTD interne du document XML. Ce gestionnaire, liée à l'analyse DTD par le parseur, reste peu utilisé.

→ L' **`EntityResolver`** qui permet de créer des `InputSource` personnalisées pour des entités externes. La méthode `resolveEntity` qui retourne un nouvel `InputSource` sert à indiquer au parseur comment trouver certaines ressources externnes (lorsque par exemple on souhaite travailler localement avec un schéma ou une DTD normalement accessible sur Internet).

➢ 2. 8 : Présentation de la classe `DefaultHandler`

La classe **`DefaultHandler`** du paquetage **`org.xml.sax.helpers`** propose une implémentation par défaut des quatre gestionnaires **`SAX`** : `ContentHandler`, `DTDHandler`, `EntityResolver` et `ErrorHandler`.

org.xml.sax.helpers

Class DefaultHandler

java.lang.Object
└ **org.xml.sax.helpers.DefaultHandler**

All Implemented Interfaces:
ContentHandler, DTDHandler, EntityResolver, ErrorHandler

Toutes les méthodes implémentées par cette classe sont vides un peu sur le même principe que les classes **`Adapter`** de **`AWT`** (exemple : interface **`MouseListener`** /classe abstraite **`MouseAdapter`**)
La classe **`DefaultHandler`** permet ainsi de ***simplifier l'implémentation d'un parserSAX:***

- d'une part le gestionnaire de contenu ne nécessitera plus, comme précédemment, de redéfinir les 11 méthodes de l'interface **`ContentHandler`**…
- d'autre part avec la classe **`DefaultHandler`** toutes les méthodes de gestionnaires sont regroupées dans la même classe : un parserSAX peut donc être écrit en 1 seule classe gestionnaire héritant de **`DefaultHandler`**.

Attention cependant, même si la classe **`DefaultHandler`** permet, une fois héritée, de n'écrire qu'un ***seul gestionnaire***, il faut quand même ***appliquer au parser autant de méthodes setXXXHandler que de gestionnaires XXX*** que l'on souhaite enregistrer.
Ainsi si la classe gestionnaire écrite redéfinit certaines méthodes de `ContentHandler`, et certaines méthodes d'`ErrorHandler`, pour que le gestionnaire de contenu et le gestionnaire d'erreur soit effectif lors du parsing, il faudra effectuer auprès du parseur un **`setContentHandler`** ***et*** un **`setErrorHandler`**.
La classe gestionnaire ne redéfinira que les méthodes utiles !!!

➢ 2. 9 : Exercice : Application de la classe `DefautHandler` au fichier XML suivant :

`TestSAX3.xml`

```
<?xml version="1.0" encoding="UTF-8"?>

<catalogue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  Xsi:noNamespaceSchemaLocation="catalogue.xsd">
    <livre>
        <titre> XML </titre>
        <auteur>Alexandre BRILLANT </auteur>
        <edition>Eyrolles </edition>
        <ISBN>92-212-12151-3 </ISBN>
    </livre>
    <livre>
        <titre>JAVA/XML </titre>
        <auteur>Renaud FLEURY </auteur>
        <edition>Cahier du Programmeur </edition>
        <ISBN>2-212-11316-1</ISBN>
    </livre>
```

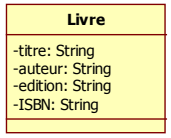
Nous souhaitons écrire un parserSAX (à partir de Xerces), qui transforme le flux XML précédent en une collection de **livres** et affiche cette collection en fin de parsing.
Trois classes sont donc nécessaires pour mettre en place ce parsing :

- une classe **Livre** qui modélisera l'objet métier Java correspondant à l'élément **livre** du fichier **xml.**
- une classe **CatalogueHandle**r qui jouera le rôle de "gestionnaire" en traitant correctement les événements SAX afin de transformer le flux XML en une **Collection <Livre>.**
- une classe **TestParsingSAXCatalogue** qui permettra de lancer sur le fichier test3SAX.xml le parsing (via le processeur XML Xerces)

***Travail à faire :* Parsing d'un catalogue : collection de livres …**

↳ ***Etape n°1 : Classe Livre.***
Ecrire dans votre projet, la classe **Livre**.
Cette classe est composée de :

- 4 attributs privés de type **String** : titre, auteur, edition, ISBN
- un constructeur par déafut et un constructeur à 4 paramètres
- des getteurs et setteurs correspondants, ainsi que de la méthode toString



↳ ***Etape n°2 : Gestion des événements dans* CatalogueHandler.**
Importer dans votre projet, la classe **CatalogueHandler.**
Consulter le code de cette classe. Cette classe hérite de **DefaultHandler**.
Elle a comme attribut un **catalogue** qui est une **Collection<Livre>.**
Elle instancie et complète cette collection au fur et à mesure du parsing de la manière suivante :

→ Balise <**catalogue**>: instanciation d'une **ArrayList<Livre>**
→ Balise <**livre**>: instanciation d'un nouveau **Livre**
→ Balise <**titre**>: on se prépare à mémoriser la valeur de cet élément dans une chaîne
→ Balise <**auteur**>: on se prépare à mémoriser la valeur de cet élément dans une chaîne
→ Balise <**edition**>: on se prépare à mémoriser la valeur de cet élément dans une chaîne
→ Balise <**ISBN**>: on se prépare à mémoriser la valeur de cet élément dans une chaîne
→ Balise </**titre**>: mise à jour de l'attribut titre(**setTitre**) du livre en cours d'analyse
→ Balise </**auteur**>: mise à jour de l'attribut auteur(**setAuteur**) du livre en cours d'analyse
→ Balise </**edition**>: mise à jour de l'attribut edition(**setEdition**) du livre en cours d'analyse
→ Balise </**ISBN**>: mise à jour de l'attribut ISBN(**setISBN**) du livre en cours d'analyse
→ Balise </**livre**>: ajout du livre à la collection de livres (**catalogue**)
A la fin du document, le **catalogue** est affiché.

Le gestionnaire de contenu a déjà été implémenté (méthodes startElement, endElement, characters, startDocument , endDocument). Consulter et comprendre le code écrit.

Rajoutez dans cette même classe **CatalogueHandler** la redéfinition des méthodes error, fatalError et warning permettant d'implémenter **le gestionnaire d'erreur**. Vous pouvez reprendre pour ces méthodes le code déjà écrit précédemment dans le tutoriel.

↳ ***Etape n°3 : Lancement du parsing SAX :* TestParsingSAXCatalogue**
Ecire la classe **TestParsingSAXCatalogue** qui après instanciation du parser et du gestionnaire, effectuera l'enregistrement du gestionnaire de contenu et du gestionnaire d'erreur auprès du parser instancié afin de pouvoir enfin lancer le parsing sur le fichier testSAX3.xml.
Exécuter **TestParsingSAXCatalogue** afin d'obtenir une copie d'écran similaire à la copie ci-contre.

```
Debut du parsing
-----------------------------
Fin du parsing
-----------------------------
Resultats du parsing
-----------------------------
Titre :  XML
 Auteur : Alexandre BRILLANT
 Edition : Eyrolles
 ISBN : 92-212-12151-3
-----------------------------
Titre : JAVA/XML
 Auteur : Renaud FLEURY
 Edition : Cahier du Programmeur
 ISBN : 2-212-11316-1
-----------------------------
```


Tutoriel n° 2 : Découverte de l'API JAXP

(Java API for XML Processing : javax.xml.parsers)

Pour instancier un parser **SAX**, vous trouverez ci-dessous un récapitulatif du code du fichier **TestParsingSAXCatalogue**

```
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;


public class TestParsingSAXCatalogue {

   public static void main(String[] args) {

      try {
      // Instanciation d'un parseurSAX à l'aide d'une factory
      XMLReader saxParser =
XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");

      // Instanciation du gestionnaire
      CatalogueHandler monGestionnaireSAX = new CatalogueHandler();
      //Enregistrement du gestionnaire de contenu...
      saxParser.setContentHandler(monGestionnaireSAX);
      //Enregistrement du gestionnaire d'erreur ...
      saxParser.setErrorHandler(monGestionnaireSAX);

      saxParser.parse("testSAX3.xml");

   } catch (SAXException e) {
      System.out.println("Erreur d'analyse: "+ e.getMessage());
   } catch (IOException e) {
      System.out.println("Erreur I/O: "+ e.getMessage());
   }
   System.out.println("-- Fin TestParsingSAXCatalogue !!! ");
   }// fin main
}// fin classe TestParsingSAXCatalogue
```

Pour instancier un parser en utilisant **JAXP**, les **import** (référence au paquetage **javax.xml.parsers**) et les instructions à l'intérieur du **try** deviennent :

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import javax.xml.parsers.ParserConfigurationException;


public class TestParsingJAXPCatalogue {

   public static void main(String[] args) {

      try {
          // Instanciation du parser à l'aide d'une SAXParserfactory
          SAXParserFactory factory = SAXParserFactory.newInstance();
          SAXParser  parser = factory.newSAXParser();

          // Instanciation du gestionnaire
          CatalogueHandler monGestionnaireSAX = new CatalogueHandler();
          parser.parse("testSAX3.xml", monGestionnaireSAX);
          }
      //… suivis des catch nécessaires …


   System.out.println("-- Fin TestParsingSAXCatalogue !!! ");
   }// fin main
}// fin classe TestParsingJAXPCatalogue
```

JAXP(Java API for XML Processing) est un ensemble de classes Java du paquetage **javax.xml.parsers** (intégrés au **JDK** depuis la version 1.4) qui permet manipuler des fichiers XML sans se préoccuper dans le code du parseur XML utilisé (Xerces, ou autre…)
En effet, l'objectif de **JAXP** est de fournir une ***couche d'abstraction permettant d'utiliser n'importe quel parseur SAX*** (mais aussi DOM **D**ocument **O**bject **M**odel). Ainsi **JAXP** ne fournit pas une nouvelle méthode pour parser un document XML **mais propose une interface commune pour appeler et paramétrer un parseur de façon indépendante**.

Package javax.xml.parsers

Provides classes allowing the processing of XML documents.

See:

Description

Class Summary	
DocumentBuilder	Defines the API to obtain DOM Document instances from an XML document.
DocumentBuilderFactory	Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.
SAXParser	Defines the API that wraps an XMLReader implementation class.
SAXParserFactory	Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents.

Exception Summary	
ParserConfigurationException	Indicates a serious configuration error.

En utilisant un code qui respecte JAXP, il est possible d'utiliser n'importe quel parseur qui répond à cette API .

Par exemple jusqu'à maintenant dans ce tutoriel pour instancier un parser SAX de "manière classique" (via le paquetage org.xml.sax), nous avons vu qu'il existait deux méthodes nécessitant une *recompilation* d'une partie du code lors du changement du parseur

- soit une instanciation « en dur » du parser *SAX* (par appel direct au constructeur)

```
XMLReader saxParser = new org.apache.xerces.parsers.SAXParser();
```

- soit une instanciation via une factory

```
XMLReader saxParser =
XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
```

JAXP propose de fournir le ***nom de la classe du parseur en paramètre à la JVM*** sous la forme **d'une propriété système**. Il n'est ainsi plus nécessaire de procéder à une recompilation mais simplement ***de mettre jour cette propriété*** et le **CLASSPATH** pour qu'il référence les classes du nouveau parseur. On peut alors écrire du code complètement indépendant du parseur utilisé.

➢ L'API JAXP fournit entre autre la classe abstraite **javax.xml.parsers.SAXParserFactory** qui définit une « usine » qui permet de configurer et d'obtenir un parseur SAX.

static SAXParserFactory	**newInstance**() Obtain a new instance of a SAXParserFactory.

↳ La méthode **newInstance** **permet de créer une instance d'une** usine de type **SAXParserFactory**
Remarque : La propriété système **javax.xml.parsers.SAXParserFActory** permet de préciser la classe fille qui hérite de la classe **SAXParserFactory** et qui sera instanciée (dans notre cas, le parseur Xerces est déjà dans le classpath !)

abstract SAXParser	**newSAXParser**() Creates a new instance of a SAXParser using the currently configured factory parameters.

↳ La méthode **newSAXParser** de la classe **SAXParserFactory** permet d'obtenir une **instance du parseur de type SAXParser** en utilisant les paramètres courants. Cette méthode peut lever une exception de type **ParserConfigurationException**.
Remarque : il est possible de fournir quelques paramètres à la Factory pour lui permettre de configurer le parseur (par exemple, setNamespaceAware pour spécifier que le parseur fournit un support pour l'utilisation de namespaces, méthode setValidating pour spécifier que le parseur doit valider le document XML… voir javadoc…)

➢ L'API JAXP fournit également la classe abstraite **javax.xml.parsers.SAXParser** qui propose plusieurs surcharges pour la méthode parse dont les surchrages suivantes :

void	**parse**(InputStream is, HandlerBase hb, String systemId) Parse the content of the given InputStream instance as XML using the specified HandlerBase.
void	**parse**(String uri, DefaultHandler dh) Parse the content described by the giving Uniform Resource Identifier (URI) as XML using the specified DefaultHandler.
void	**parse**(String uri, HandlerBase hb) Parse the content described by the giving Uniform Resource Identifier (URI) as XML using the specified HandlerBase.

➢ Les explications précédentes permettent de justificer les instructions écrites dans le fichier **TestParsingJAXPCatalogue.java** afin d'illustrer un parsing utilisant JAXP

```
// Instanciation du parser à l'aide d'une SAXParserfactory
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser  parser = factory.newSAXParser();

 // Instanciation du gestionnaire
CatalogueHandler monGestionnaireSAX = new CatalogueHandler();
parser.parse("testSAX3.xml", monGestionnaireSAX);
}
```

Travail à faire : ***Test du parsing avec*** **JAXP**

Importez le fichier **TestParsingJAXPCatalogue.java** disponible sur la zone libre et exécutez-le

➢ du **SAXParser** vers le **XMLReader** …
La javdoc de l'interface **org.xml.sax.XMLReader** nous indique que **XMLReader** est l'interface qui doit être implémentée par tout parser **SAX**.

> XMLReader is the interface that an XML parser's SAX2 driver must implement. This interface allows an application to set and query features and properties in the parser, to register event handlers for document processing, and to initiate a document parse.

… Ce qui est bien confirmé dans la javadoc de **JAXP** qui indique que la classe **java.xml.parsers.SAXParsers** nous indique est la classe qui encapsule l'implémentation de la classe **XMLReader**

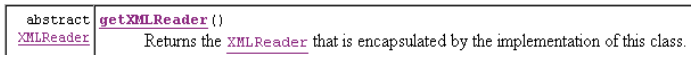
javax.xml.parsers
Class SAXParser

java.lang.Object
  └ **javax.xml.parsers.SAXParser**

public abstract class **SAXParser**
extends Object

Defines the API that wraps an XMLReader implementation class. In JAXP 1.0, this class wrapped the Parser interface, however this interface was replaced by the XMLReader. For ease of transition, this class continues to support the same name and interface as well as supporting new methods. An instance of this class can be obtained from the SAXParserFactory.newSAXParser() method. Once an instance of this class is obtained, XML can be parsed from a variety of input sources. These input sources are InputStreams, Files, URLs, and SAX InputSources.

Ainsi pour faire apparaître un un **`XMLReader`** dans notre code, il suffit d'appliquer sur le **`SAXParser`**, la méthode **`getXMLReader`**.

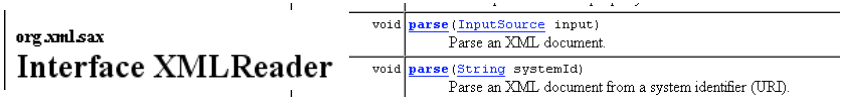


Travail à faire : *Modifier le code du fichier* **`TestParsingJAXPCatalogue.java`** *pour faire apparaître un* **`XMLReader.`**

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.XMLReader;


public class TestParsingJAXPCatalogue {

    public static void main(String[] args) {

        try {
            // Instanciation du parser à l'aide d'une SAXParserfactory
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser  parser = factory.newSAXParser();

            // Obtention du XMLReader
            XMLReader  xmlReader = parser.getXMLReader();

             // Instanciation du gestionnaire
            CatalogueHandler monGestionnaireSAX = new CatalogueHandler();
            xmlReader.parse("testSAX3.xml", monGestionnaireSAX);
            }
        //… suivis des catch nécessaires …


    System.out.println("-- Fin TestParsingSAXCatalogue !!! ");
    }// fin main
}// fin classe TestParsingJAXPCatalogue
```

Effectuez les modifications précédentes. Compilez.Que constatez-vous ?

L'erreur de compilation au niveau de la méthode **`parse`** s'explique pat le fait que l'interface **`XMLReader`** ne propose que deux surcharges pour la méthode **`parse`**…et celle-ci n'existe pas !
Ainsi Si vous utilisez un **`XMLReader`** (au dessus du **`SAXParser`**), vous devrez donc d'abord enregistrer le(s) gestionnaire(s) (`XXXHandler`) puis utiliser une des deux méthodes **`parse`** proposées par l'interface **`org.xml.sax.XMLReader.`**



Le code complet pour utiliser **`JAXP`** et un **`XMLReader`** est donc le suivant :

Travail à faire : *Modifier le code du fichier* **`TestParsingJAXPCatalogue.java`** *pour faire apparaître un* **`XMLReader`** *et tester !*

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.XMLReader;


public class TestParsingJAXPCatalogue {

    public static void main(String[] args) {

        try {
            // Instanciation du parser à l'aide d'une SAXParserfactory
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser  parser = factory.newSAXParser();

            // Obtention du XMLReader
            XMLReader  xmlReader = parser.getXMLReader();

             // Instanciation du gestionnaire
            CatalogueHandler monGestionnaireSAX = new CatalogueHandler();

            //Enregistrement des gestionnaires de contenu et d'erreur
            xmlReader.setContentHandler(monGestionnaireSAX);
            xmlReader.setErrorHandler(monGestionnaireSAX);
            //Lancement du parsing
            xmlReader.parse("testSAX3.xml");
            }
        //… suivis des catch nécessaires …


    System.out.println("-- Fin TestParsingSAXCatalogue !!! ");
    }// fin main
}// fin classe TestParsingJAXPCatalogue
```

Récapitulatif du principe de fonctionnement avec* `JAXP` *…à retenir !
L'application commence par récupérer un parseur (**`javax.xml.parsers.SAXParser`**) à partir d'une fabrique de parseurs (**`javax.xml.parsers.SAXParserFactory`**).
Ce parseur parcourt le document **`XML`** grâce à un lecteur (**`org.xml.sax.XMLReader`**) qui contient plusieurs gestionnaires (ou **`handler`**s).

A retenir !!!

SAX (**S**imple **A**PI for **X**ML) est une API **événementielle et incrémentale** qui permet de réaliser un parsing "à la volée" d'un document **XML**.
SAX génére donc différents types d'**événements** lors de la lecture séquentielle d'un document **XML**. L'utilisateur peut alors recevoir ces événements et adopter le comportement approprié grâce à un gestionnaire de contenu qu'il aura pris soin d'enregsitrer auprès du parser : un gestionnaire de contenu est une implémentation de l'interface **`org.xml.sax.ContentHandler`**. Il est également possible d'associer d'autres gestionnaires au parser comme le gestionnaire d'erreurs par exemple.

Attention, **SAX** ne fournit que des définitions. L'implémentation est laissée aux différents éditeurs qui fournissent un parseur compatible avec **SAX** (comme le processeur **`Xerces`** par exemple)

Avantages de SAX
→ c'est simple : **S**imple **A**PI for **X**ML
→ une seule lecture du document (⇒ indépendante de la taille du document initial donc peu coûteux en mémoire)
→ pratique lorsqu'on a besoin de travailler uniquement sur un petit sous-ensemble d'un document

Inconvénients de SAX
→ modification impossible du document XML (acces read-only)
→ accès direct à un endroit précis du document impossible (no random-access)

Les 2 **`API`** sur lesquelles on a travaillé dans ce tutoriel :
→ **`org.xml.sax`** qui est l'API de base pour manipuler les flux XML (**SAX** : **S**imple **A**PI for XML)
→ **`javax.xml.parsers`** qui est également appelée l'API **JAXP** (**J**ava **A**PI for **X**ML **P**rocessing) a pour but de fournir une interface commune à tous les parsers **SAX** (et DOM)

Annexe 1 : Interface `ContentHandler` du paquetage `org.xml.sax`

org.xml.sax
Interface ContentHandler

All Known Subinterfaces:
TemplatesHandler, TransformerHandler, UnmarshallerHandler

All Known Implementing Classes:
DefaultHandler, DefaultHandler2, ValidatorHandler, XMLFilterImpl, XMLReaderAdapter

Method Summary

void	**characters**(char[] ch, int start, int length) Receive notification of character data.
void	**endDocument**() Receive notification of the end of a document.
void	**endElement**(String uri, String localName, String qName) Receive notification of the end of an element.
void	**endPrefixMapping**(String prefix) End the scope of a prefix-URI mapping.
void	**ignorableWhitespace**(char[] ch, int start, int length) Receive notification of ignorable whitespace in element content.
void	**processingInstruction**(String target, String data) Receive notification of a processing instruction.
void	**setDocumentLocator**(Locator locator) Receive an object for locating the origin of SAX document events.
void	**skippedEntity**(String name) Receive notification of a skipped entity.
void	**startDocument**() Receive notification of the beginning of a document.
void	**startElement**(String uri, String localName, String qName, Attributes atts) Receive notification of the beginning of an element.
void	**startPrefixMapping**(String prefix, String uri) Begin the scope of a prefix-URI Namespace mapping.

Explication détaillée des méthodes de l'interface ContentHandler
(extrait de http://smeric.developpez.com/java/cours/xml/sax/)

`setDocumentLocator`: Un locator vous permet de localiser "le curseur" pendant le traitement du flux vous permettant par exemple de connaître le numéro de ligne et de colonne en cours d'analyse. Ceci est une fonctionnalité certes très intéressante au moment du débugage mais qu'il faut à tout prix éviter, que dis-je, vous interdire d'utiliser pour le traitement proprement dit. Dans les Helper de l'API sax, il vous est fournit une implémentation par défaut de toutes les interfaces. Si l'implémentation par défaut du ContentHandler est proprement inutile, celle du Locator devrait amplement suffire à 99% des développements, je ne m'étendrai donc pas sur ce point.

`startDocument`: Cette méthode est appelée par le parser une et une seule fois au démarrage de l'analyse de votre flux xml. Elle est appelée avant toutes les autres méthodes de l'interface, à l'exception unique, évidemment, de la méthode setDocumentLocator. Cet événement devrait vous permettre d'initialiser tout ce qui doit l'être avant le début du parcours du document.

`endDocument:` Et son contraire, cette méthode est donc appelée à la fin du parcours du flux après toutes les autres méthodes. Il peut alors être utile à ce moment de notifier d'autres objets du fait que le travail est terminé.

`ProcessingInstruction:` Cet événement est levé pour chaque instruction de fonctionnement rencontrée. Ces instructions sont celles que vous trouvez hors de l'arbre xml lui-même comme par exemple les instructions concernant les dtd ou plus simplement la déclaration :
`<?xml version="1.0" encoding="ISO-8859-1" ?>`

`startPrefixMapping:` Cet événement est lancé à chaque fois qu'un mapping préfixé, c'est à dire une balise située dans un espace de nommage (name space) , est rencontré.

`endPrefixMapping:` Son événement contraire évidemment, c'est à dire la fin du traitement dans un espace de nommage.

`startElement:` Démarrage d'un élément XML... Enfin ! Et oui, on peut en effet pour démarrer avec SAX se contenter de comprendre cet événement et son traitement ainsi que son contraire pour analyser un flux xml de manière très puissante et efficace. Nous allons donc nous pencher un peu plus sur cet événement.

```
startElement (String namespaceUri, String localName, String rawName, Attributs atts);
```

* où `nameSpaceUri` est la chaîne de caractères contenant l'URI complète de l'espace de nommage du tag ou une chaîne vide si le tag n'est pas compris dans un espace de nommage,
* `localName` est le nom du tag sans le préfixe s'il y en avait un,
* `rawName` est le nom du tag version xml 1.0 c'est à dire $prefix:$localname,
* Enfin attributs est la liste des attributs du tag que l'on étudiera un peu plus loin.

`endElement:` Evénement inverse de signature beaucoup plus simple puisque seul le nom complet du tag a besoin d'être connu. En effet, à la fermeture de la balise XML, aucun attribut n'est requis.

`characters:` Tout ce qui est dans l'arborescence mais n'est pas partie intégrante d'un tag, déclenche la levée de cet événement. En général, cet événement est donc levé tout simplement par la présence de texte entre la balise d'ouverture et la balise de fermeture comme dans l'exemple suivant:

```
<maBalise>un peu de texte</maBalise>
```

La présence de "un peu de texte" provoque la levée de l'événement characters. Attention : il est à noter que l'API SAX n'impose rien quant à pas l'implémentation de cet événement. Dans le cas d'un texte épars autour de balises filles de la balise en cours, les réactions peuvent être diverses. Ainsi le flux xml suivant :

```
<maBalise>un peu
    <baliseImbriquee nom="coucou"/> de texte<baliseImbriquee nom="toto"/>éparpillé
</maBalise>
```

Il peut soit donner lieu à trois événements contenant respectivement le texte "un peu", " de texte", "éparpillé" soit donner un seul événement contenant l'intégralité du texte à savoir "un peu de texte éparpillé". Comme l'API ne fixe rien, ce sera à vous de penser au fait que le parser que vous avez sous la main ne sera peut être pas celui de vos clients et d'agir en conséquence, c'est à dire en gérant les deux types de réactions possibles de telle sorte qu'elles fournissent le même comportement final dans les deux cas.

`ignorableWhiteSpace:` Permet de traiter les espaces et tabulations multiples, sachant qu'ils n'ont normalement aucune valeur en xml. Un ou deux ou 10 espaces, 1 espace et une tabulation et 3 retours chariot, etc sont autant d'espaces normalement ignorés en XML. Cet événement est donc levé à chaque fois que des espaces normalement ignorés sont rencontrés. En fait les paramètres de la méthode contiennent la chaîne complète de characters et les index de début et de fin de la série d'espaces ignorables. A vous de voir si vous voulez outrepasser la préconisation qui considère ces espaces comme étant inutiles.

`skippedEntity :` Evitez d'y toucher, cette méthode est levée à chaque fois qu'une entité (une balise et toute l'arborescence descendante) est ignorée. Elle le sera si vous avez demandé au parser de ne pas valider le document et que la balise en question est mal formée. Bref, vous faites face à une situation dangereuse pour votre application, soit vous décidez alors de partir sur des valeurs par défaut, soit, et c'est en général le mieux, vous interrompez le traitement pour défaut dans l'environnement.

Annexe 2 : Interface `Attributes` du paquetage `org.xml.sax`

org.xml.sax

Interface Attributes

All Known Subinterfaces:
Attributes2

All Known Implementing Classes:
Attributes2Impl, AttributesImpl

Method Summary

int	**getIndex**(String qName) Look up the index of an attribute by XML qualified (prefixed) name.
int	**getIndex**(String uri, String localName) Look up the index of an attribute by Namespace name.
int	**getLength**() Return the number of attributes in the list.
String	**getLocalName**(int index) Look up an attribute's local name by index.
String	**getQName**(int index) Look up an attribute's XML qualified (prefixed) name by index.
String	**getType**(int index) Look up an attribute's type by index.
String	**getType**(String qName) Look up an attribute's type by XML qualified (prefixed) name.
String	**getType**(String uri, String localName) Look up an attribute's type by Namespace name.
String	**getURI**(int index) Look up an attribute's Namespace URI by index.
String	**getValue**(int index) Look up an attribute's value by index.
String	**getValue**(String qName) Look up an attribute's value by XML qualified (prefixed) name.
String	**getValue**(String uri, String localName) Look up an attribute's value by Namespace name.

Remarque : une ***URI*** **(Uniform Ressource Identifiers)** peut-être :

- une URL : **Uniform Ressource Locator:**
  `http://www.mon.site/unFichier.dtd`
- une URN : **Uniform Ressource Name:**
  `http://www.mon.site/unFichier.xml#xpointer(book1)`