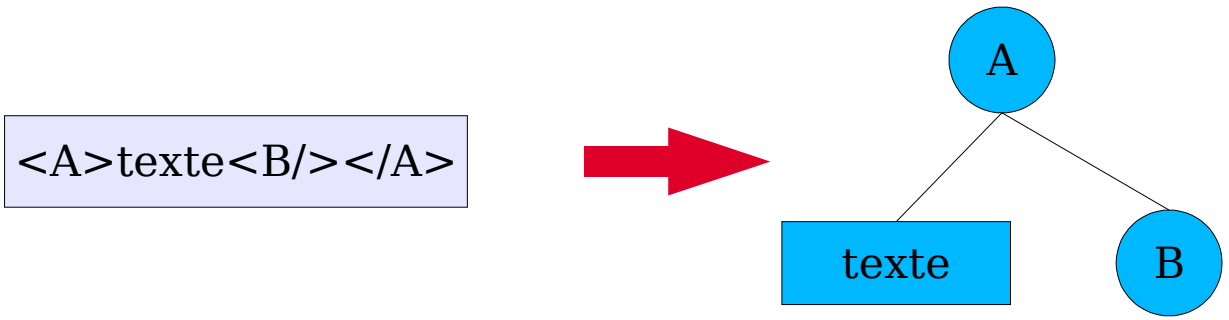
Écrire une application XML ?

Écrire une application XML ?

Simple API for XML : SAX

SAX ?

- C'est une API de niveau parseur
- Les documents sont modélisés comme un flux de d'événements
  - Les événements sont la découverte par le parseur des balises, des PCDATA, ...
- Le document est analysé par SAX du début jusqu'à la fin, élément par élément, attribut par attribut
- A partir de cette analyse, on peut fabriquer une structure de données de type arbre



Pourquoi SAX ?

- Permet de réaliser de petites applications XML, légères et adaptées à vos besoins
- Permet de faire des traitements en ligne, l'intégralité du document n'est pas chargée en mémoire
- Toutes les outils pour faire l'analyse d'un document XML sont fournis
- Traitement des erreurs avec une certaine souplesse
  - Permet de fabriquer des messages d'erreur spécifiques à votre application

Quand ne faut-il pas choisir SAX ?

- Aucun moyen d'accéder aléatoirement à des données XML
  - Le flux d'événement est strictement séquentiel
  - Impossible de revenir en arrière dans l'analyse
- Les applications utilisent SAX pour écrire un document avec une structure qui sera ensuite utilisé avec XPath

Écrire une application XML ?

Utilisation de SAX

Quels sont les parseurs SAX 2 ?

- Crimson
  - Parser suportant XML 1.0, disponible via l'API basée sur le Project X de Sun : JAXP 1.1 (Java API for XML Processing)
  - Ce parser n'est maintenant plus utilisé par les dernières versions de JAXP
- Xerces
  - Famille de parseurs XML de l'Apache XML Project
  - Disponible à http://xml.apache.org/xerces
- Depuis J2SE 1.4 JAXP 1.1 est livré en standard
- Le parseur utilisé était Crimson jusque J2SE 1.4.1. Depuis J2SE 1.4.2, c'est Xerces qui est maintenant le parseur XML de référence

Producteurs et consommateurs

- L'API SAX définit 2 rôles différents : le producteur et le consommateur
- Le rôle de producteur d'événements est tenu par un parseur XML et est représenté par une instance d'une classe
  - Le producteur est chargé de transmettre des événements d'analyse aux objets qui tiennent le second rôle : celui de consommateur d'événements
- Les consommateurs font le « vrai » travail
  - Ils interprètent les événements de l'analyse et les utilisent pour créer des structures de données spécialisées
  - Si aucun consommateur existe, rien ne se produit

Première application

- Nous allons partir du squelette d'une application SAX (fichier Exemple1.java)
- Explication du code :
  - XMLReader producer;
    - Le type le plus fréquent de producteur d'événements SAX2 qui est un parseur XML
    - Il va produire un flux d'événements lors de la rencontre d'éléments dans le document XML
  - producer = XMLReaderFactory.createXMLReader();
    - Création du parseur XML
  - consumer = new DefaultHandler();
    - Implémente la plupart des interfaces de consommations d'événements : que fait quand on trouve une balise ouvrante, fermante, un attribut, ... ?
    - Si aucun méthode n'est redéfinie, par défaut, elle ne fait rien
  - producer.setContentHandler(consumer);
    - ContentHandler permet d'annoncer aux consommateurs les principaux événements qui sont : les éléments, les attributs et le texte
  - producer.setErrorHandler(consumer);
    - Permet de gérer les erreurs
  - producer.parse (argv[0]);
    - Indique au parseur de lire le texte XML donnée en paramètre (URI absolue).

Gestion du PCDATA

- A partir de la classe DefaultHandler, nous allons redéfinir la méthode characters qui va permet de gérer les PCDATA
- Le consommateur d'événements va donc maintenant pour réagir en fonction des événements fournis par le producteur
- Redéfinition de la méthode characters de la façon suivante :

```
public void characters(char buf[], int offset, int length) throws SAXException
        {  System.out.print(new String(buf,offset,length));   }
```

- *Code source* : Exemple2.java
- Compilez et exécutez Exemple2 sur le fichier XML-Cours6-Exemple1.xml. Ne pas oublier que le paramètre doit être une URI absolue

Gestion des éléments

```
public void startElement(String namespaceURI, String localName,
          String qName, Attributes atts)
          throws SAXException
public void endElement(String namespaceURI, String localName,
          String qName)
          throws SAXException
```

- Utilisation dans Exemple3.java
- Valeur d'un attribut : atts.getValue("*nom_de_l'attribut*")
- Exercice : A partir du source Exemple4.java, générez un document HTML à partir du fichier XML-Cours6-Exemple1.xml

Parseur validant

- SAX2 permet de définir des propriétés sur le parseur via la méthode setFeature
- Forcer le parseur à valider un document XML :

```
String uri="http://xml.org/sax/features/validation";
producer.setFeature(uri,true);
System.out.println("Validation ? : " +producer.getFeature(uri));
```

- Les erreurs du parseur sont interceptées par SAXParseException mais par défaut, les erreurs ne sont pas indiquées
- Nous devons rédéfinir la méthode error(SAXParseException e) comme dans l'exemple Exemple5.java
- Exercice : A l'aide de Exemple5, rédigez correctement le fichier XML-Cours6-Exemple1.dtd pour que le fichier XML-Cours6-Exo2.xml soit valide
- Autres exercices : grâce à Exemple5, vous pouvez vérifier les solutions du Cours5-DTD