Modalités

Vous avez l'énoncé numéro 1. Vous commencerez par identifier votre énoncé :

– NOM :

– PRÉNOM :

Vous devrez **impérativement** rendre **l'énoncé complet**.

1 Documents bien formés et documents valides

On rappelle qu'un document est dit **bien formé** s'il est syntaxiquement correct, sans référence à une quelconque DTD. Un document **valide** est bien formé et respecte les contraintes indiquées dans la DTD qui lui correspond (la DTD est obligatoire).

Exercice 1 :

Pour chacun des documents suivants, indiquez s'il est bien formé ou pas. Quand le document n'est pas bien formé indiquez la nature de l'erreur (ou des erreurs). Les documents sont complètement indépendants les uns des autres.

1.

```
<?xml version="1.0" ?>
<top>
<item>Question 1<item answer="a">
<item>Question 2<item answer="b">
<item>Question 3<item answer="c">
</top>
```

2.

```
<?xml version="1.0" ?>
<text>
<font size='8pt'>petite police</font>
<font size='24pt'>grande police</font>
</text>
```

3.

```
<?xml version="1.0" ?>
<top>
<item val=2/>
<item val=3/>
<item val=12/>
</top>
```

4.

```
<?xml version="1.0" ?>
<text>
<font small>Un petit texte</font>
<font big>Un grand texte</font>
</text>
```

5.

```
<a>
<b a="toto">Et hop</b>
</a>
<a>
<b a="titi">Voilà</b>
</a>
```

6.

```
<programme titre="Internet">
<ul>
<li>XML<li/>
<li>DTD<li/>
<li>API<li/>
<li>XSL<li/>
</ul>
</programme>
```

7.

```
<?xml version="1.0"?>
<a>
<b><c/></b><d></b>
</a>
```

8.

```
<?xml version="1.0"?>
<!DOCTYPE text [
<!ELEMENT text EMPTY>
]>
<text><bf>Titre</bf></text>
```

Exercice 2 :

Pour chacun des documents suivants, indiquez s'il est valide ou pas. Quand le document n'est pas valide indiquez la nature de l'erreur (ou des erreurs). Les documents sont complètement indépendants les uns des autres.

1.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE a [
  <!ELEMENT a (b*, c)>
  <!ELEMENT b EMPTY>
  <!ELEMENT c (#PCDATA)>
  <!ATTLIST c
            x CDATA #FIXED "bold">
]>
<a>
<b/>
<b/>
```

```
<c x="medium">du texte</c>
</a>
```

2.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE a [
  <!ELEMENT a (b*, c*, d?)>
  <!ELEMENT d EMPTY>
  <!ELEMENT b (#PCDATA)>
]>
<a>
<b>ljs sldjf sljd </b>
<b>mmmqmm qqq mmm qq </b>
<d/>
</a>
```

3.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE a [
  <!ELEMENT a (b*, c)>
  <!ELEMENT b EMPTY>
  <!ELEMENT c (#PCDATA)>
  <!ATTLIST b
            truc CDATA #IMPLIED>
]>
<a>
<b/>
<b truc="bidule"/>
<c>Et voilà</c>
<b/>
</a>
```

4.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE a [
  <!ELEMENT a (b*, c)>
  <!ELEMENT b EMPTY>
  <!ELEMENT c (#PCDATA)>
  <!ATTLIST c
            x CDATA #FIXED "bold">
  <!ATTLIST a
            y CDATA #IMPLIED>
]>
<a>
<b/>
<b/>
<c>aeaeae aaaa</c>
</a>
```

2 Ecriture et utilisation de DTD

Exercice 3 :

1. Ecrivez la DTD la plus restrictive possible compatible avec le document suivant :

```
<?xml version="1.0" ?>
<!DOCTYPE list SYSTEM "UneDTD.dtd">
<list>
<item><em f="bold">bli,</em> bli, bli</item>
<sep/>
<item>Bla, bla, bla</item>
</list>
```

2. La DTD proposée comme solution à la question précédente autorise des documents valides à la structure plus complexe que celle du document de base. Donnez un exemple d'un tel document.

Exercice 4 :

Ecrivez la DTD la plus restrictive possible compatible avec les documents suivants (si un élément peut apparaître plus d'une fois, on considérera qu'il peut apparaître un nombre arbitraire de fois) :

```
<?xml version="1.0" ?>
<!DOCTYPE text SYSTEM "UneAutreDTD.dtd">
<text>
<section><titre>Première</titre>
<p align="right">Du texte <em>et aussi</em> du texte.
</p>
<p align="center">Important, ce texte.</p>
</section>
</text>
```

```
<?xml version="1.0" ?>
<!DOCTYPE text SYSTEM "UneAutreDTD.dtd">
<text>
<auteur>Quelqu'un</auteur>
<section><titre>Début</titre>
<p>Encore du texte.
</p>
<p>Toujours du texte.
</p>
</section>
<section><titre>Suite</titre>
<p>Je n'ai aucune <em>inspiration</em>. Et <font name="garamond">vous</font> ?
</p>
</section>
</text>
```

3 Utilisation de l'API SAX

On rappelle que l'API SAX est basée sur un modèle évènementiel. A chaque fois que l'analyseur syntaxique (le *parseur*) rencontre une construction XML à laquelle il peut donner un sens, il produit un évènement et appelle la méthode correspondant à la nature de l'évènement de l'objet de gestion enregistré par l'application. Cet objet doit impérativement implanter l'interface `DocumentHandler`, qui comporte (entre autres) les méthodes suivantes :

`void startDocument()`
: Appelée une fois au début de l'analyse du document XML.

`void endDocument()`
: Appelée une fois à la fin de l'analyse du document XML.

`void startElement(String name,AttributeList atts)`
: Appelée pour chaque balise ouvrante rencontrée. `name` contient le nom de la balise, alors que `atts` contient la liste des attributs de la balise.

`void endElement(String name)`
: Appelée pour chaque balise fermante rencontrée. `name` contient le nom de la balise.

`void characters(char ch[],int start,int length)`
: Appelée pour chaque morceau de texte rencontré. Le texte à prendre en considération commence à la case `start` du tableau `ch` et comporte `length` caractères.

En général, on crée un gestionnaire de document en héritant de la classe `HandlerBase` qui implante toutes les méthodes de `DocumentHandler`. L'interface `AttributeList` définit entre autres trois méthodes importantes :

`int getLength()`
: Donne le nombre d'attributs dans la liste.

`String getName(int i)`
: Donne le nom du `i`-ème attribut de la liste (numérotation à partir de 0).

`String getValue(int i)`
: Donne la valeur du `i`-ème attribut de la liste (numérotation à partir de 0).

Exercice 5 :

On propose le gestionnaire de documents suivant :

```
1  import org.xml.sax.HandlerBase;
2  import org.xml.sax.AttributeList;
3  public class Gestionnaire1 extends HandlerBase {
4    private int n;
5    public void startDocument() {
6      n=0;
7    }
8    public void startElement (String name, AttributeList atts) {
9      System.out.println(n+": "+name);
10     n++;
11   }
12 }
```

On analyse le fichier suivant en utilisant le gestionnaire :

```
<a><b/><c/><d><c/></d></a>
```

Donnez l'affichage produit par l'analyseur. Que fait le gestionnaire (on attend une description de haut niveau) ?

Exercice 6 :

On propose le gestionnaire de documents suivant :

```
import org.xml.sax.HandlerBase;
import org.xml.sax.AttributeList;
public class Gestionnaire3 extends HandlerBase {
  public void startElement (String name, AttributeList atts) {
    for(int i=0;i<atts.getLength();i++) {
      if(atts.getName(i).equals("react")
         && atts.getValue(i).equals("don't")){
        return;
      }
    }
    System.out.println(name);
  }
}
```

On analyse le fichier suivant en utilisant le gestionnaire :

```
<uh>
<a react="don't"><d/></a>
<b react="do"><c/></b>
<d/>
</uh>
```

Donnez l'affichage produit par l'analyseur. Que fait le gestionnaire (on attend une description de haut niveau) ?

Exercice 7 :

On propose le gestionnaire de documents suivant :

```
import org.xml.sax.HandlerBase;
import org.xml.sax.AttributeList;
public class Gestionnaire6 extends HandlerBase {
  private boolean action;
  private int n;
  public void startDocument() {
    action=true;
  }
  public void startElement (String name, AttributeList atts) {
    for(int i=0;i<atts.getLength();i++) {
      if(atts.getName(i).equals("action")) {
        action=atts.getValue(i).equals("no");
        break;
```

```
      }
    }
  }
  public void characters(char ch[],int start,int length) {
    if(action) {
      String s=new String(ch,start,length); // chaîne de caractères
      // correspondant au morceau de texte à prendre en compte
      s=s.trim(); // suppresion des blancs en début et en fin de chaîne
      if(s.length()>0) {
        System.out.println(s);
        n++;
      }
    }
  }
  public void endDocument() {
    System.out.println("Total : "+n);
  }
}
```

On analyse le fichier suivant en utilisant le gestionnaire :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<text action="yes">
<p>On rappelle que l'API SAX est basée sur un modèle <bf>évènementiel</bf>.</p>
<p action="no">On utilise un analyseur syntaxique, un <em>parseur</em>.</p>
</text>
```

Donnez l'affichage produit par l'analyseur. Que fait le gestionnaire (on attend une description de haut niveau) ?

Exercice 8 :

Un document `XML` peut être représenté par un arbre. Ecrire un gestionnaire de document qui affiche la hauteur de l'arbre correspondant au document analysé. On rappelle qu'un arbre réduit à une racine est de hauteur zéro, et qu'on obtient la hauteur d'un arbre quelconque en ajoutant 1 à la hauteur du plus grand sous arbre attaché directement à la racine.

Exercice 9 :

On rappelle la `DTD PLAY`, utilisée pour décrire des pièces de théatre :

```
<!-- DTD for Shakespeare    J. Bosak    1994.03.01, 1997.01.02 -->
<!-- Revised for case sensitivity 1997.09.10 -->
<!-- Revised for XML 1.0 conformity 1998.01.27 (thanks to Eve Maler) -->

<!-- <!ENTITY amp "&#38;#38;"> -->
<!ELEMENT PLAY     (TITLE, FM, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
                                          PROLOGUE?, ACT+, EPILOGUE?)>
<!ELEMENT TITLE    (#PCDATA)>
<!ELEMENT FM       (P+)>
<!ELEMENT P        (#PCDATA)>
```

```
<!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
<!ELEMENT PGROUP   (PERSONA+, GRPDESCR)>
<!ELEMENT PERSONA  (#PCDATA)>
<!ELEMENT GRPDESCR (#PCDATA)>
<!ELEMENT SCNDESCR (#PCDATA)>
<!ELEMENT PLAYSUBT (#PCDATA)>
<!ELEMENT INDUCT   (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
<!ELEMENT ACT      (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
<!ELEMENT SCENE    (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
<!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
<!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
<!ELEMENT SPEECH   (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
<!ELEMENT SPEAKER  (#PCDATA)>
<!ELEMENT LINE     (#PCDATA | STAGEDIR)*>
<!ELEMENT STAGEDIR (#PCDATA)>
<!ELEMENT SUBTITLE (#PCDATA)>
<!ELEMENT SUBHEAD  (#PCDATA)>
```

1. Proposez un gestionnaire de document qui affiche le nombre d'actes, le nombre de scènes et le nombre total de vers d'une pièce basée sur la DTD `PLAY` (un vers est représenté par le contenu d'un élément `LINE`).
2. Proposez un gestionnaire de document qui affiche le nombre de vers contenus dans la tirade contenant le plus de vers (une tirade est représentée par le contenu d'un élément `SPEECH`) et qui donne le nom du personnage qui a prononcé cette plus longue tirade (s'il existe plusieurs tirade de même longueur maximale, on choisira la première rencontrée dans la pièce). Le personnage qui prononce une tirade est donné par le contenu de l'élément `SPEAKER`.
3. Proposez un gestionnaire de document qui vérifie la cohérence de la pièce : seuls les personnages listés par l'intermédiaire des éléments `PERSONNA` doivent pouvoir apparaître comme `SPEAKER`. Ils doivent de plus apparaître au moins une fois comme `SPEAKER`. En termes moins techniques, la liste des personnages ne doit faire apparaître que des personnages qui vont au moins prononcer une tirade dans la pièce, et de plus chaque tirade doit être prononcée par un personnage qui apparaît dans la liste. Le gestionnaire doit afficher un message pour chaque problème rencontré (aucun affichage si le document est cohérent).
4. Proposez un gestionnaire de document qui affiche la liste des personnages de la pièce classés par ordre de longueur moyenne des tirades prononcées.