INF3140–10 : Devoir 2
Automne 2009

À remettre *au plus tard* jeudi 26 novembre 2009, à 15h30
Pénalité de retard de 10 % par jour (complet ou partiel)
Aucun travail ne sera accepté après jeudi 3 décembre 2009, à 15h30
À faire seul ou en équipe de deux (2) personnes

Remarques préliminaires :

- L'objectif de ce travail est double :
  a. Vous familiariser avec l'utilisation de la bibliothèque d'expressions régulières du langage Java, `java.util.regex`, plus spécifiquement :
     `http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html`
     `http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Matcher.html`
  b. Vous familiariser avec l'utilisation d'une bibliothèque de collections fondée sur l'utilisation de **valeurs** (immuables) plutôt que d'**objets** (mutables) :
     `http://www.info2.uqam.ca/~tremblay/INF3140/OclCollections`
- Je vous suggère (**fortement**) d'utiliser l'environnement Unix des machines `rayons`, puisque ce sont sur ces machines que les tests pour la correction seront effectués.
  Si vous le désirez, vous pouvez utiliser tout autre environnement où Java (≥ 5.0) et JUnit (≥ 4.0) sont disponibles... mais en sachant que si votre solution ne fonctionne pas sur les `rayons`, alors c'est comme si **elle ne fonctionnait pas du tout, point à la ligne**.[1]
  Comme pour le devoir 1, j'ai défini un `makefile` pour que diverses tâches et tests puissent être effectués de façon automatique, et ce sur les machines `rayons` (ou Unix).
  La remise électronique devra nécessairement être effectuée à partir des `rayons`, avec la commande «`make remise`». Et la correction finale se fera, elle aussi, sous les `rayons`.

Un bref aperçu de XML

XML est défini comme suit dans Wikipedia :[2]

> XML (*Extensible Markup Language* «langage extensible de balisage») est un langage informatique de balisage générique. Il sert essentiellement à stocker/transférer des données de type texte Unicode structurées en champs arborescents. Ce langage est qualifié d'extensible car il permet à l'utilisateur de définir les balises des éléments.
>
> [...]
>
> Le World Wide Web Consortium (W3C), promoteur de standards favorisant l'échange d'informations sur Internet, recommande la syntaxe XML pour exprimer des langages de balisages spécifiques.

Nous allons illustrer XML et les notions de balise et attribut à l'aide de deux exemples.

```
<DOCUMENT>
  <TITRE>The Pragmatic Programmer--From journeyman to master</TITRE>

  <AUTEURS>
    <AUTEUR>
      <PRENOM>Dave</PRENOM>
      <NOM>Hunt</NOM>
    </AUTEUR>

    <AUTEUR>
      <PRENOM>Dave</PRENOM>
      <NOM>Thomas</NOM>
     </AUTEUR>
  </AUTEURS>

  <EDITEUR>Addison-Wesley</EDITEUR>

  <ANNEE>2000</ANNEE>
</DOCUMENT>
```

Figure 1: Description XML d'un document d'une bibliothèque.

a. Un exemple de balise pour définir un document d'une bibliothèque : Figure 1.

   Chaque élément d'information est identifié par une balise ouvrante — `<NomBalise>` — et une balise fermante — `</NomBalise>`. Les éléments compris entre la balise ouvrante et fermante peuvent être soit du texte simple (e.g., `Dave`), soit d'autres balises (donc structure hiérarchique récursive).

b. Un exemple de document HTML, qui peut être défini par une application XML, plus spécifiquement, un extrait de la page d'accueil pour le site Web du cours INF3140 : Figure 2.

   Quelques points à signaler :

   - La casse des balises n'est pas significative, i.e., `EM` = `em` = `eM`, etc.
   - La présence de certains espaces blancs n'est pas significative, plus précisément, ceux qui suivent le nom de la balise. Par contre, aucun espaces blanc n'est supposé apparaitre entre le «<» ouvrant et le nom de la balise.
   - Certaines balises sont «auto-fermantes». Exemple : `<br/>` est une abréviation pour deux balises ouvrante et fermante englobant un texte vide, i.e., `<br></br>`.
   - Une balise peut contenir des **attributs**, indiqués directement dans la balise elle-même, par exemple, pour la balise `<script>`, deux attributs sont spécifiés :
     - Attribut `langage`, de valeur `javascript` ;
     - Attribut `src`, de valeur `http://www.info2.uqam.ca/~tremblay/scripts/email.js`.

---

[1] En d'autres mots, je n'irai pas tester vos classes dans un autre environnement, et donc le fait que votre solution fonctionne «*dans un autre environnement*» **ne sera pas pris en compte** lors de la correction.

[2] `http://fr.wikipedia.org/wiki/Extensible_Markup_Language`

```
<HTML>

<HEAD>
<script language="javascript" src="http://www.info2.uqam.ca/~tremblay/scripts/email.js">
</script>
<TITLE>Mod&eacute;lisation et sp&eacute;cification formelle de logiciels</TITLE>
</HEAD>

<BODY BACKGROUND="http://www.labunix.uqam.ca/~inf3140/icones/fond2.jpg">

<H1  >INF3140 Mod&eacute;lisation et sp&eacute;cification formelle de logiciels</H1 >

<P/>

Cette page contient des informations pour le cours ``INF3140 --
Mod&eacute;lisation et sp&eacute;cification formelle de logiciels'',
cours enseign&eacute; &agrave; l'automne 2009 par
<A HREF="http://www.info2.uqam.ca/~tremblay" TARGET="."> Guy Tremblay</A>
du
<A HREF="http://www.info.uqam.ca" TARGET=".">d&eacute;partement d'informatique</A>
de
<A HREF="http://www.uqam.ca/" TARGET="."> l'UQAM</A>.
<P/>
<EM>Cr&eacute;ation initiale du site: 13/08/2008</em>


[...]


<ADDRESS>
<A HREF="http://www.labunix.uqam.ca/~tremblay">Guy Tremblay</A><br/>
D&eacute;partement d'informatique, UQAM<br/>
C.P. 8888, Succ. Centre-Ville<br/>
Montr&eacute;al, Qu&eacute;.<br/>
H3C 3P8<br/>
</ADDRESS>

</BODY>

<HR/>

<EM>Cr&eacute;ation initiale du site: 13/08/2008</em>

<BR/>

<STRONG>Derni&egrave;re mise &agrave; 29/08/2009</STRONG>

</HTML>
```

Figure 2: Page d'accueil (HTML) pour le site Web du cours INF3140.

Ce que vous devez faire

Vous devez développer, en utilisant et complétant divers éléments qui vous sont fournis, une mise en oeuvre pour un analyseur de documents XML.

Le rôle de cet analyseur est de pouvoir :

- Identifier les diverses balises qui apparaissent dans un document XML.
- Pour chaque balise rencontrée, identifier les différents attributs qui apparaissent dans les diverses utilisations de la balise.
- Pour chaque balise et chaque attribut, déterminer le nombre total de fois où l'attribut est spécifié — indépendamment de la valeur qui lui est associée.

Pour des descriptions plus spécifiques et concrètes de ce qui est attendu, voir le fichier `TesterAnalyseurXML.java`, un fichier de tests JUnit.

Plus spécifiquement :

a. Dans un premier temps, vous devez définir une mise en oeuvre pour le type abstrait `InfoBalise`, et ce dans la classe concrète `InfoBaliseImpl` partiellement définie qui vous est fournie.

b. Dans un deuxième temps, vous devez définir une mise en oeuvre pour le type abstrait `AnalyseurXML`, et ce dans la classe concrète `AnalyseurXMLAvecRegex` partiellement définie qui vous est fournie.

Contraintes de mise en oeuvre

a. Vous devez utiliser, dans votre mise en oeuvre, uniquement *i*) des types primitifs (`int`, `boolean`, etc.) ou *ii*) les versions objets (*boxed*) des types primitifs (`Integer`, `Boolean`), y compris le type `String`, ou *iii*) des collections de la bibliothèque `OclCollections` qui vous est fournie (voir plus bas). Il **n'est pas permis** d'utiliser ni des tableaux, ni des collections de la bibliothèque standard (objets mutables).

b. L'analyse des balises et attributs **doit se faire en utilisant les opérations de la bibliothèque `java.util.regex`**.

Indices de mise en oeuvre

Pour l'analyse des balises et attributs, je vous suggère de définir *plus qu'une expression régulière* et d'utiliser *plus qu'une opération* de «*matchage*». Les cas «simples» peuvent possiblement être traités en une seule fois, mais il risque d'être difficile (en fait, probablement impossible?) de n'utiliser qu'une seule expression régulière et une seule expression de *matchage* pour traiter les cas plus compliqués (entre autres, lorsque plusieurs attributs sont spécifiés dans une balise donnée).

Quelques simplifications

Pour simplifier le problème, vous pouvez faire les suppositions suivantes :

a. Les balises sont correctement spécifiées, donc sont correctement balancées lorsqu'approprié, avec des spécifications correctes d'attributs lorsque présents — entre autres, pas de « `>`» ou «`"`» oublié...

b. Les caractères «<» et «>» apparaissent uniquement pour spécifier des balises — c'est une supposition raisonnable, puisqu'on peut/doit normalement utiliser « `<`» et «`>`» pour indiquer ces caractères dans du texte normal d'un document XML, y compris les pages Web.

c. Les valeurs des attributs sont toujours indiquées entre « `"..."`», par exemple :

```
<A HREF="http://www.info.uqam.ca" TARGET=".">d&eacute;partement d'informatique</A>
```

d. Bien que les balises soient insensibles à la casse, vous **pouvez et devez supposer** que les noms des attributs eux, y sont sensibles. Donc, la balise suivante utilise deux attributs *distincts* :

```
<BALISE1 Attr1="abc" attr1="def">...</BALISE1>
```

Ce qui vous est fourni

- Un fichier d'archives (`Devoir2XML.tar.gz`) est disponible sur le site Web du cours contenant divers fichiers, décrits plus bas :

  `http://www.info2.uqam.ca/~tremblay/INF3140/Devoirs/Devoir2XML.tar.gz`

  Pour décompresser ce fichier d'archives (avec «-» à la fin de la ligne) :

  ```
  $ gzcat Devoir2XML.tar.gz | tar xvf -
  ```

Les divers fichiers qui vous sont fournis sont les suivants :

- `InfoBalise.java`, `FabriqueInfoBalise.java` et `AnalyseurXML.java` : Trois spécifications d'`interface` — **que vous ne devez pas modifier**.
- `TesterInfoBalise.java` et `TesterAnalyseurXML.java` : Deux programmes de tests.
- `makefile` : Un fichier permettant d'automatiser diverses tâches — voir plus bas.
- `OclCollections.jar` : Le fichier d'archive pour les classes de la bibliothèque `OclCollections`.
- `InfoBaliseImpl.java` : Squelette de classe (concrète) pour la représentation des informations associées à une balise XML.
- `AnalyseurXMLAvecRegex.java` : Squelette de classe (concrète) réalisant l'analyseur XML demandé, et ce à l'aide d'expressions régulières Java.

Les deux (2) derniers fichiers sont des «squelettes», i.e., des classes partiellement définies **que vous devez compléter** — et remettre. Quant aux fichiers d'interface, il est absolument interdit de les modifier — sous peine que votre solution ne compile pas lorsque je la corrigerai.

Commandes make :

- La commande «`make compile1`» permet de vérifier la syntaxe du fichier `InfoBaliseImpl.java`.
- La commande «`make tests1`» permet d'effectuer les tests pour le fichier `InfoBaliseImpl.java`.
- La commande «`make compile2`» permet de vérifier la syntaxe du fichier `AnalyseurXMLAvecRegex.java`.
- La commande «`make tests2`» permet d'effectuer les tests pour le fichier `AnalyseurXMLAvecRegex.java`.
- La commande «`make remise`» permet d'effectuer la remise : voir plus bas.
- La commande «`make clean`» permet d'effectuer un nettoyage des divers fichiers.

Ce que vous devez remettre

Remise papier

Vous devez remettre des versions papier des fichiers suivants :

a. `InfoBaliseImpl.java`

b. `AnalyseurXMLAvecRegex.java`

Pour la remise de ce document papier, vous devez utiliser **la page couverture (avec grille de correction)** disponible à la page suivante.

Remise électronique

Vous devez aussi remettre une version électronique des fichiers précédents, et ce en exécutant la commande suivante sur l'une des machines `rayon` : «`make remise`»

Toutefois, vous devez *auparavant* modifier la valeur de `CODES_PERMANENTS` dans le fichier `makefile` de façon à indiquer vos codes permanent si vous travaillez en équipe (de deux personnes maximum) ou votre code permanent si vous travaillez seul.

Retard

Pour qu'un travail ne soit pas considéré en retard, tant la version papier que la version électronique doivent être remises avant l'heure indiquée. Donc :

```
(versionPapierEnRetard or versionElectroniqueEnRetard) implies travailEnRetard
```

Remarques sur la correction :

- Si les fichiers remis contiennent des erreurs de syntaxe, alors vous perdrez une (très !) grande partie des points.
- J'effectuerai la correction possiblement avec d'autres tests (privés) que ceux qui vous sont fournis (publics).
- Dans la partie «Fonctionnement correct», j'évaluerai aussi le respect des **contraintes de mise en oeuvre** spécifiées plus haut.

Travail remis à Guy Tremblay

INF3140-10 : Devoir 2
À remettre au plus tard jeudi 26 novembre 2009, 15h30
(au début du cours)
Ne pas mettre dans une enveloppe

Nom	
Prénom	
Code permanent	
Courriel	

Nom	
Prénom	
Code permanent	
Courriel	

InfoBaliseImpl		
Compilation correcte		5 pts
Fonctionnement correct		5 pts
Style de programmation		5 pts
AnalyseurXMLAvecRegex		
Compilation correcte		5 pts
Fonctionnement correct		15 pts
Style de programmation		5 pts
Total		40 pts