Nom et prénom :

Institut Paul Lambin Session de Septembre 2018

Examen de Structures de données : Avancé

Christophe Damas, José Vander Meulen

Année(s) d'études : 2ème année de baccalauréat en informatique de gestion

Date et heure : lundi 20 aout à 13h00

Durée de l'examen : 3 h ; pas de sortie durant les 60 premières minutes

Contenu

Total : **/20**

Nom et prénom :

1. Questions sur machine

Avant de commencer cette partie, suivez les étapes suivantes :

1. Dézippez le fichier compressé (**Extract here**)
2. Renommez tous les noms et prénoms des différents répertoires. Ex : athletisme_DAMAS_CHRISTOPHE. (Evitez les caractères spéciaux : accents, …)
3. Utilisez la bonne version d'Eclipse (JEE ou JBOSS) : L'eclipse normal ne contient pas le plugin XSL.
4. Switchez votre workspace Eclipse afin qu'il soit positionné à la racine du Z.
5. Importez les différents projets dans votre workspace

Pour cette partie, on doit avoir dans le Z :

- **athletisme_ NOM_PRENOM**
- **graphe_ NOM_PRENOM**
- **recursion_NOM_PRENOM**
- **course_NOM_PRENOM**

Nom et prénom :

a) Athlétisme [8 pts]

Dans le répertoire `athletisme`, vous trouverez un document XML valide intitulé `athle.xsd`. Dans un document valide selon ce schéma, un club d'athlétisme enregistre les records de tous ses athlètes dans chaque discipline de l'athlétisme. Pour chaque athlète, le document ne retient donc qu'un seul record par discipline.
Voici son contenu :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="record">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]*.[0-9]{2}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="athletisme">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="discipline" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="nomDiscipline" type="xs:string" use="required" />
            <xs:attribute name="id" type="xs:ID" use="required" />
            <xs:attribute name="type" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="course" />
                  <xs:enumeration value="lancer" />
                  <xs:enumeration value="saut" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
        <xs:element name="club">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="athlete" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="record" maxOccurs="unbounded" minOccurs="0">
                      <xs:complexType>
                        <xs:simpleContent>
                          <xs:extension base="xs:record">
                            <xs:attribute name="discipline" type="xs:IDREF" use="required"/>
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="nom" type="xs:string" use="required" />
                  <xs:attribute name="dateDeNaissance" type="xs:date" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A la fin de l'examen, dans votre répertoire `restaurant`, les trois fichiers suivants devront être présents : `athle.dtd`, `athle.xq` et `athle.xsl`.

Nom et prénom :

DTD [3 pts]

On vous demande d'écrire la DTD `athle.dtd` qui permet de valider les mêmes documents que `athle.xsd` (tout en étant plus permissive évidemment)

XQuery [2 pts]

Dans un fichier `athle.xq`, écrivez la requête qui donne pour chaque discipline de type « saut », le record du club ainsi que l'athlète qui le détient. Pour simplifier, on supposera qu'il n'y a pas d'ex-aequo. Le résultat devrait ressembler à cela :

```
<records>
  <record nomDiscipline="saut en longueur" nom="Jean Dupont">7.43</record>
  <record nomDiscipline="saut en hauteur" nom="Pierre Dubois">1.57</record>
</records>
```

XSL [3pts]

On vous demande d'écrire un fichier `athle.xsl` qui transforme un fichier xml en html.
`athle.html` est le document résultant de la transformation d'un document `athle.xml`.
Le fichier html reprend toutes les disciplines de type « course » et affiche le classement des athlètes du club selon leur record.
Dans un browser, il s'affiche comme suit :

Voici les sources de ce fichier athle.html :

```
<html>
	<head> <title>Courses</title> </head>
	<body>
		<h1>100 m&egrave;tres</h1>
		<ol>	<li>Pierre Dubois  13.36</li>
			<li>Jean Dupont  13.99</li>
		</ol>
		<hr/>
		<h1>200 m&egrave;tres</h1>
		<ol>	<li>Jean Dupont  25.37</li>
			<li>Pierre Dubois  25.38</li>
		</ol>
		<hr/>
	</body>
</html>
```

(Remarque : lors de la génération, il est normal que votre fichier html ne soit pas indenté et qu'il y ait 2 différences avec le fichier ci-dessus : les balises <hr> et une balise <META>)

Nom et prénom :

b) Graphe [5 pts]

Dans le répertoire graphe, vous trouverez un document XML valide intitulé `countries.xml` (Attention! ce document n'est pas exactement le même qu'en juin). Ce fichier contient des informations à propos des pays du monde.

Dans ce fichier, il y a un élément `country` par pays. Dans cette question, nous allons nous focaliser uniquement sur leur donnée et sur un de ses attributs (le reste des informations du fichier peut être ignoré) :

- La donnée d'un élement country est une chaine composée de 3 caractères appelée cca3 qui permet d'identifier le pays. Deux pays ne peuvent pas avoir la même donnée.
- L'attribut `borders` donne tous les pays frontaliers.
    - Si un pays n'a aucune frontière (ce pays est une ile), alors cet attribut vaut la chaine vide.
    - Si un pays a plusieurs frontières, alors les attributs `cca3` des pays frontaliers sont séparés par des virgules.

Ces deux informations peuvent être vues comme un graphe non dirigé. Les sommets de ce graphe sont les pays. Il y a un arc entre deux pays si il y a une frontière entre ces deux pays.

Dans le répertoire `Graphe`, on vous fournit un squelette de code : la classe `Graph.java`. L'objectif de cette question est de compléter cette classe.

Parseur SAX [2 pts]

La classe Graph.java contient une classe interne SAXHandler. Ecrivez cette classe interne qui permettra de remplir la structure de données borders, une map qui associe à chaque pays (identifié par son attribut `cca3`) l'ensemble de ses pays frontaliers (également identifiés par leur attribut `cca3`). Comme SAXHandler est une classe interne non statique, elle peut accéder et modifier l'attribut borders de l'objet qui l' a créé.
La méthode `split` de la classe `String` pourrait vous être fort utile.

DFS [3pts]

Implémentez la méthode `dfs()` de la classe `Graph` qui prend le code `cca3` d'un pays de départ en paramètre. Cette méthode affiche à la sortie standard les codes `cca3` des différents pays qu'ils peut atteindre dans l'ordre d'un parcours en profondeur (DFS) depuis le pays de départ.
Par exemple, une des sorties possibles de la méthode `g.dfs("USA")` est :

```
USA MEX GTM HND NIC CRI PAN COL VEN GUY SUR GUF PER BOL PRY ARG URY CHL ECU BRA
SLV BLZ CAN
```

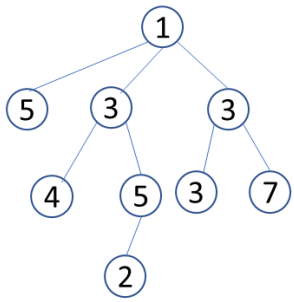
Nom et prénom :

c) Récursion [3 pts]

Dans le répertoire `recursion`, vous trouverez la classe `Tree.java`.

nbNoeudEgalA [1 pt]

Dans la classe `Tree`, implémentez la méthode récursive `nbNoeudEgalA()` qui compte le nombre de fois qu'une valeur apparait dans l'arbre.
Sur l'arbre ci-dessous, l'appel de `nbNoeudEgalA(5)` renvoie 2.



imprimerNoeudAvecProfondeur [2pts]

Dans la classe `Tree`, implémentez la méthode récursive `imprimerNoeudAvecProfondeur()` qui imprimera la valeur de chaque nœud suivi de la profondeur du nœud. Pour rappel, la profondeur d'un nœud dans un arbre est le nombre d'arêtes qu'il faut parcourir pour atteindre ce nœud à partir de la racine de l'arbre. Sur l'exemple de l'arbre ci-dessus, une réponse possible est (l'ordre des lignes n'a pas d'importance) :

```
1 : 0
5 : 1
3 : 1
4 : 2
5 : 2
2 : 3
3 : 1
3 : 2
7 : 2
```

Nom et prénom :

d) Course [2,5 pts]

Un club cycliste désire implémenter un programme simple pour gérer des courses de type « contre-la-montre ». Les « contre-la-montre » sont des épreuves lors desquelles chaque participant prend un départ individuel pour une course chronométrée avec comme objectif de réaliser le parcours défini le plus rapidement possible. Les différents concurrents partent ainsi les uns après les autres.
Le programme doit proposer 3 fonctionnalités :

- pouvoir encoder le départ d'un coureur,
- pouvoir encoder l'arrivée d'un coureur,
- afficher le classement provisoire des coureurs déjà arrivés.

Dans le répertoire course, nous fournissons un squelette de code.
La classe Coureur permet de gérer les informations sur le coureur (id unique et son nom) ainsi que son temps de départ et d'arrivée. La méthode `obtenirTemps()` permet de donner le temps global d'un coureur s'il a déjà terminé le parcours. Si possible, cette classe ne devrait pas être modifiée.
Vous devez compléter la classe Course en implémentant les méthodes `encoderDepart()`, `encoderArrivee()` et `afficherClassementProvisoire()`.
Vous aurez besoin de rajouter un/des attribut(s) dans `Course` qui devra (devront) être initialisé(s) dans le constructeur.

Vous devez garantir une efficacité maximale pour ces trois méthodes.

Dans les cadres ci-dessous, donnez les complexités de vos trois méthodes :

`encoderDepart()`:

`encoderArrivee()`:

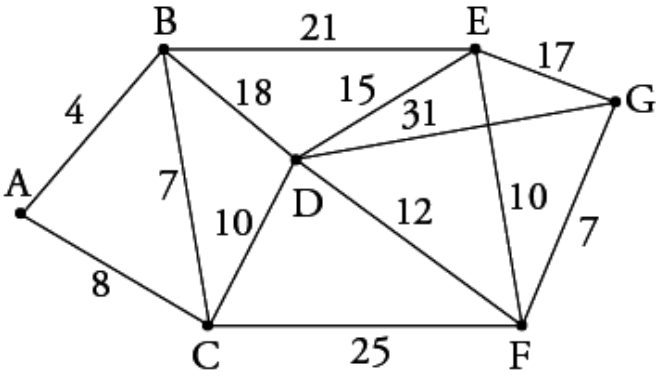
`afficherClassementProvisoire()`:

Une méthode main est fournie. L'output attendu est le suivant :

```
Coureur [id=3, nom=marc, Temps: 75]
Coureur [id=2, nom=pol, Temps: 90]
Coureur [id=1, nom=jean, Temps: 165]
----------------------------
Coureur [id=3, nom=marc, Temps: 75]
Coureur [id=4, nom=luc, Temps: 80]
Coureur [id=2, nom=pol, Temps: 90]
Coureur [id=5, nom=jim, Temps: 100]
Coureur [id=1, nom=jean, Temps: 165]
----------------------------
```

Nom et prénom :

2. Question sur papier [1,5 pts]



Ci-dessus, voici un réseau de train. Les stations sont symbolisées par les sommets A, B, C, D, E, F et G. Chaque arc représente une ligne reliant deux gares. Les temps de parcours (correspondance comprise) en minutes entre chaque sommet ont été rajoutés sur le graphe.

Déterminer le plus court chemin en minutes, reliant la gare B à la gare G. Justifiez votre réponse.

Nom et prénom :