Nom et prénom :

Institut Paul Lambin Session de Juin 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 28 mai à 14h00

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 : restaurant_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 :

- **restaurant_ NOM_PRENOM**
- **graphe_ NOM_PRENOM**
- **file_arbre_NOM_PRENOM**

Nom et prénom :

a) Restaurant [8 pts]

Dans le répertoire `restaurant`, vous trouverez un document XML valide intitulé `restaurant.xsd`. Voici son contenu :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="idPlat">
    <xs:restriction base="xs:ID">
      <xs:pattern value="p[0-9]{3}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="entree" />
      <xs:enumeration value="principal" />
      <xs:enumeration value="dessert" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="restaurant">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="client" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="nom" type="xs:string" use="required" />
            <xs:attribute name="id" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:ID">
                  <xs:pattern value="cl[0-9]{4}" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="adresse" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="carte">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="plat" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="prix" type="xs:positiveInteger" use="required" />
                      <xs:attribute name="monnaie" type="xs:string" fixed="EUR" />
                      <xs:attribute name="id" type="idPlat" use="required" />
                      <xs:attribute name="type" type="type" default="principal"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="commande" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ligneCommande" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="plat" type="xs:IDREF" use="required"  />
                  <xs:attribute name="quantite" type="xs:positiveInteger" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="client" type="xs:IDREF" use="required" />
            <xs:attribute name="date" type="xs:date" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Nom et prénom :

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

DTD [3 pts]

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

XQuery [2 pts]

Dans un fichier `restaurant.xq`, écrivez la requête qui donne pour chaque plat le nombre de fois qu'il a été commandé. Les plats doivent être triés par le nombre de fois qu'ils ont été commandés. Le résultat devrait ressembler à cela :

```
<result>
	<plat>
		<nom>Sandwich Club</nom>
		<nombre>5</nombre>
	</plat>
	<plat>
		<nom>Spaghetti bolognaise</nom>
		<nombre>3</nombre>
	</plat>
	<plat>
		<nom>Pizza Hawaii</nom>
		<nombre>1</nombre>
	</plat>
</result>
```

Remarque : Si un plat a été commandé dans deux commandes et qu'il a été commandé une fois dans la première, et deux fois dans la deuxième (attribut quantité), alors le nombre associé à ce plat dans le résultat devrait être 3.

XSL [3pts]

On vous demande d'écrire un fichier `restaurant.xsl` qui transforme un fichier xml en html. `restaurant.html` est le document résultant de la transformation d'un document `restaurant.xml`. Dans un browser, il s'affiche comme suit :

Nom et prénom :

Les clients sont triés par nom (on associe au premier client par ordre alphabétique le numéro 1, et ainsi de suite). Les commandes sont triées par date. Si un client n'a rien commandé, il faut mentionner « Pas de commande ».
Voici les sources de ce fichier restaurant.html :

```
<html>
	<head>
		<title>Commandes</title>
	</head>
	<body>
		<h1> Le fichier contient 3 clients </h1>
		<hr/>
		<h1> Client 1: Damas</h1>
		<p> Commande du 2017-05-27
			<ul>
				<li>1 Pizza Hawaii</li>
				<li>2 Sandwich Club</li>
			</ul>
		</p>
		<p> Commande du 2017-05-31
			<ul>
				<li>2 Spaghetti bolognaise</li>
				<li>1 Sandwich Club</li>
			</ul>
		</p>
		<hr/>
		<h1> Client 2: Leconte</h1>
		<p> Commande du 2017-05-29
			<ul>
				<li>1 Spaghetti bolognaise</li>
				<li>2 Sandwich Club</li>
			</ul>
		</p>
		<hr/>
		<h1> Client 3: Vander Meulen</h1>
		<p>Pas de commande</p>
		<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 [7 pts]

Dans le répertoire graphe, vous trouverez un document XML valide intitulé `countries.xml`. 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 deux attributs de ces éléments (le reste des informations du fichier peut être ignoré) :

- L'attribut `cca3` est une chaine composée de 3 caractères qui permet d'identifier le pays. Deux pays ne peuvent pas avoir le même attribut `cca3`.
- 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 DOM [2 pts]

A l'aide d'un parseur DOM, implémentez le constructeur de la classe `Graph.java`. Ce constructeur prendra le fichier xml en paramètre et remplira 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`).
La méthode `split` de la classe `String` pourrait vous être fort utile.

BFS [3pts]

Implémentez la méthode `bfs()` 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 largeur (BFS) depuis le pays de départ.
Par exemple, une des sorties possibles de la méthode `g.bfs("USA")` est :

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

Composantes Connexes [2pts]

A l'aide de la classe UnionFind.java fournie, écrivez la méthode `nbComposantesConnexes()` qui renvoie le nombre de composantes connexes du graphe.

Nom et prénom :

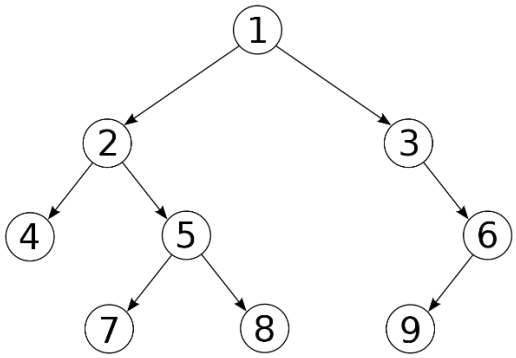
c) Files de priorité et arbres [3 pts]

Dans le répertoire file_arbre, vous trouverez les classes Tree.java et PriorityQueue.java.

Calcul du nombre de nœuds internes [1 pt]

Dans la classe Tree, implémentez la méthode nbNoeudInterne() qui calcule le nombre de nœud interne d'un arbre. Votre implémentation doit être récursive et ne peut appeler de méthode auxiliaire.

Pour rappel, un nœud interne est un nœud qui n'est pas une feuille. L'arbre ci-dessous a donc 5 nœuds internes.



Parcours DFS dans une file de priorité [2pts]

Une file de priorité représente un arbre sous forme de tableau. On vous demande d'implémenter la méthode `dfs()` qui imprime à la sortie standard les valeurs des nœuds de l'arbre dans l'ordre d'un parcours en profondeur (DFS). Une méthode `main()` est fournie. L'affichage de cette méthode `main()` doit être le suivant :

```
95 85 57 7 21 23 52 44 17
```

Nom et prénom :

2. Question sur papier

Huffman [2 pts]

Supposons qu'un texte à compresser soit formé des lettres suivantes avec comme fréquence :

A	B	C	D	E	F	G	H
13	5	8	6	14	4	2	1

Construisez sur papier l'arbre de Huffman pour ces fréquences.
Codez ensuite le texte : GAFFE.
Décodez enfin la suite de bits 11010111110.

Nom et prénom :