Nom et prénom :

Institut Paul Lambin Session de juin 2020

Examen de Structures de données

Christophe Damas, José Vander Meulen

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

Date et heure : vendredi 27 mai à 9h00

Durée de l'examen : 2 h 45

Contenu

Total : **/20**

Nom et prénom :

1. Consignes

Dans votre archive d'examen, vous devez avoir les répertoires suivants:

- **examen**
- **graphe**
- **arbre**
- **inscription**

A la fin de l'examen, soumettez uniquement les fichiers que vous avez modifié à savoir :

- **Pour la question a : examen.dtd et examen.xsl**
- **Pour la question b : Graph.java**
- **Pour la question c : Tree.java**
- **Pour la question d : tout le projet**
- **Pour la question e : huffman.txt**

Nom et prénom :

2. Questions

a) Examen [7 pts]

Dans le répertoire `examen`, vous trouverez un schéma XML valide intitulé `examen.xsd`.
Dans un document valide selon ce schéma, une école enregistre les horaires d'examens ainsi que la répartition des étudiants un peu comme sur Celcat.
Voici un exemple de répartition: `Répartition A017 pour l'examen DB2 de A à Koc - A019 pour l'examen DB2 de Köksal à Michiels - A026 pour l'examen DB2 de Molitor à Z.`

DTD [3 pts]

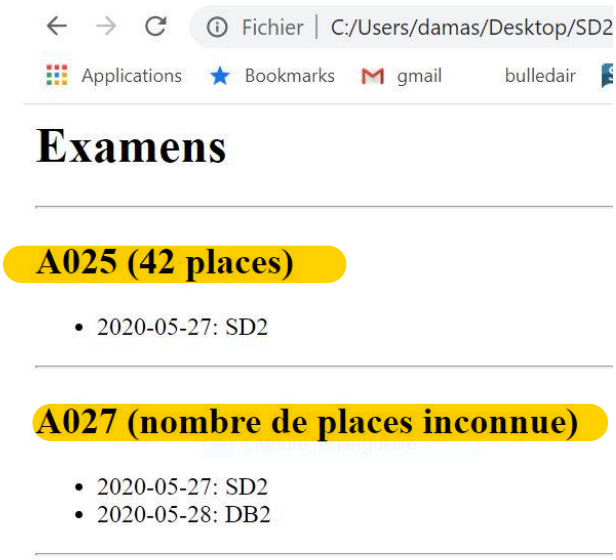
On vous demande d'écrire la DTD `examen.dtd` qui permet de valider les mêmes documents que `examen.xsd` (tout en étant plus permissive évidemment)
Remarque : Voici un exemple de date en XML : `2020-05-27`

XSL [4pts]

On vous demande d'écrire un fichier `examen.xsl` qui transforme un fichier xml valide par rapport à `examen.xsd` en html.
Le fichier généré affiche toutes les locaux avec leur nombre de places. Si le nombre de places n'est pas précisé dans le document xml, alors il faut afficher « nombre de places inconnue ».
Pour chaque local, il faut afficher tous les examens s'y déroulant triés par date.

Ci-dessous, vous trouverez la capture d'écran ainsi que le fichier source d'un `examen.html` obtenu à partir d'un fichier xml `contenant` les informations suivantes :

- il y a 2 locaux (A025 et A027) et 2 examens (SD2 et DB2),
- A025 a 42 places et le nombre de place du local A027 n'est pas précisé,
- l'examen SD2 se déroule le 27 mai dans les deux locaux,
- l'examen DB2 se déroule le 28 mai au A027.



Nom et prénom :

```
<html>
	<head>
		<title>Examens</title>
	</head>
	<body>
		<h1>Examens</h1>
		<hr/>
		<h2>A025 (42 places)</h2>
		<ul>
			<li>2020-05-27: SD2</li>
		</ul>
		<hr/>
		<h2>A027 (nombre de places inconnue)</h2>
		<ul>
			<li>2020-05-27: SD2</li>
			<li>2020-05-28: DB2</li>
		</ul>
		<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 [6 pts]

Dans le répertoire graphe, vous trouverez un document XML valide intitulé `stib.xml`. Ce fichier est une version simplifiée de celui du projet. Il contient uniquement les lignes du réseau stib (il n'y a plus de stop).

Ce fichier peut être vu comme un graphe dirigé. Les sommets de ce graphe sont les stations. Il y a un arc entre deux stations si un tronçon les relie.

Dans le répertoire `Graphe`, on vous fournit un squelette de code : la classe `Graph.java` et la classe `Troncon.java`. L'implémentation de la classe `Graph` est basée sur une liste d'arcs. Cette classe va retenir dans la liste `troncons` l'ensemble des tronçons du document xml.

L'objectif de cette question est de compléter la classe `Graphe.java`.

Parseur DOM [3 pts]

A l'aide d'un parseur DOM, implémentez le constructeur de la classe Graphe.java. Le constructeur permettra de remplir la structure de données `troncons`, une liste de "`troncon`" qui contiendra l'ensemble des tronçons du document xml. En java, un `troncon` retient sa ligne (attribut nom de l'élément ligne), son départ, son arrivée et sa durée (donnée de l'élément troncon).

BFS [3pts]

Dans la classe Graph, nous avons implémenté une méthode `bfs(String depart, String arrivee)` qui, pour l'instant, affiche à la sortie standard les nom des stations que l'on peut atteindre dans l'ordre d'un parcours en largeur (BFS) depuis la station de départ.

L'objectif de cette question est de compléter ce code afin que cette méthode affiche le chemin entre deux stations avec le moins de tronçons possibles. Pour un chemin, il faut simplement afficher les noms des stations par lesquels on passe. Vous pouvez supprimer les sysout du code initial mais vous devez garder les autres lignes.

Par exemple, voici une sortie possible pour `g.bfs("MALIBRAN", "ALMA")` :

```
MALIBRAN BLYCKAERTS IDALIE LUXEMBOURG SCHUMAN MERODE MONTGOMERY JOSEPH.-CHARLOTTE GRIBAUMONT TOMBERG ROODEBEEK VANDERVELDE ALMA
```

Il est peut-être possible de trouver d'autres itinéraires avec 13 noms de station.

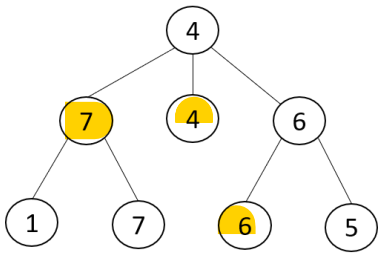
Nom et prénom :

c) Arbres [3 pts]

Dans le projet **arbre**, on vous fournit la classe `Tree` qui implémente des méthodes basiques sur les arbres

1) Dans `Tree`, implémentez la méthode `auMoins1EnfantDeMemeValeur()` qui renverra true si chaque nœud interne a au moins un fils qui lui est égal. Les nœuds internes sont tous les nœuds sauf les feuilles.

   Ex : la méthode renverra `true` pour l'arbre ci-dessous. En effet, la racine (4) a bien un fils qui a la même valeur. Son premier fils (7) et son troisième fils (6) ont la même propriété. Les autres nœuds sont des feuilles.



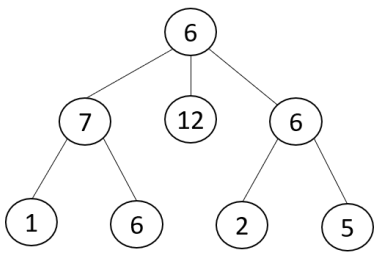
   Remarque : le plus petit arbre qu'on peut construire est un arbre contenant uniquement une racine. Si on appelle la méthode `auMoins1EnfantDeMemeValeur()` sur ce type d'arbre, la méthode renvoie `true`.

2) Dans `Tree`, implémentez la méthode `afficherNoeudsAvecMaximumAncetres()`. Cette méthode imprime à la sortie standard tous les noeuds suivis du maximum de la valeur du noeud et de celle de ses ancêtres.

   Ex : pour l'arbre dessiné au bas de la page, une sortie possible est :

```
6:6
7:7
1:7
6:7
12:12
6:6
2:6
5:6
```

   L'ordre des lignes n'a pas d'importance.



Nom et prénom :

d) Inscription [3 pts]

Dans le projet `inscription`, on vous fournit un squelette de code qui propose un système d'inscription d'élèves dans des écoles.
Dans ce système, les inscriptions se déroulent de la manière suivante :

- D'abord, les élèves peuvent faire jusqu'à trois demandes d'inscription à des écoles. Notez qu'il n'y a pas d'ordre de préférence entre les écoles.
- Ensuite, chaque école qui a un nombre limité de places va parcourir ses demandes d'inscription et inscrit les élèves selon l'ordre premier arrivé, premier servi (FIFO). Une école ne peut pas inscrire un élève déjà inscrit dans une autre école. L'ordre dans lequel on va remplir les écoles n'est pas fixé et peut varier.

Exemple :
Jean demande de s'inscrire à école1, école2, école3 et école4. Seuls les 3 premières demandes seront enregistrées car un élève ne peut pas faire plus de 3 demandes d'inscription.
Ensuite, Marc demande de s'inscrire à école1 et école2.
Ensuite, école2 qui n'a qu'une place procède en premier au remplissage de son école : elle inscrit Jean qui s'est inscrit en premier.
Finalement, école1 qui n'a qu'une place décide de remplir son école : elle inscrit Marc (car Jean est déjà inscrit à école2).
Le squelette de code contient :

- une classe `Eleve` qui retient simplement le prénom et le nom d'un élève
- une classe `Ecole` qui retient son nom, le nombre de place disponible dans l'école ainsi que l'ensemble des élèves inscrits. Elle possède une méthode `ajouterEleve` pour inscrire un élève à l'école.
- une classe `Inscription` qui s'occupe des inscriptions des élèves aux écoles.
- une classe `Test` pour tester vos classes

Vous devez compléter la classe Inscription en implémentant les méthodes `demandeInscription` et `remplirEcole`. Les autres classes ne peuvent être changées.

Vous aurez besoin de rajouter des attributs dans `Inscription` qui devront être initialisés dans le constructeur. Dans vos deux méthodes, le seul cas d'erreur à gérer est le cas où un élève fait plus de trois demandes d'inscription (`RuntimeException`).

Vous devez garantir une efficacité maximale pour ces deux méthodes. **Dans votre code, vous devez donner en commentaire les complexités de vos deux méthodes.**

Une classe `Test` est fournie. L'output attendu est le suivant :

```
Ecole [nom=IPL, nbPlace=2, inscrits=Jose Vander Meulen,Christophe Damas]
--------------
Ecole [nom=UCL, nbPlace=1, inscrits=Emmeline Leconte]
--------------
Ecole [nom=ephec, nbPlace=1]
--------------
Ecole [nom=ulb, nbPlace=1, inscrits=Gregory Seront]
--------------
Exception in thread "main" java.lang.RuntimeException
	at Inscription.demandeInscription(Inscription.java:26)
	at Test.main(Test.java:34)
```

Nom et prénom :

e) Huffman [1 pt]

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

A	B	C	D	E	F
3	1	7	6	5	12

Construisez sur papier l'arbre de Huffman pour ces fréquences.
Codez ensuite le texte « CAFE ».

Pour cette question, soumettez uniquement la réponse finale (chaîne de bit codant le mot « CAFE ») dans un fichier huffman.txt. L'arbre ne doit pas être soumis.