Examen Ecrit

Tous documents sur support papier autorisés. Durée 2h.

Dans ce sujet, nous utiliserons la classe générique `TreeSet<T>` de l'API standard de Java. Cette classe, qui implémente l'interface `Set<T>`, permet de représenter des ensembles à l'aide d'une structure interne arborescente. Un ensemble est une collection d'objets sans doublons. `TreeSet` dispose entre autres des méthodes suivantes :

- un constructeur sans paramètre pour créer un ensemble vide
- s.size() retourne le nombre d'éléments
- s.add(e) — ajoute e à s si e n'est pas déjà dans s.
- s.contains(e) — retourne vrai si s contient e.
- s1.equals(s2) — retourne vrai si s1 et s2 contiennent les mêmes objets.

Nous utiliserons également la classe `TreeMap<K,V>`, qui implémente l'interface `Map<K,V>` pour représenter des dictionnaires associatifs. Vous avez utilisé des `Map`s lors du TP sur les graphes.

On vous donne le code suivant, décrivant deux éléments d'un logiciel pour des réseaux ferroviaires.

```
public class Gare {
   private String nom = "";
   public Gare(String nom) {this.nom=nom;}
   public String getNom() {return nom;}
   public void setNom(String nom) {this.nom = nom;}
}

// Le type de signalisation en cabine est un standard de description des informations
// données au conducteur d'un train

public enum TypeSignalisationCabine { // vous pouvez l'abréger en TSC dans vos réponses
   TVM,LZB,TBL,ATB;
}
```

Question 1. Ecrire une interface `ILigne`.
On souhaite représenter les lignes (voies ferrées pour les trains) dans le cadre des réseaux ferroviaires. Les lignes disposent :

1. de deux méthodes d'accès à un identifiant qui est une chaîne de caractères, (a) la première pour en connaître la valeur (accesseur en lecture), (b) la seconde pour le modifier (accesseur en écriture) ;
2. d'une méthode `getGares()` permettant de connaître l'ensemble (`Set`) des deux gares extrémités de cette ligne ;
3. de deux méthodes d'accès à un type de signalisation en cabine correspondant aux informations sur le trafic et sur la voie, (a) la première pour en connaître la valeur (accesseur en lecture), (b) la seconde pour le modifier (accesseur en écriture) ;
4. d'une méthode de classe (et non d'instance) `boolean memeDesserte(ILigne l1, ILigne l2)` retournant vrai si les deux lignes passées en paramètre ont le même ensemble de gares extrémités ;
5. d'une méthode retournant une description de la ligne sous forme d'une chaîne de caractères composée de l'identifiant de la ligne et des noms des deux gares extrémités.

Ecrivez le code Java de l'interface représentant les objets disposant de ces méthodes.

Question 2. Implémenter une interface.
(1) Ecrivez une classe `Ligne` qui implémente l'interface `ILigne`. N'écrivez que les attributs et les méthodes nécessaires pour implémenter l'interface. Les deux gares extrémités doivent être stockées dans un ensemble (interface `Set`, classe `TreeSet`).
(2) Puis écrivez l'entête *(uniquement l'entête)* d'une classe `LigneGV` qui est sous-classe de `Ligne`. Elle représente les lignes réservées aux trains à grande vitesse.

Question 3. Ecrire exception(s) et assertion(s).
Ecrivez le corps d'un constructeur pour la classe `Ligne` avec les informations suivantes.

- il admet 4 paramètres : un identifiant, une première gare, une deuxième gare, un type de signalisation cabine.
- si l'un des deux paramètres supposés contenir une gare a la valeur `null`, une exception est signalée (vous devez tout mettre en œuvre en Java : la classe d'exception, le signalement, la déclaration dans la signature du constructeur).
- les deux gares doivent être différentes : comment traiteriez-vous ce problème (exception, assertion, message d'erreur) ? Expliquez votre décision et mettez en place la solution choisie.
- une assertion contrôle qu'à la fin du constructeur, l'ensemble de gares extrémités est de taille 2.

Question 4. Ecrire une méthode d'accès.
Ecrivez dans la classe `Ligne` une méthode `Gare gareOpposee(Gare g)`. Si `g` est bien l'une des deux gares extrémités de la ligne, la méthode retourne l'autre gare extrémité, sinon elle retourne la valeur `null`.

Question 5. Ecrire une classe générique.
Ecrivez l'entête et les attributs d'une classe générique `ReseauFerroviaire`, paramétrée par un type de ligne respectant l'interface `ILigne`. Un réseau ferroviaire contient un dictionnaire associatif (interface `Map`, classe `TreeMap`) de lignes. Les clefs sont les identifiants des lignes ; les valeurs sont les lignes. Le réseau a également un opérateur, dont on ne stocke que le nom.

Question 6. Manipulation d'un dictionnaire associatif.
Ecrivez dans la classe `ReseauFerroviaire` une méthode permettant d'ajouter une nouvelle ligne, passée en paramètre.

Question 7. Compréhension des streams.
En supposant que le dictionnaire associatif des lignes du réseau ferroviaire se nomme `collectionLignes`, on peut écrire la méthode suivante basée sur les `streams`.

```
ArrayList<Gare> xxx(Gare g){
     ArrayList<Gare> res = new ArrayList<Gare> ();
     this.collectionLignes.values()
                          .stream()
                          .filter(ligne -> ligne.getGares().contains(g))
                          .map(ligne -> ligne.gareOpposee(g))
                          .forEach(l -> res.add(l));
     return res;
}
```

Expliquez ce qu'elle fait et réécrivez cette méthode avec une itération classique (`for` ou `while`).

Question 8. Ecrire un main

Ecrivez un main : (1) Créez deux gares, (2) Créez une ligne allant de la première à la seconde, (3) Créez un réseau ferroviaire de lignes, (4) Ajoutez la ligne créée précédemment au réseau, (5) Capturez la ou les exceptions qui risquent de se produire dans le programme.