

PROGRAMMATION AVANCÉE ET STRUCTURES DE DONNÉES

Session 1 — Janvier 2015

Tous documents autorisés — durée 3 heures

Le barème indiqué est indicatif et peut légèrement varier lors de la correction. Son total est de 22 points : votre note finale sera le minimum entre 20 et la note obtenue suivant ce barème. Préciser pour chaque définition de méthode la classe dans laquelle cette définition doit être présente.

Réseaux sociaux

Nous nous proposons de programmer quelques classes qui pourraient être utiles à la mise en place de réseaux sociaux. Nous considérons pour cela 5 classes, brièvement détaillées ci-dessous.

- La classe `Membre` permet de représenter les membres d'un réseau, grâce à 2 attributs : `nom` qui permet de stocker le nom sous lequel le membre est connu et `reseau` qui référence le réseau social dont le membre fait partie.
- La classe `Moderateur` est une sous-classe de `Membre` : les modérateurs sont des membres qui ont en plus le privilège de pouvoir supprimer d'autres membres ou encore de créer des thèmes (ou centres d'intérêt).
- Une instance de la classe `Reseau` représente un réseau social. Les membres y sont référencés depuis l'attribut `membres`, alors que les 2 autres attributs regroupent les préférences de ceux-ci : `fans` est une `Map` qui associe à une clé de type `String` dénotant un thème l'ensemble des membres aimant ce thème, tandis que `antis` est une `Map` qui associe à une clé de type `String` dénotant un thème l'ensemble des membres détestant ce thème.
- La classe `ListeTrie` est une classe utilitaire dont les instances sont des listes d'éléments comparables entre eux, triés par ordre croissant.
- Une instance de la classe `Affinite` est le résultat d'un calcul d'affinité d'un membre avec un autre membre. L'affinité des membres est calculée en tenant compte des thèmes qu'ils aiment et de ceux qu'ils détestent.

1 Classe `ListeTrie` (3 points)

Chaque instance de la classe `ListeTrie` possède un attribut `elements` dans lequel sont stockés les éléments de la liste. Afin de s'assurer que ces éléments sont toujours correctement triés, il suffit d'insérer chaque nouvel élément à la bonne place, c'est-à-dire après les éléments qui lui sont inférieurs et avant les éléments qui lui sont supérieurs.

Donner la définition des constructeurs de la classe `ListeTrie`. Le constructeur sans paramètre permet d'obtenir une liste vide. Le constructeur avec un paramètre de type `Collection` doit permettre d'obtenir une liste contenant initialement tous les éléments de la collection dans le bon ordre.

Donner la définition de la méthode `get(int i)` qui retourne l'élément d'indice `i` dans la liste. Cette méthode lève une exception de type `IndexOutOfBoundsException` si l'indice fourni est incorrect.

Donner la définition de la méthode `size()` qui retourne le nombre d'éléments de la liste.

Donner la définition de la méthode `add(E e)` qui permet d'ajouter un élément à sa place dans la liste et retourne `true` si la liste a été modifiée.

2 Constructeurs (2 points)

Donner la définition des constructeurs des 4 classes `Membre`, `Moderateur`, `Affinite` et `Reseau`. Pour ces 3 premières classes, les paramètres servent à initialiser directement les attributs. Le

constructeur sans paramètre de la classe `Reseau` permet d'obtenir un réseau vide, c'est-à-dire sans membres et sans thèmes.

3 Accesseurs simples (1 point)

Donner la définition des accesseurs en lecture `getCandidat()` et `getScore()` de la classe `Affinite`.

4 Comparaison (1 point)

Donner la définition de la méthode `compareTo(Affinite a)` de la classe `Affinite`, ainsi que celle de la méthode `compareTo(Membre m)` de la classe `Membre`. Chacune de ces méthodes retourne un `int` strictement négatif si `this` est inférieur au paramètre, égal à 0 s'il est équivalent au paramètre, strictement positif s'il est supérieur au paramètre. Les affinités se comparent sur la base de leur `score`, les membres sur la base de leur `nom`.

5 Nouveau membre et nouveau thème (1 point)

Donner la définition de la méthode `nouveauMembre(String nom)` de la classe `Reseau` qui permet de créer un nouveau membre et l'ajoute dans l'ensemble des membres du réseau. Cette méthode retourne `true` si le réseau a bien été modifié.

Donner la définition de la méthode `nouveauTheme(String nom)` de la classe `Reseau` qui permet de créer un nouveau thème dans le réseau. Un thème (simplement dénoté par une `String`) est présent dans le réseau s'il est une clé de `fans` et de `antis`. On s'attache à ce que l'ensemble des clés de `fans` et l'ensemble des clés de `antis` soient toujours équivalents. Cette méthode retourne `true` si le réseau a bien été modifié.

6 J'aime ou je déteste (5 points)

Donner la définition des méthodes privées `valideMembre(Membre m)` et `valideTheme(String theme)` de la classe `Reseau`. Ces méthodes lèvent une exception de type `RuntimeException` si le paramètre fourni n'est pas respectivement un membre du réseau ou un thème du réseau.

Donner la définition des méthodes `aime(Membre m, String theme)` et `deteste(Membre m, String theme)` de la classe `Reseau` qui permettent de prendre en compte les préférences d'un membre vis à vis d'un thème. Si le membre fourni n'est pas membre du réseau ou si le thème est absent du réseau, une exception de type `RuntimeException` doit être levée.

Donner la définition des méthodes `aime(String theme)` et `deteste(String theme)` des classes `Membre` et `Moderateur`, qui permettent de prendre respectivement en compte qu'un membre déclare aimer ou détester le thème fourni en paramètre. Si un membre déclare aimer ou détester un thème absent de son réseau, une exception de type `RuntimeException` doit être levée, à moins qu'il ne soit modérateur, auquel cas le thème doit être automatiquement ajouté dans le réseau.

7 Suppression de membre et suppression de thème (2 points)

Donner la définition des méthodes `removeMembre(Membre m)` et `removeTheme(String theme)` des classes `Reseau` et `Moderateur`.

8 Meilleurs thèmes (2 points)

Donner la définition de la méthode `meilleursThemes()` de la classe `Reseau` qui retourne un `Set` des thèmes les plus aimés dans le réseau, c'est-à-dire ceux ayant le plus grand nombre de fans.

9 Recherche d'amis pour un membre (5 points)

Donner la définition de la méthode privée `nbEltsCommuns(Set<String> s1, Set<String> s2)` de la classe `Reseau` qui retourne le nombre d'éléments communs à `s1` et `s2`.

Donner la définition de la méthode `getAimesDe(Membre m)` de la classe `Reseau` qui retourne un `Set` des thèmes aimés par `m`. Cette méthode lève une exception de type `RuntimeException` si `m` n'est

pas membre du réseau. La définition de la méthode `getDetestesDe(Membre m)` **n'est pas demandée**, cette méthode est similaire à la méthode `getAimesDe(Membre m)` et retourne un Set des thèmes détestés de m.

Donner la définition de la méthode `rechercheAmis(Membre m)` de la classe `Reseau` qui retourne une liste triée des affinités de m avec tous les autres membres du réseau. Le score d'affinité entre 2 membres est calculé de la manière suivante : on compte +1 pour tout thème aimé ou détesté en commun et -1 pour tout thème aimé par l'un et détesté par l'autre ou inversement.

10 Annexes : squelettes de classe

```
package reseau;
import java.util.*;

public class ListeTrie<E extends Comparable<E>> extends AbstractList<E>
    implements List<E> {
    private List<E> elements;

    public ListeTrie() {
    public ListeTrie(Collection<? extends E> c) {...}

    public E get(int i) {...}
    public int size() {...}

    public boolean add(E e) {...}
}
```

```
package reseau;

public class Affinite implements Comparable<Affinite>{
    private Membre candidat;
    private int score;

    public Affinite(Membre candidat, int score) {...}

    public Membre getCandidat() {...}
    public int getScore() {...}

    public int compareTo(Affinite a) {...}
}
```

```
package reseau;

public class Membre implements Comparable<Membre> {
    private String nom;
    protected Reseau reseau;

    public Membre(String nom, Reseau reseau) {...}

    public int compareTo(Membre m) {...}

    public void aime(String theme) {...}
    public void deteste(String theme) {...}
}
```

```

package reseau;

public class Modérateur extends Membre {
    public Modérateur(String nom, Réseau reseau) {...}

    public void aime(String theme) {...}
    public void deteste(String theme) {...}

    public void removeMembre(Membre m) {...}
    public void removeTheme(String theme) {...}
}

```

```

package reseau;
import java.util.*;

public class Réseau {
    private Set<Membre> membres;
    private Map<String, Set<Membre>> fans;
    private Map<String, Set<Membre>> antis;

    public Réseau() {...}

    public boolean nouveauMembre(String nom) {...}
    public boolean nouveauTheme(String theme) {...}

    private void valideMembre(Membre m) {...}
    private void valideTheme(String theme) {...}
    public void aime(Membre m, String theme) {...}
    public void deteste(Membre m, String theme) {...}

    public boolean removeMembre(Membre m) {...}
    public boolean removeTheme(String theme) {...}

    public Set<String> meilleursThemes() {...}

    private int nbEltsCommuns(Set<String> s1, Set<String> s2) {...}
    public Set<String> getAimesDe(Membre m) {...}
    public Set<String> getDetestesDe(Membre m) {...}
    public List<Triee<Affinite>> rechercheAmis(Membre m) {...}
}

```