II.1102 : Algorithmique et Programmation
TP noté 2

Patrick Wang

Vendredi 08 novembre 2019

Table des matières

1 Consignes générales

— Aucun document n'est autorisé. Vous pouvez cependant vous aider de la documentation officielle de Java.
— Prenez garde à bien lire l'énoncé dans son ensemble avant de démarrer les exercices.
— Ce TP est à réaliser **individuellement**.
— Prenez bien garde à bien nommer votre projet en précisant votre groupe d'APP, prénom et nom : `TP2_GX_PRENOM_NOM`.
— L'ensemble du TP sera à réaliser à l'intérieur d'une seule et unique classe `Main`. Merci de bien respecter ce nom lors de la création de votre projet.
— Pour déposer voter projet, créez une archive au format ZIP de l'ensemble de votre projet. Les formats 7z ou RAR ne seront pas acceptés.
— Une boîte de dépôt est créée sur Moodle afin de déposer vos projets. Celle-ci se fermera 15min après la fin du temps réglementaire.
— Toute tentative de triche sera sanctionnée d'un 0.

— Un point sur 20 est attribué à la lisibilité de votre code. Cela concerne l'indentation de votre code, le choix des noms de variables, et la présence de commentaires pertinents.

2 Exercices

2.1 Détection d'anagrammes

Une anagramme est une figure de style qui inverse ou permute les lettres d'un mot pour en créer un nouveau. Par exemple, les mots suivants sont tous des anagrammes : `engager`, `regagne`, `rengage`. On note aussi que ces mots sont tous de même longueur.

Pour cet exercice, on pourra par exemple s'aider des méthodes suivantes (mais il n'est pas forcément nécessaire de toutes les utiliser pour cet exercice) :

```
String mot = "Engager";

// Transforme les majuscules en minuscules
mot = mot.toLowerCase(); // mot <- "engager"

// Retourne le caractere a l'indice donne
char lettre = mot.charAt(2); // lettre <- 'g'
String sLettre = String.valueOf(lettre); // sLettre <- "g"

// Cree un tableau de caracteres a partir des caracteres dans 'mot'
// lettres <- {'e', 'n', 'g', 'a', 'g', 'e', 'r'}
char[] lettres = mot.toCharArray();

// Trie un tableau par ordre croissant
Arrays.sort(lettres);

// Il n'y a pas de moyen 'simple' de supprimer une lettre d'une chaine
// de caracteres
// On peut arriver a un resultat identique en remplacant la premiere
// occurence de cette lettre par une chaine vide
//
// replaceFirst(String pattern, String replacement)
// pattern     : Chaine de caracteres a remplacer
// replacement : Chaine de caracteres de remplacement
mot = mot.replaceFirst(sLettre, ""); // mot <- "enager"
```

2.1.1 Fonction itérative

Questions :

1. Créez une fonction `public static boolean isAnagram(String s1, String s2)` qui prend en paramètre deux chaînes de caractères et qui retourne `true` si ce sont des anagrammes, `false` sinon.
2. Implémentez une version itérative de cette fonction (c'est-à-dire, une fonction qui n'est pas récursive).
3. Testez cette fonction avec les exemples donnés dans l'énoncé, ainsi qu'avec des exemples de mots qui ne sont pas des anagrammes.

2.1.2 Fonction récursive

Questions :

1. Créez une fonction `public static boolean isAnagramRec(String s1, String s2)` qui prend en paramètres deux chaînes de caractères et qui retourne `true` si ce sont des anagrammes, `false` sinon.
2. Implémentez une version récursive de cette fonction. On notera que créer un tableau trié de lettres pourra servir à la résolution de cet exercice.
3. Testez cette fonction avec les exemples utilisés pour la fonction itérative afin de vérifier si les résultats sont cohérents ou non.

2.2 Tri d'une liste

Dans cet exercice, nous allons trier une liste selon le principe suivant :
— La première partie de la liste triée doit contenir tous les éléments pairs de la liste, triés par ordre croissant ;
— La seconde partie de la liste triée doit contenir tous les éléments impairs de la liste, triés par ordre croissant.
Par exemple, soit la liste :

$$[4, -2, 31, -5, 12]$$

Cette liste doit être triée comme suit :

$$[-2, 4, 12, -5, 31]$$

Pour cet exercice, on pourra par exemple s'aider des méthodes suivantes (mais leur utilisation n'est pas forcément nécessaire) :

```
ArrayList<Integer> liste = new ArrayList<>();
// Code pour initialiser la liste a [4, -2, 31, -5, 12]...

// Trie la liste par ordre croissant
Collections.sort(liste);

// Retourne le minimum ou le maximum d'une liste
int min = Collections.min(liste); // min <- -5
int max = Collections.max(liste); // max <- 31

// Retourne l'indice de la premiere occurrence d'une valeur
int indice = liste.indexOf(-5); // indice <- 3

// Modifie la valeur d'un element a un indice donne
//
// set(int index, E e)
// index   : indice de l'element a modifier
// e       : element a inserer a l'indice fourni
liste.set(4, 10); // liste <- [4, -2, 31, -5, 10]

// Retourne une sous-liste d'une liste en tant qu'ArrayList
// On note que l'on force le type en un ArrayList (c'est le role de
//  l'element entre parentheses)
//
```

```java
// subList(int beginIndex, int endIndex)
// beginIndex  : premier indice a prendre (inclus)
// endIndex    : dernier indice a prendre (exclus)
//
// Ici, sousListe <- [4, -2, 31]
ArrayList<Integer> sousListe = (ArrayList<Integer>) liste.subList(0, 3);
```

Questions :

1. Créez une fonction `public static void triListe(ArrayList<Integer> liste)` qui prend en paramètre une liste d'entiers et la trie selon le principe énoncé plus haut ;
2. Implémentez cette fonction puis testez-la avec différents exemples dont celui fourni dans l'énoncé. **Attention**, on souhaite trier la liste passée en paramètre. Vous ne devez donc pas créer d'autre liste dans cette fonction.

2.3 Réseau social

Dans cet exercice, nous cherchons à représenter un réseau social. Dans ce réseau social :

— Un utilisateur peut suivre plusieurs autres utilisateurs ;
— La relation entre utilisateurs n'est pas réciproque, c'est-à-dire qu'il est possible de suivre un utilisateur sans que ce-dernier nous suive.

Sur ce réseau social, il sera donc possible de :

— Ajouter un nouvel utilisateur ;
— Suivre des utilisateurs du réseau social ;
— Lister tous les utilisateurs du réseau social ;
— Pour un utilisateur donné, lister tous les utilisateurs suivis ;
— Pour un utilisateur donné, lister tous les utilisateurs qui le suivent ;
— Arrêter de suivre un utilisateur ;
— Supprimer un utilisateur du réseau social.

Note : Pour vous aider dans cet exercice, vous pourrez créer deux variables *globales* dans votre fichier `Main.java` en saisissant les lignes suivantes (attention à l'importation des classes `HashMap` et `Scanner` nécessaires pour utiliser ces variables) :

```java
public class Main {
    static HashMap<String, ArrayList<String>> reseau = new HashMap<>();
    static Scanner scanner = new Scanner(System.in);
}
```

2.3.1 Menu de l'exercice

Écrire une fonction `public static int menu()` permettant à l'utilisateur de décider de l'action qu'il souhaite réaliser. Un exemple d'affichage est donné ci-après :

```
Bienvenue dans votre réseau social !
Tapez 1 pour ajouter un utilisateur au réseau social.
Tapez 2 pour suivre un autre utilisateur.
Tapez 3 pour lister tous les utilisateurs du réseau social.
```

```
Tapez 4 pour lister tous les utilisateurs suivis par un utilisateur donné.
Tapez 5 pour lister tous les utilisateurs qui suivent un utilisateur donné.
Tapez 6 pour arrêter de suivre un utilisateur pour un utilisateur donné.
Tapez 7 pour supprimer un utilisateur du réseau social.
Tapez 0 pour quitter.
```

Note : Comme expliqué lors du TP1, il faut faire attention avec le `Scanner` lorsque l'on enchaîne les méthodes `nextInt()` et `nextLine()`. On vous proposera donc d'utiliser plutôt l'instruction suivante :

```
int saisie = Integer.parseInt(scanner.nextLine());
```

2.3.2 Ajouter un utilisateur au réseau social

Écrire une fonction `public static void ajoutUtilisateur()` qui :

1. Demande la saisie du nom d'un utilisateur ;
2. Si cet utilisateur est déjà présent sur le réseau social, affiche un message d'erreur ;
3. Si cet utilisateur n'est pas déjà présent, l'ajoute au réseau social.

2.3.3 Suivre un autre utilisateur

Écrire une fonction `public static void suivreUtilisateur()` qui :

1. Demande la saisie du nom d'un premier utilisateur ;
2. Demande la saisie du nom d'un second utilisateur ;
3. Si les deux utilisateurs sont inscrits sur le réseau social :
   — Si le premier utilisateur suit déjà le second utilisateur, affiche un message d'erreur ;
   — Sinon, crée cette relation de suivi entre le premier utilisateur et le second utilisateur.
4. Sinon, inscrit les deux utilisateurs et crée la relation de suivi décrite juste au-dessus.

2.3.4 Lister tous les utilisateurs du réseau social

Écrire une fonction `public static void listerUtilisateurs()` qui affiche la liste de tous les utilisateurs du réseau social, classés par ordre alphabétique.

2.3.5 Lister tous les utilisateurs suivis

Écrire une fonction `public static void listerUtilisateursSuivis()` qui :

1. Demande la saisie du nom d'un utilisateur ;
2. Si cet utilisateur n'est pas sur le réseau social, affiche un message d'erreur ;
3. Si cet utilisateur est sur le réseau social, affiche la liste des utilisateurs qu'il suit, classés par ordre alphabétique.

2.3.6 Lister tous mes suiveurs

Écrire une fonction `public static void listerUtilisateursSuiveurs()` qui :

1. Demande la saisie du nom d'un utilisateur ;
2. Si cet utilisateur n'est pas sur le réseau social, affiche un message d'erreur ;
3. Si cet utilisateur est sur le réseau social, affiche la liste de tous les autres utilisateurs qui le suivent, classés par ordre alphabétique.

2.3.7 Arrêter de suivre un utilisateur

Écrire une fonction `public static void arreterSuivi()` qui :

1. Demande la saisie du nom d'un premier utilisateur ;
2. Demande la saisie du nom d'un second utilisateur ;
3. Si le premier utilisateur ne suit pas le second utilisateur, affiche un message d'erreur ;
4. Sinon, supprime la relation de suivi entre le premier utilisateur et le second utilisateur.

2.3.8 Supprimer un utilisateur

Écrire une fonction `public static void supprimerUtilisateur()` qui :

1. Demande la saisie du nom d'un utilisateur ;
2. Si cet utilisateur est inscrit sur le réseau social, le supprime du réseau ;
3. Sinon, affiche un message d'erreur.