TP 6 — les collections

Objectif(s)

- ✯ comprendre les différences entre les collections
- ✯ utiliser des collections

Exercice 1 – Comparaison des collections

Question 1

Écrivez une classe `TesteCollections` ayant une méthode `main()`. Dans cette méthode

1. Créez une `ArrayList` contenant des `Doubles`. Utilisez pour la suite la JavaDoc (`https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`) pour identifier les bonnes méthodes à utiliser.
2. Remplissez-la en insérant 100.000 valeurs aléatoires (classe `java.util.Random`, méthode `nextDouble()`) **au début** de la liste.
3. Mesurez le temps d'exécution en utilisant `System.currentTimeMillis()`) et affichez-le.
4. Videz la liste (méthode `clear()`) et répétez le remplissage, mais en ajoutant les valeurs **à la fin**.
5. Comparez les deux temps.
6. Accédez 10.000 fois à des indices aléatoires (`nextInt()`) et mesurez le temps nécessaire.
7. Recherchez 10.000 fois des valeurs aléatoires dans la liste (`contains()`) et mesurez le temps d'exécution.
8. Parcourez la `ArrayList` en utilisant une boucle `for` et des indices explicites.
9. Utilisez un itérateur (méthode `iterator()`) et un itérateur de liste (méthode `listIterator()`) pour faire la même opération et comparez les temps d'exécution pour les trois méthodes (questions 8 et 9).
10. Parcourez la `ArrayList` en utilisant une boucle `for` et des indices explicites pour afficher les 10 premiers valeurs.
11. Utilisez un itérateur (méthode `iterator()`) et un itérateur de liste (méthode `listIterator()`) pour faire la même opération et comparez l'affichage

Répétez le précédent pour

— Une `LinkedList`, JavaDoc à `https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html`,

— un `HashSet`, JavaDoc à `https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html` – **attention :** les valeurs ne doivent pas se répéter – et

— un `HashMap<Double,Integer>`, JavaDoc à `https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html` – il faut créer aléatoirement à la fois les clés et les valeurs, mais les clés ne doivent pas se répéter.

Attention : Toutes les opérations ne sont pas disponibles pour toutes les collections !

Il y a au moins deux options pour créer un itérateur d'un HashMap : récupérer l'ensemble de clés (`keySet()`) ou l'ensemble de valeurs (`values()`) et utiliser sa méthode `iterator()`. Dans le premier cas, il faut utiliser chaque clé avec la méthode `get()`.

Étant donné les différents temps d'exécution que vous avez observé, proposez un type d'application adapté à chacune de différentes collections.

Exercice 2

Sur eCampus, vous trouvez un fichier `matches-w-more-factors.csv` qui contient des données des matchs de la NBA.

Chaque ligne contient :

— Les noms de deux équipes, la première équipe joue à l'exterieur, la deuxième à domicile (dont le "at" entre les deux noms),

— La date et l'année,

— Si c'était un match pendant la saison régulière ou pendant la série éliminatoire — vous pouvez l'ignorer,

— Le nombre de points marqué par chaque équipe (séparés par un "-"),

— La location que vous pouvez ignorer,

— Des statistiques pour chaques équipe : le champ avec le suffixe "1" sont des statistiques de la première équipe, suffixe "2" pour la deuxième.

Question 1

Vous trouvez aussi deux classes :

— `Match`, qui défini des matchs

— `CSVParser`, qui vous devez compléter.

Complétez la méthode `parseSaison(String)` pour qu'elle rétourne une `HashMap<String, LinkedList<Match>>`.

Les clés de cette `HashMap` sont les dates, la `LinkedList` contient pour chaque date tous les `Matchs` qui avaient lieu.

Pour que ça fonctionne bien, il faut ignorer les ligne qui commence par "opponents" ou "%". La méthode `startsWith(String)` de la classe `String` peut être utilisée pour le faire.

Pensez à utiliser la méthode `split(String)` de la classe `String` pour diviser les lignes lues.

Question 2

Affichez pour chaque date tous les matchs.

Il n'est pas nécessaire d'afficher les dates dans l'ordre, mais l'affichage devrait être $\pm$ lisible, donc pas tous les matchs sur la même ligne.

Pour récupérer les dates, vous pouvez utiliser la méthode `keySet()` de la classe `HashMap`.

Question 3

Complétez ensuite la méthode `parseEquipes(String)` qui retourne une `HashMap<String, ArrayList<HashMap<String, Double>>>`.

Dans cette `HashMap`, les clés sont les noms des équipes. Chaque élémént de l'`ArrayList` est encore une `HashMap` dont les clés sont les noms des champs qui vous trouvez dans la première ligne du fichier (celle qui commence avec "opponents"), ainsi que le "score", les valeurs sont les statistiques d'une équipe pour un match.

Attention : chaque ligne contient les statistiques de deux équipes, les champs de la première équipe finissent avec "1" ("2" pour la deuxième équipe).

La méthode `endsWith(String)` de la classe `String` peut s'avérer utile, ainsi que la méthode `replace(CharSequence, CharSequence)`.

Question 4

Pour chaque équipe, affichez les statistiques de tous ses matchs.

Question 5

Pour chaque équipe, calculez son taux de victoire, c.-à-d. le pourcentage de matchs qu'elle a gagnés.

Affichez les 16 équipes ayant les meilleurs taux de victoire.

Question 6

Pour chaque équipe, calculez et affichez la moyenne de aux moins trois statistiques, dont une doit être l'`OEff` (l'efficacité en attaque).