TP n°3 sur les collections (Tableaux associatifs)

S. Rosmorduc, V. Aponte

Septembre 2022

Exercice 1

Préliminaires : on reprend le code de la méthode `charsIn(String)`, Tp n°1 sur les collections

Dans la question 2, exercice 2, Tp n°1 sur les collections on vous demandait d'écrire une méthode `charsIn(String)` respectant les consignes suivantes :

Extrait du Tp n°1, exercice 2, question 2 : *Ecrivez une méthode statique `charsIn(String s)` qui renvoie une collection contenant les caractères différents qui apparaissent dans la chaîne s. Quel type de collection utilisez vous ?*

Si vous ne l'avez pas traitée lors du Tp n°1, c'est le moment de le faire. Sinon récupérez le code de cette méthode, que vous allez maintenant modifier en suivant les consignes données plus bas.

Question 1

La méthode `charsIn(String s)` peut être appelée plusieurs fois sur la même chaîne, et dans ce cas, on voudrait éviter de refaire le calcul de l'ensemble de caractères apparaissant dedans. Utilisez un *cache* chargé d'enregistrer les résultats pour tous les mots déjà traités. **Vous devez l'implanter** à l'aide d'un `SortedMap` suivant l'ordre alphabétique des mots se trouvant dedans. Ce cache fera partie d'une nouvelle classe `OccurChaine`, où vous mettrez également la méthode `charsIn(String s)`. La méthode `charsIn(String s)` doit utiliser le cache : comment le fait-elle ? Cette méthode peut-elle continuer à être statique ? Justifiez votre réponse. Ajoutez dans cette classe une méthode `afficheCacheAlpha()` permettant d'afficher le contenu du cache par l'ordre alphabétique des mots qu'il contient. Ajoutez une méthode `afficheCachePoids()` qui affiche tous les mots du cache accompagnés du nombre de caractères différents que chacun possède. Cette méthode affichera les mots dans l'orde décroissant du nombre de caractères différents qu'ils possèdent : celui qui possède le plus de caractères différents doit être affiché en premier. Par ailleurs, si deux mots m_1 et m_2 ont le même nombre de caractères différents, le plus petit dans l'ordre alphabétique sera affiché en premier. Utilisez impérativement une classe qui implante `Comparator` dans la méthode `afficheCachePoids()`.

Question 2

Ajoutez quelques tests pour la méthode `charsIn`. Vous devrez tester : (1) que son comportément est correcte, (2) que le cache est correctement mis à jour lors de l'ajout d'une chaîne non présente, et qu'il ne change pas si la chaîne est déjà dedans. Comment mettre en place les tests autour de la modification du cache sans compromettre l'encapsulation de celui-ci ?

Exercice 2 : Repértoire téléphonique

On souhaite implanter un repértoire de contacts téléphoniques. Un contact téléphonique est caractérisé par son nom, son prénom, et une collection de numéros de téléphone. Un numéro de téléphone est quant à lui caractérisé par son code, le numéro lui-même et par un label parmi *domicile*, *mobile*, *pro*. Ce label peut être implanté par un type

énuméré, par exemple. Vous aurez à écrire une classe `Repertoire` **implantée par un attribut de type** `SortedMap` de tous les contacts selon l'ordre alphabétique de leurs noms et prénoms.

Question 1 : choix de la représentation

Enumérez toutes les classes que vous devez définir. Quels sont leurs attributs ? Quel type de collection utilisez vous pour repésenter la collection de numéros d'un contact ? Il y a au moins deux manières de modéliser l'association entre contacts et leurs numéros de téléphone, tout en respectant la contrainte d'utilisation d'une `SortedMap` :

1. Un objet `Contact` possède le nom, prénom et une collection de numéros sans numéros en double. Le repértoire est représenté par le type `SortedMap<String, Contact>`, où la clé d'un contact est la concaténation de son nom et prénom.
2. Un objet `Contact` ne contien pas ses numéros, mais seulement son nom et prénom. Le repértoire est représenté par le type `SortedMap<Contact, Collection<NumTel> >`, où `Collection<NumTel>` est la collection de numéros de l'objet `Contact` qui se trouve en position de clé.
3. Il est judicieux de remplacer ici, `Collection<NumTel>` par l'implantation de collection qui vous semble la plus pertinente et qui vous permettra de garantir l'absence de doublons (une liste ? un set ?, ...).

Essayez de comparer ces deux représentations : dans quel cas faut-il redéfinir `equals` et `haschcode` et pour quelles classes ? Faut-il implanter une des interfaces pour la comparaison d'objets vues en cours de manière à assurer le bon fonctionnement du `SortedMap` ? Justifiez votre réponse et faîtes ensuite votre choix !

NB : le git du projet correspond plutôt à l'un des choix mais vous pouvez prendre l'autre solution en changeant les classes et la méthode principale.

Question 2 : écriture des classes

Ecrivez toutes les classes nécessaires à la représentation des contacts et de leurs numéros, puis écrivez la classe repértoire avec la représentation de votre choix.

Dans la classe `Contact`, ajoutez des méthodes pour chercher un numéro, pour en ajouter numéro, et pour en supprimer. Pour simplifier, on ne prendra pas en compte le code dans ces opérations.

Dans la classe `Repertoire` ajoutez une méthode pour ajouter un nouveau contact, une méthode pour ajouter un numéro d'un contact, et pour supprimmer un de ses numéros. Ajoutez y une méthode `toString()`. Ajoutez des méthodes pour chercher un nom et prénom de contact, pour afficher tous les numéros d'un nom et prénom de contact, et pour obtenir la liste de tous les contacts d'un nom donné, triée par ordre aphabétique du nom puis des prénoms. Certains numéros sont partagés par plusieurs contacts (même domicile ou lieu de travail). Ajoutez une méthode permettant d'obtenir la liste triée de tous les contacts possèdant un numéro de téléphone passé en paramètre. Pour simplifier, on ne tiendra pas compte dans cette question du code du numéro.