Série H : Collections, Maps

Table des matières

Série H : Collections, Maps....1
Exercice H.1: Collections & Maps....1
Exercice H.2: Spellchecker....2
Exercice H.3: Dictionnaire....2
Exercice H.4: HashMap/TreeMap – en pratique....2
Exercice H.5: hashing & HashMap – détails du fonctionnement....3
Exercice H.6: HashMap – réalisation....3

Exercice H.1: Collections & Maps

a) Etablir l'arbre d'héritage des interfaces suivants : **`Iterable`**, **`Collection`**, **`List`**, **`Map`**, **`Set`**, **`SortedMap`**, **`SortedSet`**, **`NavigableMap`**, **`NavigableSet`**.

b) Etablir l'arbre d'héritage des collections prédéfinies **`AbstractCollection`**, **`AbstractList`**, **`AbstractSet`**, **`TreeSet`**, **`ArrayList`**, **`LinkedList`**, **`AbstractSequentialList`**, **`Vector`**, **`Dictionary`**, **`Hashtable`**, **`AbstractMap`**, **`HashMap`**, **`TreeMap`**.

c) Qu'est-ce qu'une **`Collection`**? Quelles sont ses caractéristiques ?

d) Qu'est-ce qu'un **`Set`** ? Quelles sont ses caractéristiques (en comparant avec **`Collection`**) ?

e) Qu'est-ce qu'une **`List`**? Quelles sont ses caractéristiques ?

f) Qu'est-ce qu'une **`Map`** ? Quels sont ses caractéristiques et champs d'application ?

g) Qu'est-ce qu'une **`HashMap`** ? Quels sont ses champs d'application ?

h) Quelles sont les caractéristiques des interfaces **`SortedMap`** et **`SortedSet`** ?

i) Quelles sont les caractéristiques de **`TreeMap`** et **`TreeSet`** ?

j) Quelles sont les différences entre **`ArrayList`** et **`Vector`** ?

k) Quelles sont les différences entre **`HashMap`** et **`Hashtable`** ?

l) Que peut-on dire de **`Vector`** et **`Hashtable`** et de leur utilisation ?

m) Qu'est-ce que la **synchronisation** dans le contexte des collections ? Quand est-ce qu'elle joue un rôle ?

n) Comment peut-on **synchroniser** une collection qui ne l'est pas dès le départ ?

Exercice H.2: Spellchecker

Réaliser un correcteur d'orthographe pour des textes allemands en employant le fichier *"DE_Wortliste_UTF8.txt"*. Entrez les textes à contrôler dans un champ texte (**TextArea**). Les mots sont contrôlés un à un et le programme affiche les mots incorrects (qui ne se trouvent pas dans le fichier) dans un champ texte pour qu'on puisse les corriger.

Ajoutez une méthode **analyseText** qui fournit une liste de tous les mots qui ne se trouvent pas dans un texte donné. La liste contient aussi les positions auxquelles ces mots sont situés dans le texte original.

Utilisez différentes structures de données : **ArrayList** (prédéfinie), **LinkedList** (prédéfinie), **TreeSet** (prédéfini)**, BST** (développé dans le cours). Testez et comparez la performance des différentes structures, p.ex. en déterminant le temps que votre algorithme met pour analyser un texte allemand assez long.

Développez d'abord une structure de classes appropriée pour éviter la duplication de code et pour faciliter les tests des différentes structures de données (héritage, etc.).

Amélioration:

Créez des versions pour **ArrayList** et **LinkedList** qui utilisent la recherche dichotomique.

Comparez et expliquez les performances que vous constatez.

Exercice H.3: Dictionnaire

Développez un dictionnaire allemand-français (bi-directionnel) en employant

(a) la structure prédéfinie **HashMap**,

(b) un arbre de recherche défini par vous-même.

Le fichier *"DE-FR_dictionary_UTF8.txt"* vous est fourni.

Sur quels difficultés tombez-vous (pour chacune des deux structures), si vous voulez ajouter plusieurs traductions pour le même mot.

Exercice H.4: HashMap/TreeMap – en pratique

Développez une application qui sait mémoriser dans une **HashMap** les données (nom, prénom, adresse, classe, date de naissance) d'étudiants. Utilisez le code Untis comme clé.

L'application saura effectuer les opérations suivantes (fenêtre texte ou GUI selon vos préférences) :

1. saisir les données et ajouter un étudiant
2. afficher toutes les données de tous les étudiants (code Untis inclus)
3. rechercher un étudiant (par son code Untis)
4. supprimer un étudiant (trouvé par son code Untis)
5. rechercher et afficher tous les étudiants d'une classe

Essayez d'ajouter plusieurs fois le même étudiant / la même clé. Que remarquez-vous ?

Remplacez la structure **HashMap** par **TreeMap.** Quelles modifications faut-il faire ? Quelles différences voyez-vous dans votre programme ?

Exercice H.5: hashing & HashMap – détails du fonctionnement

Veuillez répondre aux questions suivantes :

- Quelle est l'avantage d'une **HashMap** par rapport à une structure comme **ArrayList** ou *Array* ? De quoi dépend l'efficacité d'une **HashMap** ?
- Quelles conditions doit remplir la classe utilisée comme clé ?
- Quel est le rôle de la méthode **hashcode** et quelle condition doit-elle remplir pour garantir la plus grande efficacité ?
- Que se passe-t-il exactement si deux clés ont le même **hashcode** ?
- Comment est-ce que Java retrouve une valeur si deux clés ont le même **hashcode** ?
- Que se passe-t-il si la **HashMap** devient trop petite pour le nombre de valeurs à mémoriser ?
- Quel est l'avantage d'une **ConcurrentHashMap** ?

Exercice H.6: HashMap – réalisation

Reprenez l'exercice H.4 - HashMap/TreeMap – en pratique et réalisez-le en définissant votre propre implémentation **GI_HashMap** d'une **HashMap**. Utilisez les méthodes **hashcode** et **equals** de la classe de la clé. Réalisez les méthodes de base d'une **HashMap** : **put**, **get**, **remove**, **size**, …

Consultez la description *JavaDoc* de **HashMap** pour les détails de la réalisation des différentes méthodes.