TP – Table d'index
Avec la classe HashMap

Principe

Le but de ce TP est de développer une classe pour représenter un index de mots. L'idée est la suivante. A partir d'un document source, on souhaitera constituer un index contenant les mots jugés essentiels et présents dans le document. Chaque entrée de l'index contiendra le mot et une liste de numéros de pages, les pages dans lesquelles le mot apparaît. Cette liste ne devra pas contenir de valeurs en double. De même, l'index ne pourra pas contenir plusieurs fois le même mot.

On fera la distinction entre les minuscules et les majuscules. On ne vérifiera pas lors de l'ajout d'un mot à l'index que ce mot est valide. Toute chaîne de caractères non vide sera considérée comme un mot. Les numéros de pages seront des entiers strictement positifs.

Par exemple, un index pourrait être le suivant :

```
C++ [5, 10, 15, 20]
HTML [2, 4, 8]
Java [5, 10, 15, 20]
JavaScript [10]
PHP [1, 2, 3, 4, 5]
```

Les mots figurant dans l'index sont C++, HTML, Java, JavaScript et PHP. Les valeurs entre les crochets sont les numéros des pages sur lesquelles le mot apparaît, dans le document indexé.

Indications

- ✓ Un fichier avec les méthodes de tests unitaires est disponible: *TestIndexMot.java*
- ✓ Les numéros de pages pourront être stockés dans une ***ArrayList.***
  Remarque : Un ensemble peut sembler plus adapté car il permettrait de gérer plus facilement l'absence de double parmi les numéros de pages. Toutefois, le parcours des numéros est plus simple si ceux-ci sont stockés dans une ***ArrayList***)
- ✓ A plusieurs reprises, il sera nécessaire de trier cette liste dans l'ordre croissant. Pour ce faire, il faut utiliser la méthode ***sort*** de la classe ***Collections***.
- ✓ La table d'index sera de type ***HashMap***.

Enoncé

La classe ***IndexMot*** proposera les services suivants :

- ✓ un **constructeur sans argument** permettant de créer un index vide

- ✓ une méthode ***ajouterMot*** permettant **d'ajouter une paire (mot, numéro de page) à l'index**. Ces deux informations seront données en paramètre.

  - ⇨ Plus précisément, si ce mot est déjà présent dans l'index, l'ajout se réduira à l'insertion d'un numéro de page supplémentaire pour ce mot, si bien sûr ce numéro n'est pas déjà présent dans la liste.
  - ⇨ Si le mot n'est pas encore présent dans l'index, il faudra créer une nouvelle entrée dans la table pour ce mot, et ce numéro de page.
  - ⇨ Si le numéro de page est négatif ou nul, l'ajout ne se fera pas.
  - ⇨ La méthode renverra un booléen pour indiquer si l'ajout a été possible.

- ✓ une méthode ***getListeNuméro*** ayant en paramètre un mot et qui renverra **la liste des numéros de page associée à ce mot**. La méthode renverra ***null***, si le mot n'est pas présent dans l'index.

- ✓ une méthode ***getListeMot*** qui renverra une **chaîne** de caractères contenant **tous les mots présents dans l'index**, et classés dans l'ordre alphabétique. Par exemple, cette chaîne pourrait être `"C++ HTML Java JavaScript PHP"`

- ✓ une méthode ***toString*** qui renverra sous la forme d'une chaîne de caractères le contenu de l'index. Les mots devront apparaître dans l'ordre alphabétique, et les numéros de pages seront classés dans l''odre croissant. Exemple de résultat :

```
C++ [5, 10, 15, 20]
HTML [2, 4, 8]
Java [5, 10, 15, 20]
JavaScript [10]
PHP [1, 2, 3, 4, 5]
```

- ✓ une méthode ayant en paramètre un mot et un numéro de page, et qui **supprimera l'occurrence de ce mot sur cette page**, si c'est possible. Si ce numéro de page est l'unique apparition du mot dans le document, alors le mot lui-même devra être supprimé de l'index. La méthode renverra un booléen pour indiquer si la suppression a pu être effectuée ou pas.

- ✓ une méthode permettant de prendre en compte **l'insertion d'une page** (voir partie 2)

- ✓ une méthode permettant de prendre en compte **la suppression d'une page** (voir partie 2)

Partie 1

1) Commencer à coder la classe ***IndexMot*** en déclarant l'attribut et en codant le constructeur et la méthode ***ajouterMot***.
Tester cette méthode.
Indication : les méthodes utiles de la classe ***HashMap*** sont ***get*** et ***put***.

2) Ajouter à la classe la méthode ***getListeNuméro***. Cette méthode se réduit à une ligne de code.
Tester cette méthode.

3) Ajouter à la classe la méthode ***getListeMot :***
- ✓ la méthode ***keySet*** appliquée à la table renvoie un ensemble contenant toutes les clés
- ✓ cet ensemble peut être transformé en ***ArrayList***
- ✓ cette liste peut ensuite être triée

Tester la méthode obtenue.

4) Ajouter à la classe la méthode ***toString***.

Il sera nécessaire de transformer la table en une instance de ***TreeMap***. Cette transformation permet ensuite d'accéder aux clés dans l'ordre croissant.

L'instance ***TreeMap*** sera parcourue entrée par entrée. Chaque entrée est de type ***Map.Entry*** (voir l'exemple du parcours de l'annuaire dans le support de cours).

Tester la méthode obtenue.

5) Ajouter à la classe la méthode ***supprimerNumero.***

Tester la méthode obtenue.

Partie 2

En principe, l'index associé à un document sera généré une fois ce dernier finalisé. Toutefois, on peut supposer que le document subisse au fil du temps quelques mises à jour. Une idée intéressante serait alors, dans la mesure du possible, de ne pas générer entièrement un nouvel l'index. En effet, si le document est long, cette opération sera coûteuse en temps.

Voici deux exemples de modifications qui pourraient être gérées sans pour autant nécessiter la génération complète d'un nouvel index : insertion d'une page dans le document, suppression d'une page.

Concrètement, supposons que l'on souhaite insérer une nouvelle page juste après la page 10 du document (donc entre les anciennes pages 10 et 11). Pour mettre à jour l'index, il suffira de parcourir toutes les entrées de la table, et pour chacune d'elles de modifier tous les numéros de page strictement supérieurs à 10 de manière à leur ajouter 1.

Supposons maintenant que l'on souhaite supprimer la page numéro 10 du document. Pour mettre à jour l'index, il faudra parcourir toutes les entrées de la table, et pour chacune d'elles il sera nécessaire de supprimer le numéro de page 10, et décrémenter de 1 les numéros de pages strictement supérieurs à 10. De plus, si la liste des numéros de page associée à un mot devient vide, il faudra supprimer ce mot de l'index.

Questions

1) Ajouter à la classe la méthode ***incrementerApres***. Cette méthode aura en paramètre un numéro de page. Son rôle consistera à ajouter 1 aux numéros de page strictement supérieurs à l'argument.

Pour effectuer le parcours de la table, on pourra passer par une vue de type ***Set*** (en appliquant la méthode ***entrySet()*** à la table). Les entrées de cette vue seront de type ***Map.Entry*** et seront parcourues une à une (voir l'exemple du parcours de l'annuaire dans le support de cours).

Tester la méthode obtenue.

2) Ajouter à la classe la méthode ***supprimerDecrementerApres***. Cette méthode aura en paramètre un numéro de page à supprimer.

Tout comme dans la méthode précédente, il faudra faire appel à une vue de la table de type ***Set.*** Cette vue sera ensuite parcourue.

Tester la méthode obtenue.