ISET de Sousse

Atelier Programmation Objet Avancée

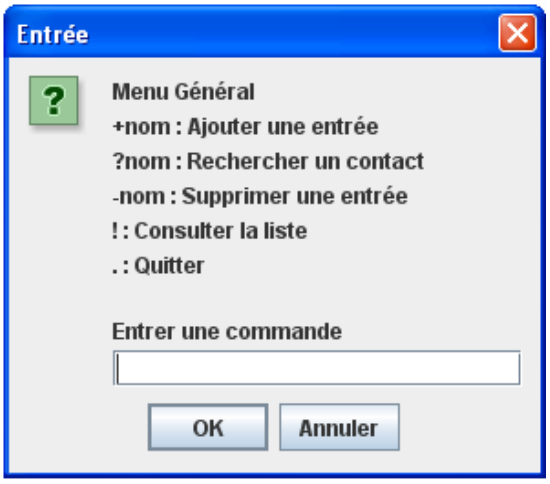
Les collections de type « Map » (Tableaux associatifs)

Enoncé

On veut écrire un programme en Java permettant de gérer un annuaire qui contient un ensemble de contacts. Pour chaque personne dont on cannait le nom, on crée une fiche comportant le numéro de téléphone et l'adresse.

1. Créer une classe « Fiche » qui contient deux attributs (nom, adresse), un constructeur et une redéfinition de la méthode toString().

2. Créer une classe « Annuaire » réduite à une méthode main() qui crée une table associative « annu » de type (HashMap<String, Fiche>)) puis propose de gérer cette table à travers le menu suivant :



   L'utilisateur doit à chaque fois entrer une commande dans l'un des formats suivants : ″+nom″, ″?nom″, ″-nom″, ″!″, ″.″.

   A titre d'exemple, la commande ″ **?Ali**″ affiche la fiche complète de la personne qui s'appelle « Ali » (s'il existe dans l'annuaire).

3. On constate que la commande ″**!**″ produit l'affichage des fiches dans un ordre imprévisible. Modifiez le programme précédent pour que les fiches apparaissent toujours dans l'ordre alphabétique croissant des noms.

Rappel

Une collection de type `Map` fonctionne avec un couple (clé, valeur). On y trouve les objets `Hashtable`, `HashMap`, `TreeMap`, etc. La clé, qui sert à identifier une entrée dans notre collection, est unique. La valeur, au contraire, peut être associée à plusieurs clés.

- **L'objet Hashtable (table de hachage)**

Cet objet nous offre tout un panel de méthodes utiles :

- `isEmpty()` retourne « vrai » si la colletcion est vide ;
- `put(Object key, Object value)` ajoute le couple (`key, value`) dans la collection ;
- `get(Object key)` retourne l'objet qui correspond à la clé indiquée ;
- `remove(Object key)` supprime une entrée de la table ;
- `clear()` vide la table ;
- `contains(Object value)` retourne « vrai » si la valeur est présente dans la table. Cette méthode est identique à `containsValue(Object value)` ;
- `containsKey(Object key)` retourne « vrai » si la clé passée en paramètre est présente dans table ;
- `elements()` retourne une énumération des éléments de l'objet ;
- `keys()` retourne la liste des clés sous forme d'énumération.

De plus, il faut savoir qu'un objet `Hashtable` n'accepte pas la valeur `null` et qu'il est `Thread Safe`, c'est-à-dire qu'il est utilisable dans plusieurs threads (cela signifie que plusieurs éléments du programme peuvent l'utiliser simultanément sans qu'il y ait un risque de conflit de données).

Exemple

```
import java.util.Hashtable;
public class HTable {
   public static void main(String[] args) {
      Hashtable<Integer,String> ht = new Hashtable<>();
      ht.put(1, "Said");
      ht.put(2,"Tarek");
      ht.put(2,"Khaled");
      Enumeration e = ht.elements();
      while(e.hasMoreElements())
            System.out.print(e.nextElement()+"\t");
   }
}
```

Output : `{4=Said, 2=Khaled, 1=Said}`

- **Remarques**

1. L'objet `HashMap` ne diffère que très peu de la `Hashtable` :

- il accepte la valeur `null` ;
- il n'est pas `Thread Safe`.

2. Dans un TreeMap, les éléments sont toujours triés dans l'ordre croissant des clés.