TD14

Ensembles - suite

Au cours du TD/TP précédent, nous avons créé un ensemble en utilisant l'idée suivante :

La classe contient un attribut de type `List<T>`, *gigantesque* et quasiment vide (i.e. contenant principalement des cases à `null`). Quelques cases ne sont pas `null`, ce sont les éléments de notre ensemble.

Pour déterminer la position d'un élément dans notre tableau, on utilise son hashCode : l'élément de hashCode `h` sera en position `h` (modulo la taille du tableau).

Voici la classe obtenue :

1. Complétez le code.
2. Testez le code avec un exécutable créant les ensembles `{1, 56, 3}` et `{"bonjour", "tout", "le", "monde"}`.
3. Que se passe-t-il si l'on a deux objets différents qui ont le même hashcode modulo 10000 ? Trouvez deux tels objets et essayez de les ajouter à un même ensemble.

Ensembles - Version finale

Au lieu de stocker nos éléments dans une `List<T>`, on va les stocker dans une `List<List<T>>`. La plupart des listes seront vides, sauf certaines qui la plupart du temps ne contiendront qu'un seul élément mais parfois plusieurs si ces éléments ont le même hascode modulo la taille de la liste de listes.

Pour bien fixer les idées, on va dessiner les choses dans un cas "simplifié" où la liste de listes, au lieu d'avoir une taille 10000, a une taille 10. On suppose que le hashCode de `"Bonjour"` est `12332`, que celui de `"tout"` est `756341`, celui de `"le"` est `745742` et celui de `"monde"` est `245234235`.

1. Dessinez le contenu de la liste si l'ensemble contient `"Bonjour"`, `"tout"`, `"le"` et `"monde"`.
2. Comment savoir si un élément appartient à l'ensemble ?
3. Quelle est la complexité selon vous de l'ajout d'un élément à l'ensemble ? Et de la méthode contains ?
4. Complétez le code suivant :

```java
import java.util.AbstractSet;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;

class EnsembleOk<T> extends AbstractSet<T> {
    private  List<List<T>> listeInterne;
    private int nbElements ;

    public EnsembleOk() {
        this.nbElements = 0;
        listeInterne =                                  // Complétez
        for(int i = 0; i < 10000; ++i)
            listeInterne.add(                           // Complétez

    }

    public int size(){
        return this.nbElements;
    }

    public Iterator<T> iterator(){
        return new Iterateur<>(this.listeInterne);  // On verra après !
    }

    public boolean add(T elem){


                                                    // Complétez









    }

    @Override
    public boolean contains(Object o){


                                                //  Complétez

    }
}
```

5. Ce qui reste à faire est de définir l'itérateur sur notre structure. Pour faire ça, il faut suivre deux choses : la liste actuellement pointée, et dans cette liste, l'élément actuellement pointé. Complétez le code suivant :

```java
import java.util.AbstractSet;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;



class Iterateur<T> implements Iterator<T>{
    private List<List<T>> valeurs;
    private Iterator<T> iterActuel;
    // Iterateur pointant vers l'élément actuel
    private Iterator<List<T>> iterListes;
    // Iterateur pointant vers la liste actuelle

    Iterateur(List<List<T>> liste)
    {
        this.valeurs = liste;
        this.iterListes = this.valeurs.iterator();
        this.iterActuel = this.iterListes.next().iterator();
    }

    @Override
    public T next()
    {




    }

    @Override
    public boolean hasNext()
    {



    }
}
```

Map

1. Proposez, sur papier, une manière d'implémenter une Map.

Avancé (à faire chez soi pour le plaisir)

1. En utilisant la classe AbstractMap, implémentez une Map non modifiable contenant les entrées données sous forme d'une liste d'EntrySet au constructeur de votre Map.
2. Proposez, sur papier, une manière d'implémenter un Map.