TP 9 : Les Collections en Java (II)

christina.boura@prism.uvsq.fr, stephane.lopes@prism.uvsq.fr

3 avril 2015

1 La classe `BitSet`

La classe `BitSet` implémente un vecteur de bits dont la taille peut être augmentée en fonction du besoin. Chaque élément de cet ensemble de bits peut être soit vrai (`true`) soit faux (`false`). Les bits de `BitSet` sont indexés avec des entiers non-négatifs. Par exemple, un vecteur `(false, false, true, false, true, false, false, true)` dont le bit de poids faible se trouve à gauche, est représenté comme :

`[2,4,7]`

Un objet de type `BitSet` peut être utilisé pour modifier le contenu d'un autre objet de type `BitSet` à l'aide des opérations `AND` (`et` logique), `OR` (`ou` logique) ou `XOR` (`ou exclusive`).

Exemple : Soient `bitSet1 = [2,4,7]` et `bitSet2 = [4,9]`. Alors

`bitSet1 OR bitSet2 = [2,4,7,9]`

`bitSet1 AND bitSet2 = [4]`

`bitSet1 XOR bitSet2 = [2,7,9]`.

2 Un ensemble creux de bits

Un ensemble *creux* de bits est un grand ensemble de valeurs booléennes dont la plupart des valeurs sont égales à `false`. Afin de representer ces ensembles creux, la classe `BitSet` peut être très inefficace. Comme la majorité des bits ont une valeur `false`, beaucoup d'espace est utilisé sans raison. Nous proposons ici de créer une representation alternative d'un ensemble de bits, qui serait beaucoup plus efficace en termes d'espace mémoire utilisé. Pour cela nous allons utiliser une table de hachage (`HashSet`). Seulement les positions qui ont une valeur `true` seront alors sauvegardés en mémoire.

On va créer alors une classe `BitSetCreux` qui va hériter de la classe `BitSet` et on va redéfinir les méthodes nécessaires. Pour cela téléchargez la structure de la classe `BitSetCreux` et complétez les méthodes `clear, get, length, set, size, and, andNot, or` et `xor`.

1. Pour les méthodes `clear, get, set` et `size` pensez à utiliser les méthodes de l'interface `Set`.
2. La méthode `length` renvoie la taille *logique* de l'ensemble qui est définie comme le plus grand élément de l'ensemble +1. Si par exemple le plus grand bit égal à `true` est le bit 127, cette méthode doit renvoyer 128, puisque on commence à compter à partir du bit 0. Vous pouvez ici utiliser la méthode statique `max` de la classe `Collections`.
3. Les méthodes `and, andNot, or` et `xor` doivent effectuer les opérations logiques correspondantes entre l'ensemble de bits sur lequel on applique la méthode et l'ensemble passé en paramètre.

2.1 Testez votre implémentation

Téléchargez le fichier `TesterClasse.java` et testez votre implémentation. L'execution doit être identique à ça :

```
Ensemble vide:
Longueur: 0
Taille: 0
Ensemble rempli:
Elements: [64, 70, 103]
Longueur: 104
Taille: 3
64 est-il présent ? :
[64, 70, 103] true
[70, 103] false
Les deux ensembles sont-ils egaux ? :
Ensemble 1: [70, 103]
Ensemble 2: [70, 104, 78]
Egaux ? false
OU :
[70, 103, 104, 78]
XOR:
[103, 104, 78]
AND:
[70]
```

3 Travailler avec des listes chaînées

Le but de cet exercice (très facile) est de vous aider à se familiariser un peu plus avec les listes chaînnées `LinkedList` et notamment pour voir comment travailler avec des objets `ListIterator` qui permettent de parcourir une liste dans les deux senses et de modifier ses éléments. Pour cela, téléchargez le fichier `ListeCouleurs.java` et complétez les méthodes suivantes :

- Le constructeur qui prend en entrée un tableau de chaînes de caractères et ajoute ses éléments à la liste.
- `public void afficherListe()` qui affiche les éléments de la liste à l'écran.
- `public void convertirMajuscules()` qui convertit tous les caractères de la chaîne en caractères majuscules.
- `public supprimerElements(int debut, int fin)` qui supprime de la liste les éléments dont l'index est compris entre `debut` et `fin`.
- `public afficherOrdreInverse()` qui affiche les éléments de la liste dans l'ordre inverse.

Vous pouvez tester votre implémentation avec le fichier `TesterCouleurs.java`.