Programmation Java - LI381

TP 02 – Héritage et premier exemple de polymorphisme

Julien Sopena

Octobre 2015

Ce deuxième TP introduit les notions d'héritage et de polymorphisme. Ainsi, il reprend l'implémentation de la liste chaînée en ajoutant une valeur dans chaque élément enregistré. Cette nouvelle structure permettra l'implémentation d'une table de hachage.

Attention : Ce TP nécessite d'avoir terminé le premier TP. Si tel n'est pas le cas, commencer par le terminer avant d'entamer ce nouveau travail.

Exercice 1 : Table de hachage

Le but de cet exercice est de réaliser une table de hachage : à chaque valeur (ici une chaîne de caractères) enregistrée on associe une clé (ici un entier) qui permettra d'enregistrer et de retrouver la valeur dans la table.

Notre table de hachage utilisera les mécanismes suivants :

— **Fonction de hachage :** $f(k) = k \ \% \ tailleTable$.. Chaque clé sera enregistrée dans la table à la case correspondant au résultat de cette fonction.

— **Résolution des collisions :** par chaînage. Les clés qui ont la même valeur de hachage (résultat de la fonction de hachage), seront enregistrées dans une liste simplement chaînée.

Question 1

On veut spécialiser la classe `Element` en lui ajoutant un attribut `value` de type `String`. Quels mécanismes du Java doit-on utiliser pour faire cette spécialisation ? Quelles méthodes devront être ajoutées et/ou corrigées.

Question 2

Implémentez une nouvelle classe `ElementWithValue` associant une clé à une valeur. Cette classe s'affichera de la façon suivante :

```
(5,"cinq")
```

Question 3

Faut-il modifier la classe `List` pour faire une liste de **ElementWithValue** ?

Question 4

Faites, si nécessaire, les modifications sur la classe liste puis affichez le contenu d'une liste après l'ajout des éléments suivants : `(5,"cinq")`, `(9,"neuf")` et `(3,"trois")`.

Question 5

On veut maintenant faire une classe **`HashTable`**, qui implémente une table de hachage en utilisant la liste précédemment implémentée pour résoudre les collisions. Donnez ses attributs et un constructeur qui prend en paramètre la taille de la table.

Question 6

Ajouter à la classe **`HashTable`** une méthode **`toString`** qui affiche l'état courant de la table de hachage de la manière suivante :

```
[0] : ->
[1] : -> (5,"cinq") -> (9,"neuf")
[2] : ->
[3] : -> (3,"trois")
```

Question 7

Ajoutez à la classe **`HashTable`** une méthode **`push`** qui ajoute l'élément, passé en argument, dans la table de hachage.

Question 8

Ajoutez à la classe **`HashTable`** une méthode **`valueOf`** qui retourne la valeur associée à une clé passée en paramètre (ou `"NOT RECORDING"` si la clé n'existe pas).

Question 9

Les performances d'une table de hachage dépendent beaucoup du nombre de collisions. Afin d'en limiter le nombre, on va s'assurer qu'elle ne contient pas plus de $\frac{tailleTale}{2}$ éléments. Pour cela, on implémente une méthode **`doubleHashTable`** qui double la taille de la table. Cette fonction sera alors appelée si le nombre d'éléments dépasse la limite.

