Algorithmique avancée - TD7

—o000o——o000o—

Implémentation de filter, map et reduce

1 Calcul de la liste des racines carrées

On considère dans cet exercice une liste contenant des éléments de type `Double` à partir de laquelle on souhaite construire une liste contenant les racines carrées des valeurs initiales. Il faut préalablement uniquement considérer les valeurs positives de la liste initiale.

1. Écrire la méthode `positive()` qui ne conserve dans la liste initiale que les éléments dont la valeur est supérieure ou égale à 0. Vous utiliserez un itérateur en précisant celui que vous choisissez ainsi que l'interface à laquelle vous l'associez pour obtenir le résultat escompté.

2. Proposer maintenant la méthode `racineCarree()` qui modifie la liste initiale en remplaçant les valeurs, pour lesquelles cela est possible, par leur racine carrée. Vous utiliserez encore un itérateur en précisant lequel et l'interface que vous lui associez.

3. Écrire enfin une méthode `nbSup10()` qui prend en entrée une liste de réels représentant des valeurs de racines carrées et retourne un entier indiquant combien de ses valeurs sont supérieures ou égales à 10.

2 Itérateur `filter`

On s'intéresse ici à définir des méthodes tirant profit de l'itérateur `filter` vu en cours. On considère ici une liste d'entiers qui étend `LList<Integer>`.

1. Écrire une méthode `moinsOccurrences(Integer e)` qui supprime toutes les occurrences de e dans la liste initiale.

2. Écrire une méthode `plusGrand(Integer seuil)` qui ne conserve que les éléments plus grand que le seuil passé en argument dans la liste

3 Itérateur `map`

On s'intéresse ici à définir des méthodes tirant profit de l'itérateur `map` vu en cours. On considère ici une liste d'association de type `Assoc` qui pour rappel étend le type liste de couples `LList<Couple>`. On suppose par la suite que la première valeur du couple est un nom sous forme de chaîne de caractères `String` et qu'une valeur réelle de type `Double` y est associée. Par exemple, la liste peut avoir le contenu suivant :

```
(("Riri", 13.5), ("Fifi", 14.5), ("Loulou", 17.0))
```

1. Écrire une méthode `listeNoms()` qui **retourne** la liste des noms des personnes présentes dans la liste d'association initiale.
2. Écrire une méthode `multiplication(Double e)` qui **modifie** la liste initiale et mulitplie toutes les valeurs de la liste initiale par e.

4 Itérateur `reduce`

On s'intéresse ici à définir des méthodes tirant profit de l'itérateur `reduce` vu en cours.

1. en considérant la liste d'associations précédente, écrire la méthode `produit()` qui retourne le produit des valeurs contenues dans la liste d'association précédente.
2. proposer une méthode `nomMin()` pour récupérer le nom associé à la valeur minimale de la liste.