TD-TP N°11– R2.01

Objectifs du TP

Implémenter de algorithmes de recherche et de tri
Lire et écrire des fichiers de texte
Utiliser la généricité

Retour sur les classes Pays et Population

Au TP précédent nous avons appris à manipuler des structures de données de type HashMap pour représenter des informations sur des pays.

Maintenant nous allons ajouter des fonctionnalités à la classe Population.

Exercice 1 : Calcul de la densité de population des pays et sauvegarde

La densité de population d'un pays c'est le nombre d'habitant au km2, donc on divise la population par la surface pour obtenir la densité.

Nous allons créer un nouveau HashMap qui contiendra les densités et on le sauvegardera dans un nouveau fichier. Vous pouvez si vous le souhaitez ajouter un nouveau HashMap en attribut d'instance pour conserver la densité mais ce n'est pas obligatoire.

- Ajouter une méthode à la classe **`Population`** qui calcule la densité pour chaque pays et retourne un HashMap contenant pour chaque clé (pays) la valeur de la densité :
    - **`public HashMap<String, Double> computeDensity()`**
- Ajouter une méthode à la classe `RWFile` qui permet d'écrire le HashMap dans un ficher. Je vous donne le code ci-dessous.
- Ecrire une classe **`DensityScenario`** pour vérifier vos méthodes : elle crée un objet Population, fait calculer les densités et sauve les données dans un fichier nommé « worldDensity.txt » qui sera mis dans le dossier « data » avec les autres fichiers. Ensuite vous regarderez si votre fichier est bien créé et si son contenu est correct.

```
/**
  * cette methode sert a ecrire un HashMap dans un fichier de texte :
  * la cle et la valeur sont ecrits sur une ligne
  * *@param map un HasMap contenant les donnees
  * @param fileName le nom du fichier a ecrire
  */
public static void writeMap (HashMap<String, Double> map, String fileName) {
   // ajouter les tests de paramètres
   try {
     // ouverture du fichier a ecrire
     PrintWriter out = new PrintWriter (fileName) ;
     for (String key : map.keySet())
       // concatene une chaine et un Double.toString()
       out.println (key + "     " + map.get(key).toString()) ;
     // fermeture du fichier
     out.close() ;

   } catch (FileNotFoundException e) {
     System.out.println ("Fichier non trouve : '" + fileName + "'") ;
} }
```

Exercice 2 : Tri des noms de pays

En utilisant des `HashMap` on a perdu l'ordre alphabétique des Pays.

Nous souhaitons trier notre liste listePays par ordre alphabétique des noms des Pays. Pour cela nous allons créer une classe **`TriParSelectionAlpha`**, sur le même modèle que la classe `TriParSelection` du TP8 (le code diffère seulement pour la comparaison).

La classe **`String`** implémente l'interface **`Comparable`** et possède donc une méthode :

```
public int compareTo(String anotherString)
```

qui retourne
– la valeur 0 si l'argument **`anotherString`** est égale à cette chaîne ;
– une valeur négative si la chaîne est lexicographiquement plus petite que l'argument **`anotherString`**
- et une valeur strictement positive si la chaîne est lexicographiquement plus grande que l'argument **`anotherString`**

`String` possède aussi une méthode qui ne tient pas compte des majuscule/minuscule :
`public int compareToIgnoreCase(String str)` : compare deux chaînes en ignorant la casse.

A faire :

- Réaliser le tri alphabétique de l'attribut **`listePays`** de la classe **`Population`** en adaptant la comparaison et en utilisant l'interface **`ITri`** qui se trouve dans le package *tri* .Mettez votre nouvelle classe **`TriParSelectionAlpha`** dans le package *tri.*
- Ecrire une classe TriAlphaScenario pour vérifier que votre tri fonctionne sur la liste listePays.
- **Déposer dans la zone de rendu une archive contenant les sources des exercices 1 et 2**

Exercice 3 : Rendre le tri générique

Les tris que nous avons fait jusqu'alors utilisant un tableau de Pays.
Il serait préférable d'avoir un tri générique sur un tableau d'un type quelconque d'objet.
Dans le cours 8 nous avons vu des exemples de classes paramétrées avec un type générique.

4.1 Ajouter la généricité :

Nous allons rendre notre tri alphanumérique générique.
Ce qui est particulier dans notre tri c'est l'utilisation de l'interface `Comparable` avec la méthode `compareTo`.
Nous avons implémenté cette interface pour la classe `Pays` et dans le tri on sait que la classe `String` implémente aussi cette interface.
Donc il faut que notre type générique T implémente aussi l'interface `Comparable`.

En Java la façon d'indiquer que l'on a une sorte de contrainte sur la type générique est d'utiliser :

```
<T extends TypeLimitant>
```

- qui indique que T et le type limitant peuvent être une classe ou une interface
- Le mot clé *extends* a été choisi car il constitue une approximation raisonnable du concept de sous-type et que les concepteurs Java ne souhaitaient pas ajouter un nouveau mot clé (comme sub).
Dans notre cas l'objet de type T devra implémenter `Comparable`, soit `Comparable<T>`.
Donc il faudra indiquer dans les entêtes de nos classes génériques utilisant l'interface Comparable :

```
<T extends Comparable<T>>
```

4.2 Utiliser des packages avec la généricité

Deuxième chose à prendre en compte : les packages.
Nous allons placer notre classe de tri en dehors des packages comme les classes scénario, car elle ne doit pas importer les types des tableaux.
Si on veut utiliser l'interface ITri qui se situe dans le package tri on va écrire :

```
public class TriParSelectionG <T extends Comparable<T>> implements tri.ITri
```

Maintenant si on veut trier les Pays, le type sera **`pays.Pays`** au moment de la création du tri.