Interfaces Comparator Comparable

Personne

Écrivez une classe Personne contenant un attribut nom et un attribut age.

1. Définir une classe **ListePersonnes** ayant un constructeur de la forme *ListePersonnes(List<Personne> liste)*
2. Définir une méthode permettant de trier une telle liste par âge croissant ?
3. Définir une méthode qui renvoie l'écart d'âge minimum entre deux personnes.
4. Testez toutes vos méthodes.

Complexes

On rappelle qu'un complexe s'écrit sous la forme *partie réelle* + *i partie imaginaire* On considère la classe suivante

```
public  class Complexe {
   private double reel;
   private double imaginaire;
   public Complexe(double reel, double imaginaire)
   {
       this.reel = reel;
       this.imaginaire = imaginaire;
   }
 }
```

1. Écrivez une classe implémentant l'interface **Comparator** et permettant de trier une List<Complexe> par partie réelle croissante.
2. Testez votre code dans un exécutable.
3. Définir une classe **ListeComplexes** possédant un constructeur de la forme ``ListeComplexes(List<Complexe> listeComplexes)`
4. Comment trier une telle liste par norme croissante ? (la norme d'un complexe est donnée par $reel^2 + imaginaire^2$)
5. Comment trier une telle liste par norme décroissante ?
6. Comment trouver le complexe de plus grande norme ?
7. Écrivez méthode prenant en paramètre un Comparator et un Complexe et renvoyant vrai si ce Complexe est plus petit que tous les complexes de la liste et faux sinon. Testez votre code avec les différents comparateurs écrits précédemment.
8. Ajoutez une méthode prenant en entrée un Comparator de complexes et deux complexes et renvoyant vrai si tous les complexes de la liste sont compris entre ces deux complexes. Testez votre code avec les différents comparateurs écrits précédemment.

Algorithmes sur les Listes d'entiers

1. Définir une classe **ListeEntiers** possédant un constructeur de la forme *public ListeEntiers(List<Integer> liste);*
2. Définir une méthode **occurrences** retournant la liste des occurrences d'une liste d'entiers.
3. Ajouter une méthode retournant un booléen indiquant si deux listes ont exactement les mêmes éléments.
4. Ajouter une méthode retournant une liste contenant la fusion (triée) de deux listes, sans utiliser la méthode *sort()*. Les deux listes sont supposées triées
5. Définir une méthode indiquant s'il existe deux éléments dont la somme est nulle

6. Définir une méthode permettant de trouver la valeur de la plus petite différence positive entre deux éléments. Par exemple dans la liste `[1, 7, 10, 8]`, la plus petite différence est 1, obtenue en faisant la différence entre 7 et 8.

Attention : votre code devra être testé sur de *grandes* listes. Un algorithme quadratique (c'est-à-dire une algorithme dont la complexité est en $O(N^2)$) ne sera pas validé !

Outils

Finalisez l'exercice sur les Outils du TD7.