TD8

Interface Comparable<T>

L'interface Comparable<T> permet de définir des objets 'naturellement comparables', elle est utilisée par la méthode sort de Collections :

```
public static <T extends Comparable<T>>  void sort(List<T extends Comparable<T>>)
```

Pour être `Comparable<T>` il faut implémenter :

```
public int compareTo(T)
```

- Écrivez une classe CoupleEntiers qui soit naturellement triable par premier élément croissant.
- Écrivez une classe CoupleEntiers qui soit naturellement triable par premier élément croissant et dans le cas d'égalité, compare par second élément.
- Écrivez une classe CoupleDouble qui soit naturellement triable par premier élément croissant. Pour cela utilisez la méthode Double.compare .

Classe générique

- Écrivez une classe générique `Couple<T extends Comparable<T>>` qui soit naturellement triable par premier élément croissant. *Remarque* faites attention au profil de la méthode compareTo.
- Remarquez qu'il suffirait pour la classe T d'être comparable avec n'importe lequel de ses ancêtres dans la hiérarchie, c'est-à-dire que si par exemple T étend une classe qui est Comparable< un des parent de T> alors T peut être comparé avec T. On peut alors modifier la classe pour avoir le profil : `Couple<T extends Comparable<? super T>>`

Intersection

Écrivez une fonction qui prend en entrée deux tableaux d'entiers triés et qui renvoie l'intersection des deux tableaux. Par exemple :

- sur l'entrée `[1,3,6,7,9,10], [3,5,6,8,11]` votre fonction doit renvoyer `[3,6]`.
- sur l'entrée `[0,10,12,13], [2,3,4]` votre fonction doit renvoyer `[]`.

Intersection ++

Même question que précédemment mais pour 3 tableaux.

Plus long préfixe commun

Écrivez une méthode qui prend en entrée une liste de chaines et renvoie le plus long préfixe commun à toutes les chaines de la liste.

Par exemple:

- sur l'entrée `["bonjour","bonsoir","bordure"]` le plus long préfixe commun est `"bo"`
- sur l'entrée `["bonjour","tout","le","monde"]` le plus long préfixe commun est `""` (chaine vide).

Élément unique

Supposons qu'on vous donne une liste d'entiers pour laquelle chaque valeur apparaît exactement deux fois excepté une valeur qui n'apparaît qu'une fois. Vous devez écrire un algorithme qui trouve cet élément.

Note: pouvez vous trouver un algorithme en temps O(n) ? (indice: utilisez le xor bit à bit, réalisé avec l'opérateur `^`)