TD1: parcours et tris simples de tableaux (génériques)

Avant de commencer, récupérez l'archive du TD sur moodle.

Classe TableauGenerique

Afin de pouvoir travailler sur des tableaux de n'importe quel type comme vous le faites pour les `ArrayList<>`, vous allez utiliser la classe *TableauGenerique*.

Cette classe est **générique** car elle permet de gérer des objets d'un type quelconque que l'on représente par `K` dans le code. La signature:

`TableauGenerique<K extends Comparable<K>>`

signifie que la classe `TableauGenerique` est définie sur des éléments de type `K` qui héritent d'une classe qui implémente `Comparable<K>`. `Comparable` est une interface, mais quand on définit une classe générique on écrit `extends` au lieu de `implements`. Cela est plus général et signifie "une classe dont un ancêtre (ou elle-même) implémente `Comparable<K>`".

Les attributs et méthodes de la classe *TableauGenerique* sont définis en utilisant le type `K`.

Nota: on pourrait tout aussi bien noter ce type P ou T ... mais la convention de code java est d'utiliser E (Element), K (Key), N (Number), T (Type) et V (Value). Comme ici il s'agit de trier des éléments en fonction d'une clé (Key), nous avons choisi K.

```
import java.util.Arrays;
public class TableauGenerique<K extends Comparable<K>> {
    K[] leTableau;

    public TableauGenerique(K[] objets){
        leTableau = objets;
    }
    public String toString() {
        return Arrays.toString(leTableau);
    }

    ////////////////////
    // méthodes de recherche d'éléments
    ////////////////////

    public int recherche(K x) {
        for (int i=0;i<leTableau.length;i++)
            if (x.compareTo(leTableau[i])==0) return i;
        return -1;
    }
}
```

La classe `TestTableau` montre comment utiliser la classe `TableauGenerique` sur des entiers ou des String (qui implémentent `Comparable`).

```
public class TestTableau {

    public static void main(String[] s) {
        Integer[] os = {33, 5, 8, 16, 41,8};
        TableauGenerique t = new TableauGenerique(os);
        System.out.println(t);
        System.out.println(t.recherche(8));

        String[] oss = {"il", "fait", "beau", "aujourd'hui"};
        TableauGenerique ts = new TableauGenerique(oss);
        System.out.println(ts);
        System.out.println(t.recherche("beau"));
   }
}
```

Exercices

1. **Etudiants comparables**. Créer une classe `EtudiantComparable` qui implémente `Comparable<EtudiantCompara` et qui a comme attribut un étudiant. Soyons élitistes: comparons les étudiants sur leur rang. Ajoutez quelques `EtudiantComparable` au `main` de la classe `TestTableau`.

2. **Recherche de la ième occurrence d'un élément**. Complétez la méthode `public int recherche(K x, int i)` qui renvoie la position de l'occurence d'ordre i de x dans le tableau en partant de la gauche, -1 si elle n'existe pas. Par exemple, avec le main ci-dessus, `t.recherche(8,2)` renvoie 5 et `t.recherche(8,5)` renvoie -1.

3. **Recherche dichotomique**. On suppose ici que le tableau est trié. On vous demande de compléter la méthode `rechercheVite(K x)` de la classe `TableauGenerique`. Cette méthode implémente un recherche dichotomique qui consiste à :

   - comparer l'élément cherché (noté x) à l'élément qui est au milieu du tableau (noté m)
   - si x est égal à m, c'est fini,
   - si x est inférieur à m, puisque le tableau est trié, alors il faut continuer à chercher x à la gauche de m,
   - si x est supérieur à m, puisque le tableau est trié, alors il faut continuer à chercher x à la droite de m.

   D'un point de vue *java*, la comparaison se fait en utilisant `compareTo`. Pour gérer la partie du tableau dans laquelle on cherche x, on vous recommande d'utiliser deux index *gauche* et *droite* qui sont initialisés respectivement à *0* et à *leTableau.length - 1*. Au fur et à mesure de la recherche, selon les cas ci-dessus, gauche augmente ou droite diminue.