Correction examen 3 de Programmation en Java

Christian Rinderknecht

4 février 2015

1 Dictionnaire d'occurrences

```
// Corrigé très détaillé et inefficace de l'exercice 4 du partiel de
// Java.
// Test:
// $ java Dico  1 3 0 3 3 1 7 7
// elt = 1 3 0 7
// occ = 2 3 1 2

public class Dico {
  // La méthode 'unique_occ' prend un tableau d'entiers et retourne le
  // nombre d'occurrences uniques (c'est-à-dire d'éléments non
  // répétés). Pour cela on parcours le tableau par index
  // croissants. Pour chaque élément on reparcourt le tableau du début
  // jusqu'à l'élément courant en déterminant si cet élément n'a pas
  // déjà été rencontré. S'il n'a pas été déjà rencontré, on
  // incrémente un compteur. Dans tous les cas, on passe alors à
  // l'élément suivant. Il faut donc deux boucles, l'une étant
  // imbriquée dans l'autre.
  public static int unique_occ (int[] t) {
    int n = 0; // Compteur
    int i = 0;
    int j = 0;
    for (i = 0; i < t.length; i++) {
      j = 0; // On repart du début à chaque fois.
      while (t[j] != t[i]) j++;
      if (i == j) n++; // t[i] == t[j] et i == j impliquent que t[j]
                       // est nouveau, donc n++.
    }
    return n;
  }

  // La méthode 'strip' prend un tableau d'entiers et retourne un
  // nouveau tableau contenant les mêmes éléments dans le même ordre
  // mais sans les répétitions.
  public static int[] strip (int[] t) {
    // Il nous faut d'abord déterminer la taille du tableau
    // résultant, car la taille d'un tableau est fixe. Pour cela on
    // appelera la méthode 'unique_occ'.
    int[] elt = new int [unique_occ (t)];

    // Maintenant on remplit 't' avec les occurrences uniques. Pour
```

```
    // cela on reparcours ‘t’ d’une façon identique à celle de
    // ‘unique_occ’, sauf qu’on copie en plus ‘t[i]’ dans ‘elt[n]’
    // avant d’incrémenter ‘n’.
    int n = 0;
    int i = 0;
    int j = 0;

    for (i = 0; i < t.length; i++) {
      j = 0; // On repart du début à chaque fois.
      while (t[j] != t[i]) j++;
      if (i == j) {
        elt[n] = t[i]; // Ajout par rapport à ‘unique_occ’.
        n++;
      }
    }
    return elt;
  }

  // La méthode ‘count’ prend deux tableaux d’entiers, le second
  // (‘elt’) contenant les éléments du premier (‘t’) non répétés et
  // dans le même ordre. Le tableau résultant ‘occ’ est tel que
  // ‘occ[i]’ vaut le nombre d’occurrences de ‘elt[i]’ dans ‘t’.
  public static int[] count (int[] t, int[] elt) {
    // Le tableau résultant possède donc la même dimension que
    // ‘elt’.
    int[] occ = new int [elt.length];

    // Pour le remplir, on parcourt ‘elt’ par index croissants. Pour
    // chaque élément ‘elt[i]’ on parcourt tout le tableau ‘t’ et on
    // compte le nombre de fois où cet élément apparaît. Puis ‘occ[i]’
    // prend cette valeur.
    int i = 0;
    int j = 0;
    int n = 0; // Compteur

    for (i = 0; i < elt.length; i++) {
      n = 0;
      for (j = 0; j < t.length; j++)
        if (t[j] == elt[i]) n++;
      occ[i] = n;
    }

    return occ;
  }

  // La méthode ‘print’ affiche le contenu d’un tableau d’entiers.
  public static void print (int[] t) {
      int i = 0;
      for (i = 0; i < t.length; i++)
System.out.print (t[i] + " ");
      System.out.println ();
  }

  public static void main (String[] args) {
```

```
        int[] t = new int [args.length];
        int i = 0;
        for (i = 0; i < t.length; i++)
    t[i] = Integer.parseInt (args[i]);
        int[] elt = strip (t);
        System.out.print ("elt = ");
        print (elt);
        int[] occ = count (t, elt);
        System.out.print ("occ = ");
        print (occ);
    }
}
```