TD N°06 JAVA

Exercise 1 :

```
package du.tptableaux;

import java.util.Arrays;

/**
 * Test de la classe java.util.Arrays.
 * Remplit un tableau avec des nombres aléatoires,
 * le trie et fait une recherche dichotomique
 * pour savoir si des nombres appartiennent au tableau.
 */
public class TestArrays {

  /**
   * Tire un nombre au hasard, compris au sens large
   * entre 1 et max.
   * @param max la borne haute des nombres tirés au hasard
   * @return le nombre tiré au hasard
   */
  private static int aleatoire(int max) {
    return (int)(Math.random() * max) + 1;
  }

  public static void main(String[] args) {
    int taille = 150;
    int max = 200;
    int[] t = new int[taille];
    // Remplit le tableau avec 150 nombres tirés au hasard entre 1 et 200
    for (int i = 0; i < taille; i++) {
      t[i] = aleatoire(max);
    }
    System.out.println(Arrays.toString(t));
    // Trie le tableau pour pouvoir faire des recherches dichotomiques
    Arrays.sort(t);
    // Affiche le tableau pour voir
    System.out.println(Arrays.toString(t));
    // Tire 20 nombres entre 1 et 200
    // et affiche s'ils sont dans le tableau, et en quelle position.
    for (int i = 0; i < 20; i++) {
      int n = aleatoire(max);
      int pos = Arrays.binarySearch(t, n);
      if (pos >= 0) {
        System.out.println(n + " trouvé en position " + pos);
      }
      else {
        System.out.println(n + " pas trouvé");
      }
    }
  }

}
```

Exercise 2 :

```
import java.util.Arrays;

public class Main {

  /**
   * @param args
   */
  public static void main(String[] args) {
    for (String c : args) {
      System.out.println(c);
    }
    // On voit que args est un tableau de longueur 0 s'il n'y a pas de
paramètres
    System.out.println(Arrays.toString(args));
  }
}
```

Exercise 3 :

```
import java.util.Arrays;

public class Tableau {
  private static final int VIDE = -1;

  /**
   * Afficher les éléments d'un tableau d'entiers
   * @param tableau le tableau dont on affiche les éléments
   */
  public static void afficheTableau(int[] tableau) {
    for (int i = 0; i < tableau.length; i++) {
      System.out.print(tableau[i] + " ; ");
    }
  }

  /**
   * Afficher les éléments d'un tableau d'entiers
   * @param tableau le tableau dont on affiche les éléments
   */
  public static void afficheTableau2(int[] tableau) {
    for (int val : tableau) {
      System.out.print(val + " ; ");
    }
  }

  /**
   * Afficher les éléments "non vides" d'un tableau d'entiers
   * (-1 correspond à un élément vide).
   * @param tableau le tableau dont on affiche les éléments
   */
  public static void afficheTableau3(int[] tableau) {
    int i = 0;
    while (i < tableau.length && tableau[i] != VIDE) {
      System.out.print(tableau[i] + " ; ");
      i++;
    }
  }
```

```
public static void initialise(int[] tableau) {
  for (int i = 0; i < tableau.length; i++) {
    tableau[i] = VIDE;
  }
}

public static boolean ajouterElement(int[] tableau, int element) {
  // Trouver la 1ère place vide pour ajouter l'élément
  int i = 0;
  // Remarquez le raccourci avec &&.
  // Essayez par exemple d'intervertir les conditions.
  while (i < tableau.length && tableau[i] != VIDE) {
    i++;
  }
  // Attention, l'indice le plus élevé d'un tableau à n élement est n - 1
  if (i == tableau.length) {
    return false;
  }
  tableau[i] = element;
  return true;
}

public static void remplir(int[] tableau, int element) {
  initialise(tableau);
  // Le corps du while est vide (";" suit la condition).
  while (ajouterElement(tableau, element));
}

/**
 * Teste la méthode remplir.
 * @param tableau
 * @param element
 * @return
 */
public static boolean testRemplir(int[] tableau, int element) {
  remplir(tableau, element);
  for (int valeur : tableau) {
    if (valeur != element) {
      return false;
    }
  }
  return true;
}

public static int rechercher(int[] tableau, int element) {
  int i = 0;
  while (i < tableau.length && tableau[i] != VIDE) {
    if (tableau[i] == element) {
      return i;
    }
    else {
      i++;
    }
  }
  return -1;
}

/**
 * Recherche toutes les occurrences d'une valeur dans un tableau
 * @param tableau le tableau dans lequel on cherche
```

```
   * @param valeurCherchee la valeur cherchée
   * @return un tableau de la longueur du tableau passé en paramètre
   * qui contient les indices où valeur a été trouvée
   * (les cases vides de la fin contiennent -1).
   */
  public static int[] rechercherTous(int[] tableau, int element) {
    // Le tableau qui contient les positions de element dans tableau
    int [] positionsTrouvees = new int[tableau.length];
    initialise(positionsTrouvees);
    int i = 0;
    while (i < tableau.length && tableau[i] != VIDE) {
      if (tableau[i] == element) {
        ajouterElement(positionsTrouvees, i);
      }
      i++;
    }
    return positionsTrouvees;
  }

  /**
   * Recherche toutes les occurrences d'une valeur dans un tableau.
   * Manière classique de faire (ne tient pas compte d'une valeur spéciale
"vide").
   * @param tableau le tableau dans lequel on cherche
   * @param valeurCherchee la valeur cherchée
   * @return un tableau complètement rempli
   * qui contient les indices où valeur a été trouvée.
   */
  public static int[] rechercherTousBis(int[] tableau, int valeurCherchee)
{
    // Le tableau qui contient les positions de element dans tableau
    int [] positionsTrouvees = new int[tableau.length];
    int i = 0;
    // Nombre d'éléments du tableau égaux à valeurCherchee
    int nbPositionsTrouvees = 0;
    while (i < tableau.length && tableau[i] != VIDE) {
      if (tableau[i] == valeurCherchee) {
        positionsTrouvees[nbPositionsTrouvees++] = i;
      }
      i++;
    }
    // Il faut maintenant retourner un tableau complètement rempli
    int[] tableauIndices = new int[nbPositionsTrouvees];
    // Recopie les positions trouvées dans le tableau que l'on va retourner
//    for (int j = 0; j < nbPositionsTrouvees; j++) {
//      tableauIndices[j] = positionsTrouvees[j];
//    }
    // Autre façon de faire qui utilise Arrays :
    return Arrays.copyOf(positionsTrouvees, nbPositionsTrouvees);
  }

  /**
   *
   * @param tableau tableau trié dans lequel element est cherché.
   * @param element élément cherché.
   * @return la 1ère position du tableau qui contient element.
   * -1 si element n'est pas trouvé dans le tableau.
   */
  public static int rechercherTableauTrie(int[] tableau, int element) {
    int i = 0;
    while (i < tableau.length && tableau[i] != VIDE) {
```

```
      if (tableau[i] == element) {
        return i;
      }
      // Si on est ici c'est que t[i] est différent de element
      if (tableau[i] > element) {
            // On a dépassé la valeur de element ; comme le tableau est
trié,
                // inutile d'aller plus loin
        return -1;
      }
          // On n'a pas dépassé la valeur de element ; on va plus loin
      i++;
    }
    // Pour le cas où le dernier élément du tableau est plus petit que
element
    return -1;
  }

  public static boolean croissant(int[] tableau) {
    int valeurPrecedente = -1;
    for (int valeur : tableau) {
      if (valeur == VIDE) {
        return true;
      }
      if (valeur < valeurPrecedente) {
        return false;
      }
      valeurPrecedente = valeur;
    }
    return true;
  }

  public static int rechercher2(int[] tableau, int element) {
    if (croissant(tableau)) {
      return rechercherTableauTrie(tableau, element);
    }
    else {
      return rechercher(tableau, element);
    }
  }

  /**
   * Supprimer la 1ère occurence d'une élément dans un tableau.
   * @param tableau
   * @param element
   * @return l'indice où l'élément a été supprimé.
   * -1 si l'élément n'a pas été trouvé.
   */
  public static int supprimer(int[] tableau, int element) {
    int i = rechercher(tableau, element);
    if (i == -1) {
      return -1;
    }
    // L'élément a bien été trouvé.
    // S'il y a des éléments non vides à sa droite, il faut les
    // tasser.
    int result = i; // valeur à renvoyer à la fin
    int borne = tableau.length - 2; // pour gagner un peu de temps...
    while (i < borne && tableau[i + 1] != -1) {
      // décaler la valeur vers le début du tableau
      tableau[i] = tableau[i + 1];
```

```
      i++;
    }
    // Si on n'est pas à la fin du tableau il faut vider
    // la dernière case non vide
    if (i < tableau.length - 1) {
      tableau[i] = VIDE;
    }
    return result;
  }

  /**
   * @param args
   */
  public static void main(String[] args) {
    int [] t = new int[] {1, 2, 3, 4, 12, -1, -1};
    afficheTableau(t);
    System.out.println();
    int valeurASupprimer = 1;
    System.out.println(valeurASupprimer + " trouvé en " + supprimer(t,
valeurASupprimer));
    afficheTableau(t);
    System.out.println();
    valeurASupprimer = 158;
    System.out.println(valeurASupprimer + " trouvé en " + supprimer(t,
valeurASupprimer));
    afficheTableau(t);
    System.out.println();
    valeurASupprimer = 12;
    System.out.println(valeurASupprimer + " trouvé en " + supprimer(t,
valeurASupprimer));
    afficheTableau(t);
    System.out.println();
  }

}
```

Exercise 4 :

```
/**
 * Construction triangle de Pascal et affichage.
 */
public class TrianglePascal {
  public static int[][] construireTriangle(int n) {
    int[][] trianglePascal = new int[n + 1][];
    // La première ligne pour initialiser le processus
    trianglePascal[0] = new int[]{1};
    int[] lignePrecedente = trianglePascal[0];
    // Construction des lignes
    for (int i = 1; i < n + 1; i++) {
      // ligne numéro i
      int[] ligne = new int[i + 1];
      // Valeurs aux 2 bouts
      ligne[0] = 1;
      ligne[i] = 1;
      // autres valeurs (rien pour la 2ème ligne car i = 1)
      for (int col = 1; col < i; col++) {
        ligne[col] = lignePrecedente[col - 1] + lignePrecedente[col];
      }
      // Raccroche la ligne au triangle
      trianglePascal[i] = ligne;
      // Pour la boucle suivante
      lignePrecedente = ligne;
    }
    return trianglePascal;
  }

  public static void afficherTriangle(int[][] tableau) {
    for (int ligne = 0; ligne < tableau.length; ligne++) {
     for (int colonne = 0;  colonne < tableau[ligne].length; colonne++) {
        System.out.printf("%5d", tableau[ligne][colonne]);
      }
      System.out.println();
    }

  }

  public static void main(String[] args) {
    afficherTriangle(construireTriangle(5));
  }

}
```