MOOC Init Prog Java
Corriges semaine 7

Les corrigés proposés correspondent à l'ordre des apprentissages : chaque corrigé correspond à la solution à laquelle vous pourriez aboutir au moyen des connaissances acquises jusqu'à la semaine correspondante.

Exercice 23: Tri de Shell (Tableaux statiques)

La principale difficulté de cet exercice réside dans la manipulation des indices.

Le but premier de cet exercice est en effet de vous faire réfléchir au passage de la notation mathématique (qui utilise des indices de 1 à N) à la notation C++ (qui utilise des indices de 0 à N-1).

Très souvent, il suffit de changer la boucle 1->N en une boucle 0->N-1 et le tour est joué. Mais en fait la vraie bonne traduction est plus subtile que cela.

Quand en mathématique on note u_i, i.e. le `i`-ème élément de `u` ($i \geq 1$), cela correspond en C++ à un `u[i-1]`. Donc, en fait, c'est *l'accesseur* aux éléments du tableau qui translate les indices de 1, **et lui seul** !

La seule et unique bonne façon de passer de la notation mathématique à la notation C++ à tous les coups sans se tromper consiste donc à translater (de 1) tous les accesseurs. Ni plus, ni moins.

Ensuite, et seulement ensuite, si une écriture plus simple existe, alors on peut réécrire le tout (avec un changement d'indice). C'est exactement ce qui se passe pour le cas des boucles les plus simples que l'on peut donc écrire de 0 à N-1. Mais ceci n'est qu'un cas particulier, très simple, qui ne fonctionne pas à tous les coups... ...en particulier pas pour le tri Shell qui a des boucles plus compliquées (voir le corrigé).

```
class Shell
{
    public static void main(String[] args)
    {
      int[] tab = { 3, 5, 12, -1, 215, -2, 17, 8, 3,
                    5, 13, 18, 23, 5, 4, 3, 2, 1    };
      System.out.println("A trier  : ");
      affiche(tab);
      triShell(tab);
      System.out.println("Résultat :");
      affiche(tab);
    }

  static void affiche(int[] tab)
    {
        for(int el : tab) {
          System.out.print(el +  " ");
        }
```

```
      System.out.println();
    }


  // inverse le contenu des cases d'indices index1 et index2
  // du tableau tab
  static void swap(int[] tab, int index1, int index2)
    {
      int temp = tab[index1];
      tab[index1] = tab[index2];
      tab[index2] = temp;
    }


  static void triShell(int[] tab)
    {
      for(int k = (tab.length)/2; k >= 1; k /= 2)
        for (int i = k+1; i <= tab.length; ++i) {
          int j = i-k;
          while (j > 0) {
            if (tab[j-1] > tab[j+k-1]) {
                swap(tab, j-1, j+k-1);
              j -= k;
            } else {
              j = 0;
            }
          }
        }
    }
}
```

Exercice 24: Sapin et guirlandes (algorithme)

```java
/*
 * Voici un codage possible de cet exercice
 * La code fourni permet de mettre en oeuvre les deux parties de l'exercice.
 * La première partie pourrait être réalisée sans passer par un tableau.
 * L'usage du tableau est surtout destiné à faciliter la programmation
 * de la seconde partie.
 */


import java.util.Scanner;


public class Sapin {


  /*
   * méthode remplissant le triangle, on construit à chaque ligne
   * le tableau de char de la dimension
   *  correspondante (1 a la ligne 0, 3 à la ligne 1, etc)
   */
  static void remplirTriangle(char[][] triangle, char c) {
    int nbLigne = triangle.length;
    int nbColonne;

    for (int i = 0; i < nbLigne; ++i) {
      nbColonne = i*2 + 1;
      triangle[i] = new char[nbColonne];
      for (int j = 0; j < nbColonne; ++j) {
        triangle[i][j] = c;
      }
    }
  }


  /*
   * Méthode ajoutant des guirlandes au triangle qui représente un sapin
   * sans decorations.
   */
  static void mettreGuirlande(char[][] sapinTriangle, String guirlande) {
    int nbLigne = sapinTriangle.length;

    // La variable pointeurChar stocke l'indice du prochain caractère de
    // la guirlande à placer
    int pointeurChar = 0;
    int longueurChaine = guirlande.length();

    // Cette variable permet de différencier le cas où l'on pose les
    // guirlandes complètement, et le cas où l'on étend une
    // guirlande sur les lignes supplémentaires
    boolean completerGuirlande = false;

    //les deux variables d'indice que l'on fera varier
    int i = 0;
    int j = 0;
```

```
while(i < nbLigne) {
  // on vérifie d'abord que nous sommes dans une ligne impaire
  // ou alors que l'on est en train de finir d'étendre une
  // guirlande sur les lignes suivantes
  if (i % 2 == 1 || completerGuirlande) {

    // avec ces 2 instructions on remplace le caractère (i,j)
    // du sapin par le prochain caractère à placer
    // de la guirlande. pointeurChar donne la position
    // de ce caractère dans la guirlande. On incrémente ensuite
    // cette position en prenant le reste de la division
    // par rapport à la longueur de la guirlande
    //(quand on dépasse la position maximale possible, pointeurChar
    // reprend la valeur 0)
    sapinTriangle[i][j] = guirlande.charAt(pointeurChar);
    pointeurChar = (pointeurChar + 1) % longueurChaine;

    //ensuite il faut incrémenter les i et j en fonction de la
    //situation

    if ((j >= sapinTriangle[i].length - 1 && pointeurChar != 0) || complete
      // Ce if gère le cas assez compliqué où il faut finir d'étendre la
      // guirlande sur la ligne d'en dessous. On utilise un boolean pour en
      // dans ce if jusqu'à ce que pointeurChar vaille 0; à ce moment on sa
      // que l'on a finit d'étendre la guirlande et on remet le boolean à f
      // incrémente alors le j d'une quantité aléatoire. On n'incrémente pa
      // est tout à fait possible d'avoir couvert la deuxième ligne entière
      // d'être revenu à une ligne impaire, dans ce cas on doit simplement
      // avec la guirlande suivante

      if (pointeurChar == 0) {
        completerGuirlande = false;
        j += random() + 1;
      } else {
        // sinon on gère les 3 cas possibles:
        // 1.  on arrive pour la première fois ici et on change de ligne et
        // au dernier index de la ligne d'en dessous
        //  sinon, on est en train de compléter la guirlande (variable à vr
        // 2. soit la ligne est paire, auquel cas on remplit de droite à  g
        // 3. elle est impaire et on remplit de gauche à  droite (cas de lo
        // a recouvert toute la ligne supplémentaire et qu'on continue d'ét
        // suivante

        if (!completerGuirlande) {
          completerGuirlande = true;
          ++i;
          if (i < nbLigne) {
            j = sapinTriangle[i].length - 1;
          }
        } else if (i % 2 == 0) {
          --j;
          if (j < 0) {
            ++i;
            j = 0;
          }
        } else {
          ++j;
          if (j >= sapinTriangle[i].length) {
```

```
              ++i;
              if (i < nbLigne) {
                j = sapinTriangle[i].length - 1;
              }
            }
          }
        }
      } else {
        // else de la condition
        // (j >= sapinTriangle[i].length - 1 && pointeurChar != 0) || complet
        //Dans ce cas on fait l'incrémentation normale de j, et éventuellemen
        //le nombre aléatoire pour laisser un peu d'espace
        if (pointeurChar == 0) {
          j += random();
        }
        ++j;
      }
      //finalement on fait la vérification finale, qui incrémente la ligne et
      if (i < nbLigne && j >= sapinTriangle[i].length && pointeurChar == 0) {
        ++i;
        j = 0;
      }
    } else {
      //si la ligne n'était pas paire, on incrémente la ligne
      j = 0;
      ++i;
    }
  }
}

/* Méthode affichant le tableau sous forme triangulaire.
 */
static void afficherTriangle(char[][] triangle) {
  int nbLigne = triangle.length;
  int nbColonne;
  int nbEspace;

  for (int i = 0; i < nbLigne; ++i) {

    nbEspace = ((2*nbLigne - 1) - (2*i + 1))/2;
    for (int j = 0; j < nbEspace; ++j) {
      System.out.print(" ");
    }

    nbColonne = triangle[i].length;
    for (int j = 0; j < nbColonne; ++j) {
      System.out.print(triangle[i][j]);
    }

    System.out.println();
  }
}

/* Méthode ajoutant un tronc à l'arbre, elle choisit la hauteur et
 * la largeur en fonction du nombre de ligne du sapin.
 */
static void afficherTronc(int nbLigne) {
  int largeur;
```

```
  int hauteur;

  largeur = (nbLigne*2 - 1) / 5;
  if (largeur % 2 == 0)
    ++largeur;
  hauteur = Math.max(1, nbLigne/3);

  for (int i = 0; i < hauteur; ++i) {
    for (int j = 0; j < ((nbLigne*2 -1) - 3)/2; ++j)
      System.out.print(" ");
    for (int j = 0; j < largeur; ++j)
      System.out.print("|");
    System.out.println();
  }
}

/*
 * Méthode choisissant aléatoirement entre 2 et 3
 */

static int random() {
  int val = (int)(Math.random()*2); // 0 or 1
  return (val+2); //return 2 or 3
}

/*
 * Programme principal
 * on demande  les données nécessaires à l'utilisateur et vérifie que
 * les informations saisies soient correctes.
 * On appelle ensuite les méthodes dans le bon ordre pour construire puis
 * afficher le sapin
 */
public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  char[][] triangle;
  char symbole;
  int nbLigne;
  String guirlande;


  System.out.print("Quel symbole voulez-vous pour les épines du sapin? ");
  symbole = sc.nextLine().charAt(0);
  do {
    System.out.print("Combien de ligne (de 8 a 35)? ");
    nbLigne = sc.nextInt();
  } while (nbLigne < 8 || nbLigne > 35);
  do {
    System.out.print("Quelles guirlandes voulez-vous mettre " +
             "(taille de 3 a 25 caractères et elles ne peuvent \n" +
             "pas contenir le même caractère que celui utilisé pour les épine
    guirlande = sc.next();
  } while (guirlande.length() < 3 || guirlande.length() > 25 || guirlande.con

  System.out.println();

  //on initialise seulement le nombre de lignes que contiendra ce tableau (li
  //choisir à chaque ligne le nombre d'éléments (colonne)
  triangle = new char [nbLigne][];
```

```
    remplirTriangle(triangle, symbole);
    mettreGuirlande(triangle, guirlande);
    afficherTriangle(triangle);
    afficherTronc(triangle.length);

    /* Note pour la partie 1: il suffit d'invoquer les méthodes suivantes
     * (après la saisie des données utiles) :
     *
     * remplirTriangle(triangle, symbole);
     * afficherTriangle(triangle);
     */
  }

}
```

Exercice 25: Master-Mind(tm) (facultatif) (Modularisation, algorithme)

```java
import java.util.Scanner;

class Mastermind {

    private static Scanner scanner = new Scanner(System.in);

    /**
     * Lancement du Master-Mind(tm).
     * @param args Inutilisé dans ce programme
     */
    public static void main(String[] args) {
        mastermind(4, 6, 10);
    }

    /**
     * Tirage d'un entier au hasard entre 1 et max.
     * @param max La valeur maximale pouvant être retournée
     * @return Un entire aléatoire entre 1 et max
     */
    static int hasard(int max) {
        return (1 + (int) (Math.random() * max));
    }

    /**
     * Tire une combinaison à deviner.
     * C'est la référence du tableau qui est passée
     * en paramètre. La méthode tirerCombinaison pourra directement
     * modifier la combinaison en mémoire.
     * Ici, on suppose que n < combinaison.length.
     * @param combinaison Référence vers le tableau à modifier
     * @param n Longueur de la combinaison.
     * @param m Valeur maximale à tirer aléatoirement
     */
    static void tirerCombinaison(int[] combinaison, int n, int m) {
        for (int i = 0; i < n; i++) {
            combinaison [i] = hasard(m);
            }
    }

    /**
     * Permet de lire la combinaison proposée
     * par le joueur.
     * @param combinaison Tableau dans lequel stocker les valeurs entrées par l
     * joueur
     * @param n Longueur de la combinaison
     */
    static void demanderCoup(int[] combinaison, int n) {
        System.out.print("Entrez les ");
        System.out.print(n);
        System.out.print(" chiffres de votre proposition ");
        System.out.println("(terminés par un retour chariot) :");
        for (int i = 0; i < n; i++) {
            combinaison[i] = scanner.nextInt();
        }
    }
```

```java
/**
 * Permet de comparer la combinaison à deviner avec la
 * combinaison proposée par le joueur.
 * Dans reponse[0] sera stocké le nombre d'éléments bien devinés
 * et correctement placés.
 * Dans reponse[1] sera stocké le nombre d'éléments bien devinés
 * mais mal placés.
 * @param n Longueur de la combinaison
 * @param combinaison1 Combinaison proposée par le joueur
 * @param combinaison2 Combinaison à deviner
 * @param reponse Tableau de deux int contenant le nombre de bien placés
 * (index 0) et le nombre de mal placés (index 1)
 * @return true si la bonne combinaison est trouvée et false sinon
 */
static boolean compare(int n, int[] combinaison1, int[] combinaison2,
            int[] reponse) {
    // nombre de bien placés
    int nbOk = 0;
    // nombre de mal placés
    int nbCol = 0;
    boolean [] marque = new boolean[n];
    boolean trouve = true;
    // cette première boucle sert à trouver
    // les éléments bien devinés et correctement placés.
    // Le tableau marque permet de marquer de tels
    // éléments pour qu'ils ne soient pas considérés
    // plusieurs fois.
    for (int i = 0; i < n; i++) {
        if (combinaison1 [i] == combinaison2 [i]) {
            nbOk++;
            marque[i] = true;
        } else {
            trouve = false;
            marque[i] = false;
        }
    }
    // la deuxième boucle suivante sert à identifier les
    // éléments bien devinés mais mal placés.
    for (int i = 0; i < n; i++) {
        if (combinaison1[i] != combinaison2[i]) {
            int j = 0;
            boolean trouveMalPlace = false;
            while ((j < n) && !trouveMalPlace) {
                if (!marque[j] && (combinaison1[i] == combinaison2[j])) {
                    nbCol++;
                    marque[j] = true;
                    trouveMalPlace = true;
                }
                j++;
            }
        }
    }

    reponse[0] = nbOk;
    reponse[1] = nbCol;
    return trouve;
}
```

```java
/**
 * Affichage d'une combinaison.
 * @param combinaison Combinaison à afficher
 * @param n Longueur de la combinaison
 */
static void afficheCombinaison(int[] combinaison, int n) {
    for (int i = 0; i < n; i++)
        System.out.print(combinaison[i]);
    System.out.println(" ");
};

/**
 * Affichage des indications destinées au joueur.
 * @param reponse Tableau de deux int contenant le nombre de bien placés
 * (index 0) et le nombre de mal placés (index 1)
 */
static void afficheReponse(int[] reponse) {
    for (int i = 0; i < reponse[0]; i++)
        System.out.print('#');
    for (int i = 0; i < reponse[1]; i++)
        System.out.print('O');
    System.out.println();
}

/**
 * Affichage du texte d'acceuil.
 * @param n Longueur du code
 * @param m Valeur maximale permise dans la combinaison
 * @param maxCoups limite de coups autorisée
 */
static void bienvenue(int n, int m, int maxCoups) {
    System.out.print("Pouvez vous trouver ma combinaison de ");
    System.out.print(n);
    System.out.print(" chiffres [compris entre 1 et ");
    System.out.print(m);
    System.out.print(" avec répétitions possibles]\n en moins de ");
    System.out.print(maxCoups);
    System.out.println(" coups ?");
}

/**
 * Jeu du Master-Mind.
 * Le programme tire une combinaison au hasard.
 * Le joueur cherche à la deviner et fait des propositions de
 * combinaisons.
 * Le programme indique à chaque coup au joueur combien d'éléments
 * sont bien devinés et correctement placé ou bien devinés mais
 * mal placés.
 * Le jour a droit à <maxCoups> tentatives.
 * @param size Longueur du code à deviner
 * @param maxDigit Valeur maximale pouvant se trouver dans le code
 * @param maxCoups Limite de coups pour trouver la réponse
 */
static void mastermind(int size, int maxDigit, int maxCoups) {
    int[] solution = new int[size];
    int[] proposition = new int[size];
    int nbCoups = 0;
    boolean trouve = false;
```

```
        int[] reponse = new int[2];

        bienvenue(size, maxDigit, maxCoups);
        tirerCombinaison(solution, size, maxDigit);

        do {
            demanderCoup(proposition, size);
            nbCoups++;
            trouve = compare(size, solution, proposition, reponse);
            afficheReponse(reponse);
        } while (!trouve && (nbCoups < maxCoups));

        if (trouve) {
            System.out.print("Bravo ! Vous avez trouvé en ");
            System.out.print(nbCoups);
            System.out.println(" coups");
        } else {
            System.out.println("Désolé vous n'avez pas trouvé...");
            System.out.println("La bonne réponse était ");
            afficheCombinaison(solution, size);
            System.out.println(".");
        }
    }

}
```
