Université Paris 7 - Denis Diderot
IF1 - Corrigé de l'examen du 3 janvier 2006

Exercice 1.

1.1. Les variables a, b, et c définies au lancement de l'application (appel de la méthode `main`) sont initialisées respectivement à 1, 2 et 3. La variable d n'est pas initialisée.
- À l'appel de la méthode `fonc1`, création (appel par valeur), sur la pile d'exécution, de variables locales de noms a, b et c (paramètres formels dans l'en-tête de la méthode). Pour marquer le fait que les variables a, b et c de la méthode `fonc1` sont différentes de celles de la méthode `main`, nous les désignerons par a', b' et c'. Chacune de ces variables est initialisée à la valeur du paramètre d'appel correspondant. Ainsi :

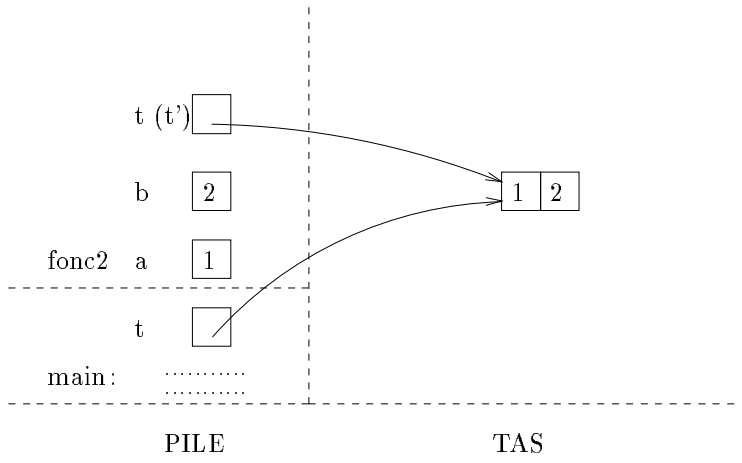
- $a' = a' + 1$, c'est-à-dire $a' = 2$;
- $b' = b' + 2$, c'est-à-dire $b' = 4$;
- $c' = a' + b'$, c'est-à-dire $c' = 6$;
- la fonction renvoie l'entier $2 * c'$, c'est-à-dire 12.

- Au retour de l'appel de la méthode `fonc1`, la variable d prend comme valeur 12 (valeur renvoyée par l'appel de la méthode), les variables a, b et c de la méthode `main` n'ayant quant-à-elle pas été modifiées. D'où l'affichage (un nombre par ligne) de 1, puis 2, puis 3 et enfin 12.

1.2. Le lancement du programme crée une variable t de type référence sur tableau d'entiers. Une demande de création d'un tableau de 2 entiers est ensuite réalisée (`new int[2]`) : l'espace nécessaire en mémoire est alloué et l'adresse de cet espace, valeur de l'expression `new int[2]` est affecté à la variable t (soit α cette valeur). Les deux éléments du tableau référencé par t sont initialisés respectivement aux valeurs 1 et 2.
- Un appel à la méthode `fonc2` est ensuite réalisé et :

- des variables variables a et b de type `int` et une variable t de type référence sur tableau d'entiers sont créées. Pour distinguer cette dernière variable de la variable t définie dans la méthode `main` nous la désignerons par t' ;
- les variables a, b et t' sont initialisées respectivement aux valeurs 1, 2 et α. Une conséquence de cette dernière affectation est que $t[0]$ et $t'[0]$ correspondent au même emplacement en mémoire. Il en est de même pour $t[1]$ et $t'[1]$.

Cela correspond au schéma suivant :

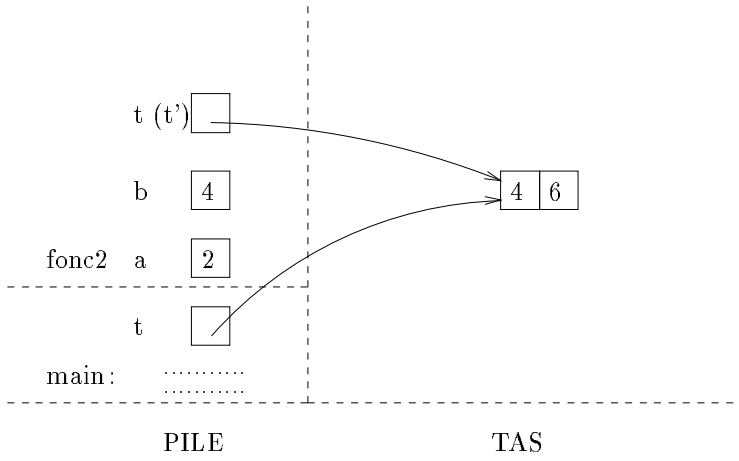


- Le corps de la méthode est exécuté :

- a prend la valeur 2 $(1 + 1)$;

- b prend la valeur 4 $(2+2)$;
- $t'[0]$ (donc $t[0]$) prend la valeur 4 (1+3) ;
- $t'[1]$ (donc $t[1]$) prend la valeur 6 (2+4) ;

Avant retour de l'appel de la méthode, l'état de la mémoire correspond au schéma suivant :



Les informations sur la pile liées à l'invocation de la méthode `fonc2` disparaissent et l'exécution reprend dans le corps de la méthode `main` après l'instruction d'appel de la méthode `fonc2`. Les valeurs de $t[0]$ et $t[1]$ sont écrites, c'est-à-dire 4 et 6 (un nombre par ligne).

Exercice 2.

```
import fr.jussieu.script.*;
class Exo2 {
   public static void main(String[] args) {
      int pg1 = 0;  // le plus grand nombre lu
      int pg2 = 0;  // le deuxième plus grand nombre lu
      int x;        // le nombre courant (qu'on lit)
      int nbLu = 0; // le nombre d'entiers >=0 deja lus
   x = Deug.readInt();
   while(x > 0) { // test si le nombre lu x est >= 0
      nbLu ++;    // compteur de nombres lus incrémenté
      // si x > le plus grand déja lu (pg1)
      if (x > pg1) {
         pg2 = pg1;  // l'ancien plus grand devient devient second plus grand
         pg1 = x; }  // x devient le plud grand
      // si x est <= plus grand déja lu (pg1) et est > second plus grand (pg2)
      else if (x > pg2)
         pg2 = x;    // x devient le second plus grand
      x = Deug.readInt(); // on lit un nouveau nombre
      }
   if(nbLu < 2) { // si moins de deux nombres lus, erreur et terminaison
      Deug.println("On a lu moins de deux nombres");
      Deug.exit(); }
   // sinon on affiche la valeur de pg2
   Deug.println("Resultat : " + pg2);
   }
}
```

Exercice 3.

3.1. Méthode vérifiant qu'un tableau est strictement croissant.

```
static boolean croissant(int[] a) {
   for(int i = 0; i < a.length - 1; i++)
       if (a[i + 1] <= a[i])
           return false;
   return true;
}
```

3.2. Méthode vérifiant que deux tableaux strictement croissants sont disjoints.
La méthode utilisée exploite le fait que les tableaux sont croissants (on peut cependant être plus brutal et comparer systématiquement chacun des couples (a[i],b[j])).

```
static boolean disjoints(int[] a, int[] b) {
   int indA = 0; // indice dans tableau a
   int indB = 0; // indice dans tableau b
   for(indA = 0; indA < a.length; indA++){
      while((indB < b.length) && (b[indB] < a[indA])) indB++;
         if(indB == b.length) return true;
         if(b[indB] == a [indA]) return false;
   }
   return true;
}
```

3.3. Méthode construisant un tableau strictement croissant obtenu par fusion des éléments de deux tableaux strictement croissants donnés en paramètres.

```
static int[] melange(int[] a, int[] b) {
   int[] tab = new int[a.length + b.length];
   int indTab; // indice dans le tableau tab
   int indA = 0; // indice dans le tableau a
   int indB = 0; // indice dans le tableau b
   for(indTab = 0; indTab < tab.length; indTab++)
       if(indA == a.length){
              tab[indTab] = b[indB]; indB++; }
       else if(indB == b.length){
              tab[indTab] = a[indA]; indA++; }
       else if(b[indB] < a[indA]){
              tab[indTab] = b[indB]; indB++; }
       else { tab[indTab] = a[indA]; indA++; }
   return tab;
}
```

3.4. Programme principal.

```
public static void main(String[] args) {
   int tailleA, tailleB;
   int[] a, b, c;
   // lecture de la taille du premier tableau
   Deug.print("Taille de a ? "); tailleA = Deug.readInt();
   if (tailleA < 0) Deug.exit(); // taille négative : erreur
   // lecture de la taille du second tableau
   Deug.print("Taille de b ? "); tailleB = Deug.readInt();
```

```
      if (tailleB < 0) Deug.exit(); // taille négative : erreur
      a = new int[tailleA];  // allocation du tableau a
      b = new int[tailleB];  // allocation du tableau b
      Deug.println("Lecture du tableau a");
      for(int i = 0; i < tailleA; i++) a[i] = Deug.readInt();
      Deug.println("Lecture du tableau b");
      for(int i = 0; i < tailleB; i++) b[i] = Deug.readInt();
      // si les tableaux ne sont pas tous les deux croissants,  erreur
      if( !croissant(a) || !croissant(b) || !disjoints(a,b)){
         Deug.println("Tableaux incorrects"); Deug.exit(); }
      // appel de melange et récupération de la référence du résultat
      c = melange(a, b);
      // impression du resultat
      for(int i = 0; i < c.length; i++) Deug.print(c[i] + " ");
      Deug.println();
   }
}
```

Exercice 4.
Les réponses aux différentes questions sont données à l'intérieur d'une même classe. Aucun contrôle n'est fait sur les paramètres (on suppose qu'ils correspondent à ce qu'on attend).

```
import fr.jussieu.script.*;
class ExamSudoku {
   // Question 4.1.1 : plus simple qu'un switch
   static int ligne(int carre) { return 3 * (carre / 3); }

   // Question 4.1.2 : plus simple qu'un switch
   static int colonne(int carre) { return 3 * (carre % 3); }

   // Question 4.1.3 : plus simple qu'un switch
   static int sousCarre(int lig, int col) { return 3 * (lig / 3) + (col / 3); }

   // Question 4.1.4
   static boolean dansLigne(int[][] t, int lig, int val) {
      // parcours des éléments de la ligne
      for(int i = 0; i < 9; i++)
         if(t[lig][i] == val) return true; // val a été trouvé
      // val n'a pas été trouvé
      return false; }

   // Question 4.1.5
   static boolean dansColonne(int[][] t, int col, int val) {
      // parcours des éléments de la colonne
      for(int i = 0; i < 9; i++)
         if(t[i][col] == val) return true; // val a été trouvé
      // val n'a pas été trouvé
      return false; }
```

```
// Question 4.1.6
static boolean dansSousCarre(int[][] t, int carre, int val) {
   int lig = ligne(carre); // première ligne du sous-carré
   int col = colonne(carre); // première colonne du sous-carré
   // parcours du sous-carré ligne par ligne
   // pour rechercher la valeur donnée
   for(int i = lig; i < lig + 3; i++)
      for(int j = col; j < col + 3; j++)
         // trouve-t-on la valeur en t[i][j] ?
         if(t[i][j] == val) return true; // oui, on renvoie true
   // on n'a pas trouvé la valeur dans le sous-carré
   return false; }

// Question 4.1.7 : le tableau booléen trouve mémorise les
// nombres rencontrés dans le tableau donné en paramètre.
static boolean estPerm(int[] t) {
   boolean[] trouve = new boolean[9];
   // initialisation du tableau à false
   for(int i = 0; i < 9; i++) trouve[i] = false;
   // parcours du tableau donné en paramètre
   for(int i = 0; i < 9; i++)
      // si l'élément a déjà été rencontré, on renvoie flse
      if(trouve[t[i] - 1]) return false;
      // sinon, on mémorise qu'on l'a rencontré
      else trouve[t[i] - 1] = true;
   // on a parcouru tout le tableau : c'est une permutation
   return true; }

// Question 4.2 : on appelle directement estPerm sur la ligne
static boolean verifLig(int[][] t, int lig) {
   return estPerm(t[lig]); }

// Question 4.3 : on recopie la colonne dans un tableau d'entiers
// puis on appelle la méthode estPerm sur ce tableau
static boolean verifCol(int[][] t, int col) {
   int[] colonne = new int[9]; // allocation du tableau
   // recopie de la colonne col dans le tableau colonne
   for(int lig = 0; lig < 9; lig++)
      colonne[lig] = t [lig][col];
   // appel et renvoi de la valeur de la méthode estPerm
   return estPerm(colonne); }

// Question 4.4 : on recopie le sous-carré dans un tableau d'entiers
// puis on appelle la méthode estPerm sur ce tableau
static boolean verifSousCarre(int[][] t, int carre) {
   int[] petitCarre = new int[9]; // allocation du tableau
   int lig = ligne(carre); // première ligne du sous-carré
   int col = colonne(carre); // première colonne du sous-carré
   int ind = 0; // indice pour remplir le tableau
```

```
   // recopie du sous-carré dans le tableau petitCarre
   for(int l = lig; l < lig + 3; l++)
      for(int c = col; c < col + 3; c++){
          petitCarre[ind] = t [l][c]; ind ++; }
   // appel et renvoi de la valeur de la méthode estPerm
   return estPerm(petitCarre); }

// Question 4.5 : on vérifie que les 9 lignes, les 9 colonnes et
// les 9 sous-carrés sont des permutations
static boolean verifTableau(int[][] t) {
   for(int i = 0; i < 9; i++)
      if(!verifLig(t,i) || !verifCol(t,i) || !verifSousCarre(t,i))
           return false;
   return true; }

// Question 4.6
static boolean placer(int[][] t, int lig, int col, int val) {
   // vérification que la case est vide
   if(t[lig][col] != 0) return false;
   // on détermine à quel sous-carré elle appartient
   int carre = sousCarre(lig, col);
   // la valeur peut y être placée si elle n'est pas déjà
   // dans la ligne, la colonne et le sous-carré
   return(!dansLigne(t, lig, val) &&
          !dansColonne(t, col, val) &&
          !dansSousCarre(t, carre, val));
   }

// Question 4.7
static int[] tabValeurs(int[][] t, int lig, int col) {
   // si la case est occupée, on renvoie null
   if(t[lig][col] != 0) return null;
   int n = 9; // nombre de valeurs possibles plaçables ( a priori 9)
   // le tableau de booléens va mémoriser les valeurs interdites
   // c-à-d appartenant  à la ligne, la colonne et le sous-carré
   boolean[] impossibles = new boolean[9];
   for(int i = 0; i < 9; i++){
        impossibles[i] = dansLigne(t, lig, i + 1) ||
        dansColonne(t, col, i + 1) ||
         dansSousCarre(t, petitCarre(lig,col), i + 1);
        // si la valeur i a été trouvée dans la ligne,
        // ou la colonne ou le sous-carré, elle est interdite
        if(impossibles[i]) n = n - 1; // on diminue n de 1
        }
   // il n'y a aucune valeur possible, on renvoie null
   if(n == 0) return null;
   // sinon il y en a n
   int[] res = new int[n]; // on alloue untableau de n entiers
   int ind = 0; // indice pour remplir ce tableau
```

```
        // on parcours le tableau des valeurs impossibles
        for(int i = 0; i < 9; i ++)
             // si impossibles[i] est false, la valeur i+1 est possible
             // on la mémorise donc dans le tableau res à l'indice ind
             // on incrémente l'indice ind pour la prochaine écriture
             if(!impossibles[i]){ res[ind] = i + 1; ind++;}
        return res; // on renvoie le tableau res en résultat
        }
}
```