Introduction à la Programmation **Java** (IP1-Java)
Examen – **Durée : 3 heures**

Université Paris-Diderot – Lundi 11 Décembre 2017

- Aucun document ni aucune machine ne sont autorisés. Les téléphones doivent être rangés.
- Les exercices sont tous indépendants.
- **Attention** : Indiquez au début de votre copie quel langage de programmation vous utiliserez dans le restant de votre copie. Il n'est pas possible de changer de langage une fois celui-ci choisi. Pour rappel, les filières INFO, MATHS-INFO et L3-M1 MIASHS Linguistique doivent composer en JAVA mais si vous voulez composer en PYTHON, merci de nous l'indiquer.
- Une réponse peut utiliser les réponses attendues à une question précédente (même si elle est non traitée).
- Les fragments de code doivent être correctement indentés.

Exercice 1

1. *Décrire le contenu des tableaux* t1 *et* t2 *après exécution du programme suivant.*

```
public class Test{
  public static int[] func(int[] t){
    for(int i=0;i<t.length;i=i+1){
      if((i%2==0) && (i<(t.length-1))){
        int tmp = t[i];
        t[i] = t[i+1];
        t[i+1] = tmp;
      }
    }
    return t;
  }

  public static void main(String[] args){
    int []t1 = {1,2,3,4,5};
    int []t2 = func(t1);
  }
}
```

2. *Donner la valeur des variables* a *et* b *après exécution du programme suivant.*

```
public class Test2{
  public static int f(int c){
    int d = 2*c;
    int e = g(d+1);
    return e;
  }

  public static int g(int x){
    int z = 0;
    if(x>0){
      z = f(x/3);
    } else {
      z = f(2*x);
    }
    return z;
  }

  public static void main(String[] args){
    int a = f(1);
    int b = g(3);
  }
}
```

□

Exercice 2 *On considère la suite arithmétique définie de la façon suivante :*

$$U_0 = 1$$
$$U_1 = 2$$
$$U_2 = 3$$
$$U_N = U_{N-3} + U_{N-1} \text{ pour } N > 2$$

Ainsi on a $U_3 = U_0 + U_2 = 1 + 3 = 4$, $U_4 = U_1 + U_3 = 2 + 4 = 6$, *etc.*

1. *Écrire une fonction* suiteU *qui prend en argument un entier positif* n *et renvoie un tableau d'entiers avec le contenu suivant* $\{U_0, U_1, ..., U_n\}$, *c'est-à-dire un tableau où l'élément à la position* i *vaut* U_i.
   Exemple : suiteU(5) *renvoie le tableau* {1,2,3,4,6}.
2. *On s'intéresse maintenant à la série mathématique de terme général* U_N *définie de la façon suivante :*

   $$S_N = \Sigma_{i=0}^{N} U_i = U_0 + U_1 + U_2 + \ldots + U_N \text{ pour } N \geq 0$$

   Ainsi $S_0 = U_0 = 1$, $S_1 = U_0 + U_1 = 1 + 2 = 3$, $S_2 = U_0 + U_1 + U_2 = 1 + 2 + 3 = 6$, *etc. On remarque que pour tout* $N > 0$, *on a* $S_N = S_{N-1} + U_N$. *Écrire une fonction* serieU *qui prend en argument un entier positif* n *et renvoie un tableau d'entiers avec le contenu suivant* $\{S_0, S_1, ..., S_n\}$, *c'est-à-dire un tableau où l'élément à la position* i *vaut* S_i.
3. *On veut maintenant savoir si un entier* x *est présent parmi les* $S_0, S_1, \ldots, S_n$ *pour un* n *donné. Écrire une fonction* presentSerieU *qui prend en arguments un entier positif* n *et un entier* x *et qui renvoie* true *s'il existe un* $i \leq n$ *tel que* $S_i = x$ *et* false *sinon.*

□

Exercice 3 *Le but de cet exercice est de programmer un algorithme sur des tableaux d'entiers et de comprendre ce qu'il fait.*

1. *Écrire une fonction* echange *qui prend en arguments un tableau d'entiers* tab *et une position* p *comprise entre* 0 *et* tab.length-2 *inclus et qui échange les contenus des cases* p *et* p$+1$ *si et seulement si* tab[p] *est strictement plus grand que* tab[p+1]. *Cette fonction modifie le tableau mais ne renvoie rien.*
   Exemple : *Si* tab *indique le tableau* {4,2,8,5}, *après* echange(tab,0), *le tableau indiqué par* tab *vaut* {2,4,8,5}.
2. *Écrire une fonction* balayage *qui prend en argument un tableau d'entiers* tab *et applique successivement la fonction* echange *sur toutes les positions allant de* 0 *à* tab.length-2 *inclus. Cette fonction modifie le tableau mais ne renvoie rien.*
   Exemple : *Si* tab *indique le tableau* {3,2,5,8,4}, *après* balayage(tab), *le tableau indiqué par* tab *vaut* {2,3,5,4,8}.
3. *Écrire une fonction* algo *qui prend en argument un tableau d'entiers* tab *et applique successivement* tab.length *fois la fonction* balayage. *Cette fonction modifie le tableau mais ne renvoie rien.*
4. *Si l'on exécute les deux instructions suivantes* int [] t={4,1,2,3}; *et* algo(t);, *quelle est le contenu du tableau* t *à la fin ? À votre avis, que fait la fonction* algo *?*

□

Exercice 4 *Le but de cet exercice est de déterminer si étant donné un tableau de nombres entiers positifs tous différents, un nombre donné peut s'écrire comme somme de deux d'entre eux.*

1. *Écrire une fonction* tousDifferents *qui étant donné un tableau d'entiers renvoie* true *si tous les éléments du tableau sont différents et* false *si le tableau contient au moins deux éléments identiques.*
2. *Écrire une fonction* sommeN *qui étant donné un entier* n *supposé positif renvoie la somme des entiers de* 0 *à* n.
3. *Écrire une fonction* sousTab *qui prend comme arguments un tableau* tab *et une position* p *comprise entre* 0 *et* tab.length-1 *et renvoie le tableau des éléments situés strictement après* p *dans* tab.
   Exemple : *Pour le tableau* {12,3,2,8} *et la position* 0, *la fonction* sousTab *renverra le tableau* {3,2,8}, *pour la position* 2, *elle renverra le tableau* {8} *et pour la position* 3, *elle renverra le tableau de taille 0* {}.
4. *Écrire une fonction* toutesLesPaires *qui étant donnée un tableau d'entiers* tab *renvoie le tableau vide (de taille 0) si tous les éléments de* tab *ne sont pas tous différents et sinon renvoie un tableau de tableaux d'entiers correspondant à toutes les paires d'entiers de* tab .
   Exemple : *Pour le tableau* {12,3,2,8}, *la fonction renverra* {{12,3},{12,2},{12,8},{3,2},{3,8},{2,8}}.
   Aide : *Il est important de connaître la taille de du tableau renvoyé. Si le tableau* tab *est de taille* n, *le tableau produit est de taille* $0 + 1 + 2 + ... + (n - 1)$.
5. *Écrire une fonction* sommePaire *qui étant donnés un tableau d'entiers* tab *et un entier* x *renvoie :*
   — *Le tableau vide (de taille 0) si les éléments de* tab *ne sont pas tous différents.*
   — *Le tableau vide (de taille 0) s'il n'existe pas deux entiers* a *et* b *dans* tab *tels que* x=a+b.

— *Et dans les autres cas, un tableau de deux éléments* a *et* b *tels que* a *et* b *sont dans* tab *et vérifient* x=a+b.
Exemple : *Pour le tableau* {12,3,2,8} *et l'entier* 14, *la fonction* sommePaire *renverra le tableau* {12,2}.
Remarque : *Il peut y avoir plusieurs paires d'entiers* a *et* b *qui correspondent, la fonction en renverra alors une.*

□

Exercice 5 *La fourmi de Langton se déplace sur une grille bicolore, noire et blanche. Elle peut modifier la couleur d'une case de la grille lors de son déplacement et elle se déplace en fonction des couleurs qu'elle rencontre. Nous modélisons une grille par un tableau de tableaux d'entiers ne comportant que deux valeurs :* 0 *pour les cases noires et* 1 *pour les cases blanches. Au début la grille ne contient que des* 0, *c'est-à-dire que toutes les cases sont noires et la fourmi se trouve dans le coin supérieur gauche (case de coordonnées* $(0,0)$, *pour ligne* 0 *et colonne* 0*). Pour la suite si* gr *est une grille, l'entier situé à la ligne* i *et à la colonne* j *est dans* gr[i][j]. *De plus on supposera que la ligne 0 est en haut, la ligne 1 en-dessous, etc. et la colonne 0 est à gauche, la colonne 1 à sa droite, etc.*

La fourmi a le comportement suivant. Si elle se trouve sur une case noire, elle tourne de 90 degrés vers la gauche, elle change la case où elle se trouve en blanc et avance à la case suivante. Si elle se trouve sur une case blanche, elle tourne de 90 degrés vers la droite, elle change la case où elle se trouve en noir et avance à la case suivante. L'orientation de la fourmi sera donnée par une valeur entière comprise entre 0 et 3 :

— *0 pour dire que la fourmi est tournée vers le haut*
— *1 pour dire que la fourmi est tournée vers la droite*
— *2 pour dire que la fourmi est tournée vers le bas*
— *3 pour dire que la fourmi est tournée vers la gauche*

Ainsi si l'orientation de la fourmi est 0 (vers le haut) et qu'elle se trouve sur une case noire, son orientation deviendra 3, en revanche si elle se trouve sur une case blanche son orientation deviendra 1. Si son orientation est 1 (vers la droite) et qu'elle se trouve sur une case noire, son orientation deviendra 0. Au début la fourmi est orientée vers le bas.

La fourmi ne sort jamais de la grille. Si elle se trouve à la case de coordonnées $(0,3)$ *dans une grille avec* 10 *lignes, et qu'elle doit bouger vers le haut, elle passe à la case* $(9,3)$. *Ainsi la première et la dernière lignes sont considérées comme voisines, tout comme la première et la dernière colonnes.*

1. *Écrire une fonction* creerGrille *qui prend comme arguments deux entiers* n *et* m *et renvoie une grille avec* n *lignes et* m *colonnes où toutes les cases sont noires.*
2. *Écrire une fonction* afficheGrille *qui prend comme argument une grille* gr *et affiche son contenu sur le terminal ligne par ligne en séparant chaque case par un espace.*
3. *Écrire une fonction* nouvelleOrientation *qui prend comme arguments :*
   — *une grille* gr,
   — *un tableau de deux entiers* pos *dont le premier élément est un numéro valide de ligne de* gr *et le deuxième élément un numéro valide de colonne de* gr,
   — *un entier* o *représentant l'orientation d'une fourmi,*
   et qui renvoie un entier correspondant à la nouvelle orientation de la fourmi en supposant que celle-ci se trouve à la case de coordonnées pos *de la grille.*
   Exemple : nouvelleOrientation (gr ,{2,3},2) *renverra 1 si* gr [2][3] *vaut 0 et renverra 3 si* gr [2][3] *vaut 1.*
4. *Écrire une fonction* avanceEtChange *qui prend comme arguments :*
   — *une grille* gr,
   — *un tableau de deux entiers* pos *dont le premier élément est un numéro valide de ligne de* gr *et le deuxième élément un numéro valide de colonne de* gr,
   — *un entier* o *représentant l'orientation d'une fourmi,*
   et qui renvoie un tableau de deux entiers correspondant à la nouvelle position dans la grille de la fourmi après qu'elle ait avancé d'une case en suivant l'orientation donnée par o *depuis la case donnée par* pos. *Cette fonction changera également la couleur de la case de départ de noir à blanc ou de blanc à noir.*
   Exemple : avanceEtChange(gr,{2,3},1) *renverra* {2,4} *si* gr [2]. length *est strictement plus grande que 4 et si* gr [2][3] *vaut 0, sa nouvelle valeur après exécution sera 1. En revanche si* gr [2]. length *est égale à 4, la fonction renverra* {2,0}.
5. *Écrire une fonction* simulation *qui prend en arguments un entier* k, *un entier* n *et un entier* m *et qui simule* k *étapes de la fourmi de Langton sur la grille avec* n *lignes et* m *colonnes et affiche à la fin la grille obtenue. Pour rappel au début toutes les cases de la grille sont noires, la fourmi se trouve dans le coin en haut à gauche et son orientation est vers le bas. À chaque étape, la fourmi calcule sa nouvelle orientation, change la couleur de la case où elle se trouve et se déplace d'une case en suivant sa nouvelle orientation.*

□