Introduction à la Programmation **Java** (IP1-Java)
Examen – **Durée : 2 heures**

Université Paris-Diderot – Samedi 8 Février 2020

- Aucun document ni aucune machine ne sont autorisés. Les téléphones doivent être rangés.
- Les exercices sont tous indépendants.
- 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.
- Dans les énoncés, on propose parfois une liste de fonctions et procédures utilisables. Vous pouvez cependant utiliser tout équivalent JAVA (Par exemple, `s.length()` à la place de `stringLength(s)`). Attention cependant à ne pas utiliser des fonctions de la bibliothèque standard qui n'auraient pas été abordées en cours.
- Le sujet est long et il est possible qu'il soit noté sur plus que 20.

Exercice 1

1. *Si à la fin de l'exécution le tableau* tab2 *vaut* {2,3,4,1}, *que doit valoir* tab1 *au début ?*

```
public class Test{
  public static void f(int[]t){
    int tmp=0;
    for(int i=0;i<t.length/2;i=i+1){
      tmp=t[i];
      t[i]=t[t.length-1-i];
      t[t.length-1-i]=tmp;
    }
  }

  public static int[] g(int[]t){
    int[] t2=new int[t.length];
    for(int i=1;i<t.length;i=i+1){
      t2[i]=t[i-1];
    }
    t2[0]=t[t.length-1];
    return t2;
  }

  public static void main(String[] args){
    int []tab1 =...;
    f(tab1);
    int []tab2 = g(tab1);
  }
}
```

2. *Donner la valeur de la variable* m *après exécution du programme suivant.*

```
public class Test2{
  public static int fo(int x){
    int n=0;
    int a=1;
    while(a<x){
      n=n+1;
      a=go(2,n);
    }
    return n;
  }

  public static int go(int c,int d){
    int res=1;
    for(int i=0;i<d;i=i+1){
      res=res*c;
```

```
    }
    return res;
  }

  public static void main(String[] args){
    int m = fo(18);
  }
}
```

□

Exercice 2 *Dans un fichier* Mystere.java, *il y a une fonction* public static int f(int x) *que vous ne connaissez pas, mais à laquelle vous avez accès. Pour l'appeler, avec l'argument 2 par exemple, il suffit de faire* Mystere.f(2). *Dans la suite de l'énoncé, on appellera simplement* f *cette fonction.*

1. *Écrire une fonction* minF *qui prend en arguments deux entiers* a *et* b *tels que* a<b *et renvoie la valeur minimale prise par* f *entre* a *et* b *inclus, c'est à dire qu'elle renvoie* $min(f(x) \mid a \leq x \leq b)$.
2. *Écrire une fonction* zerosF *qui prend en arguments deux entiers* a *et* b *tels que* a<b *et renvoie un tableau d'entiers contenant les entiers* v *compris entre* a *et* b *inclus tels que* f(v) *est égal à 0.*
3. *Écrire une fonction* composeF *qui prend en arguments deux entiers* n *(avec* n>0*) et* x *et qui renvoie* $f^{(n)}(x)$. *On rappelle que* $f^{(1)}(x)=f(x)$ *et* $f^{(i+1)}(x)=f(f^{(i)}(x))$ *pour tout* i>0.
4. *Pour un entier* $n > 0$, *on définit la suite* $(U_i^n)_{i\geq 0}$ *de degré* n *associée à* f *de la façon suivante. Pour tout entier* $0 \leq i < n$, *on a* $U_i^n = f(i)$ *et pour tout entier entier* $i \geq n$, *on a* $U_i^n = \Sigma_{j=1}^{j=n} U_{i-j}^n = U_{i-1}^n + U_{i-2}^n + \ldots + U_{i-n}^n$. *Écrire une fonction* suiteF *qui prend comme arguments deux entiers* n *et* i *avec* $n > 0$ *et* $i \geq 0$ *et qui renvoie la valeur de* U_i^n.

□

Exercice 3 *Dans cet exercice, nous allons proposer des questions pour programmer une liste de courses interactive. Cette liste sera représentée par un tableau de chaînes de caractères. Le début de la liste sera en position 0 et la fin de la liste sera la dernière position du tableau. Quand un utilisateur voudra rajouter un élément à la liste, il le mettra au début de la liste et quand il voudra retirer un élément de la liste il retirera le dernier élément. Ces deux opérations produiront une nouvelle liste. Pour les questions suivantes, vous pouvez utiliser : «* boolean stringEquals (String s1, String s2) *» qui renvoie* true *si les deux chaînes* s1 *et* s2 *sont égales et* false *sinon.*

1. *Écrire une fonction (procédure)* afficheListe *qui prend en argument une liste de courses (i.e. un tableau de chaînes de caractères) et l'affiche dans le terminal en allant à la ligne après chaque élément.*
2. *Écrire une fonction* creerListe *qui ne prend pas d'argument et renvoie une liste de courses vide (un tableau de chaînes de caractères de taille 0) .*
3. *Écrire une fonction* insereListe *qui prend comme arguments une liste de courses et une chaîne de caractères et renvoie une nouvelle liste où la chaîne a été ajoutée au début de la liste donnée. Par exemple, si l'on a* int [] lis ={"Tomates","Carottes"}, *alors* insereListe (lis ,"Poivrons") *renverra la liste* {"Poivrons","Tomates","Carottes"}.
4. *Écrire une fonction* retireListe *qui prend comme argument une liste de courses et renvoie une nouvelle liste où la dernière chaîne de caractères a été supprimée (si la liste donnée est vide, la fonction renverra la liste vide). Par exemple si l'on a* int [] lis ={"Tomates","Carottes"}, *alors* retireListe (lis) *renverra la liste* {"Tomates"}.
5. *Écrire une fonction* retireElement *qui prend comme arguments une liste de courses et une chaîne de caractères et renvoie une nouvelle liste dans laquelle la chaîne donnée a été retirée. Attention, ici une chaîne peut apparaître plusieurs fois dans la liste. Si la chaîne n'apparaît pas dans la liste, alors on renvoie la liste donnée en argument inchangée. Par exemple si l'on a* int [] lis ={"Tomates","Carottes","Tomates"}, *alors* retireElement (lis ,"Tomates ") *renverra la liste* {"Carottes"}.

□

Exercice 4 *Dans cet exercice, on veut générer des grilles du jeu de démineur. Dans ce jeu on a une grille où sont placées des mines et sur les cases autour des mines on indique combien de mines sont voisines. Nous représenterons une grille de démineur par un tableau de tableaux d'entiers. Les mines seront représentées par des -1. Et dans les autres cases, on aura le nombre de mines voisines, c'est-à-dire un entier entre 0 et 4 (on regarde les voisins haut, bas, gauche et droite mais pas les diagonaux). L'exemple ci-dessous montre une grille avec quatre mines. Cette grille est représentée par le tableau de tableaux d'entiers* {{−1,2,−1,3,−1},{1,0,2,−1,2},{0,0,0,1,0}}.

-1	2	-1	3	-1
1	0	2	-1	2
0	0	0	1	0

1. *Écrire une fonction* caseCorrecte *qui prend comme arguments un tableau de tableaux d'entiers* tab, *un numéro de ligne* lig *et un numéro de colonne* col *et renvoie* true *si dans la case* tab[lig][col] *il y a un entier égal à -1 ou un entier dont la valeur est comprise entre 0 et 4 et qui correspond au nombre de voisins égaux à -1 (pour rappel, on ne regarde pas les diagonales) et dans les autres cas, la fonction renvoie* false . *On suppose ici, que* tab *encode une grille, i.e. un tableau de tableaux d'entiers où tous les sous-tableaux ont la même taille et que les valeurs* lig *et* col *ne sortent pas des limites de* tab. *Par exemple, si l'on a* int [][] tab$=\{\{-1,2,-1,3,-1\},\{1,1,1,-1,2\},\{0,0,0,1,0\}\}$, *alors* caseCorrecte(tab ,0,1) *renvoie* true caseCorrecte(tab ,1,1) *renvoie* false .
2. *Écrire une fonction* estDemineur *qui prend comme argument un tableau de tableaux d'entiers et renvoie* true *si il correspond à une grille de démineur et* false *sinon. Pour correspondre à une grille de démineur, le tableau doit avoir tous ses sous-tableaux de la même taille, et chaque case des sous-tableaux doit être correcte comme décrit à la fonction précédente.*
3. *Écrire une fonction (procédure)* afficheDemineur *qui prend comme argument un tableau de tableaux d'entiers et si il correspond à une grille de démineurs, l'affiche ligne par ligne en séparant chaque case (sur une même ligne) par des espaces et en mettant des X à la place des mines, des espaces à la place des 0 et sinon le contenu de la case. Si le tableau de tableaux ne correspond pas à une grille de démineur, la fonction n'affiche rien. Par exemple, si* int [][] tab$=\{\{-1,2,-1,3,-1\},\{1,0,2,-1,2\},\{0,0,0,1,0\}\}$, *alors* afficheDemineur(tab) *affichera :*

   X 2 X 3 X
   1 2 X 2
   1

4. *Écrire une fonction* grilleMine *qui prend comme arguments trois entiers (supposés positifs)* L, C *et* M *et si* M *est inférieur ou égal à* L*C *renvoie un tableau de tableaux d'entiers correspondant à une grille de démineurs où seulement les mines ont été placées (c'est une grille qui ne contient que des 0 ou des -1) de* L *lignes et* C *colonnes et qui contient exactement* M *mines placées au hasard. On pourra se servir d'une fonction* int alea(int a, int b) *qui renvoie un entier au hasard supérieur ou égal à* a *et strictement inférieur à* b. *Par exemple, une sortie possible de* grilleMine (3,2,4) *est le tableau de tableaux d'entiers* $\{\{-1,-1\},\{-1,0\},\{0,-1\}\}$. *Si* M *est strictement supérieur à* L*C, *la fonction ne renvoie rien.*
5. *Écrire une fonction* creerGrille *qui prend comme arguments trois entiers (supposés positifs)* L, C *et* M *et si* M *est inférieur ou égal à* L*C *renvoie un tableau de tableaux d'entiers correspondant à une grille de démineurs (cette fois-ci le tableau contient les chiffres indiquant le nombre de voisins qui sont une mine) de* L *lignes et* C *colonnes et qui contient exactement* M *mines placées au hasard. Là encore, si* M *est strictement supérieur à* L*C, *la fonction ne renvoie rien. Par exemple, une sortie possible de* grilleMine (3,2,4) *est le tableau de tableaux d'entiers* $\{\{-1,-1\},\{-1,3\},\{2,-1\}\}$.

□

Exercice 5 *Dans cet exercice, on s'intéresse au fonctionnement d'emprunt de livres d'une bibliothèque et en particulier aux dates d'emprunt et de retour. Une date sera encodée par un tableau à trois entiers, le premier correspondant au jour, le second au mois et le troisième à l'année. Pour faciliter cet exercice on ne prendra pas en compte les années bissextiles, c'est à dire que l'on supposera que le mois de février a toujours 28 jours. L'encodage d'une date sera celui classique, ainsi le 3 février 2006 sera encodé par le tableau d'entier* $\{3,2,2006\}$. *On rappelle que les mois ayant 31 jours sont janvier, mars, mai, juillet, aout, octobre et décembre, les autres mois ont 30 jours sauf février qui en a donc 28.*

1. *Écrire une fonction* estDate *qui prend comme arguments un tableau d'entiers et renvoie* true *si il encode une date et* false *sinon. Il faut donc vérifier que le tableau est de taille 3 et que chacune de ses données correspond à un numéro de jour, de mois et d'année (les années doivent de plus être strictement positives). Par exemple, si* int [] tab $=\{30,2,2004\}$, *la fonction* estDate(tab) *renverra* false *car le 30 février ne peut pas exister (vu que l'on a supposé qu'il y avait 28 jours en février).*
2. *La période d'emprunt d'un livre à la bibliothèque est de maximum 21 jours. Écrire une fonction* finEmprunt *qui prend en arguments un tableau d'entiers correspondant à une date d'emprunt et renvoie un tableau d'entiers correspondant à la date de retour. Par exemple, pour* int [] tab$=\{2,2,2004\}$, *l'appel* finEmprunt(tab) *renverra le tableau* $\{23,2,2004\}$, *pour* int [] tab$=\{20,3,2010\}$, *l'appel* finEmprunt(tab) *renverra le tableau* $\{10,4,2010\}$ *et pour* int [] tab$=\{28,12,2010\}$, *l'appel* finEmprunt(tab) *renverra le tableau* $\{18,1,2011\}$.
3. *Écrire une fonction* estApres *qui prend en arguments deux dates (encodées dans des tableaux d'entiers) et renvoie* true *si la deuxième date est strictement après la première et* false *sinon.*
4. *Écrire une fonction* nbreJours *qui prend en arguments deux dates (encodées dans des tableaux d'entiers) et renvoie le nombre de jours entre ces deux dates (les jours de ces dates étant aussi comptés) si la deuxième date est strictement après la première et 0 sinon. Par exemple si l'on a* int [] date1$=\{2,3,2015\}$ *et* int [] date2$=\{8,3,2015\}$ *alors* nbreJours(date1,date2) *renverra* 7. *On rappelle que comme on suppose que le mois de février a toujours 28 jours, une année compte donc 365 jours.*

□