# Introduction à la Programmation **Java** (IP1-Java)
# Examen Session 2 – **fait en distanciel**

Université de Paris – Mardi 23 juin 2020

**MODALITÉS DE RENDU**

— Les réponses sont à renvoyer par mail à sangnier@irif.fr avant le **mercredi 24 juin 2020 à 10h00**
— Les formats acceptés sont pdf, jpg ou png
— Tout mail où ne figure pas le nom, prénom et numéro d'étudiant risque de ne pas être pris en compte

— 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.

**Exercice 1 (Lapins et renards)**

*On souhaite modéliser un écosystème où cohabitent des lapins et des renards. On note $R_n$ le nombre de lapins pour l'année $n$ et $F_n$ le nombre de renards pour l'année $n$.*

*Le nombre de lapins pour l'année $n+1$ est proportionnel à ce nombre pour l'année $n$ mais aussi au nombre de renards pouvant manger ces lapins lors de l'année $n$ :*

$$R_{n+1} = max(0, (2 * R_n) - F_n)$$

*Quant aux renards, leur nombre pour l'année $n+1$ dépend du nombre de lapins à manger et de leur nombre pour l'année $n$ :*

$$F_{n+1} = ((F_n * F_n) + (F_n/2)) * R_n/(max(1, F_n))$$

*Dans les formules précédentes, l'opérateur / correspond à la division entière. Dans les questions suivantes, on pourra utiliser la fonction « public static int max (int x, int y) » qui attend deux entiers x et y et retourne le maximum de ces deux entiers.*

1. *Écrire une fonction* public static int rabbits (int pastRabbits, int pastFoxes) *qui attend un entier* pastRabbits *représentant $R_n$, un entier* pastFoxes *représentant $F_n$ et qui retourne $R_{n+1}$. Par exemple,* rabbits (2, 1) *retourne* 3 *tandis que* rabbits (1, 3) *retourne* 0.
2. *Écrire une fonction* public static int foxes (int pastRabbits, int pastFoxes) *qui attend un entier* pastRabbits *représentant $R_n$, un entier* pastFoxes *représentant $F_n$ et qui retourne $F_{n+1}$. Par exemple,* foxes (2, 1) *retourne* 2 *tandis que* foxes (1, 3) *retourne* 3.
3. *En utilisant les fonctions issues des deux questions précédentes, écrire une fonction* public static int simulateRabbits (int initialRabbits , int initialFoxes , int n) *qui attend un entier* initialRabbits *représentant $R_0$, un entier* initialFoxes *représentant $F_0$, un entier positif* n *et qui retourne la taille de la population de lapins de l'écosystème après* n *années, c'est-à-dire la valeur $R_n$. On introduira quatre variables entières pour représenter $R_n$, $R_{n+1}$, $F_n$ et $F_{n+1}$ à chaque itération du calcul. Par exemple,* simulateRabbits (2, 1, 2) *retourne* 4 *tandis que* simulateRabbits (1, 3, 2) *retourne* 0.

□

**Exercice 2**

*Soit $n$ un entier positif. Dans cet exercice, on représente des relations amoureuses entre des individus numérotés de 0 à $n-1$. Pour cela, on utilise un tableau* t *d'entiers tel que la longueur de* t *soit* n *et tels que si* t[i] = j *alors* i *est amoureux de* j.

1. *On appelle "couple" une paire d'individus distincts* $(i, j)$ *tels que* $i$ *est amoureux de* $j$ *et* $j$ *est amoureux de* $i$. *Écrire une fonction* nbCouples *qui prend en paramètre un tableau* t *tel que* t *représente des relations amoureuses comme décrit dans le paragraphe précédent. Cette fonction renvoie le nombre de couples de* t.
   *Par exemple, si* t *vaut* {1,0,3,2,6,0,4} *alors* nbCouples(t) *renvoie* 3.
2. *Écrire une fonction* loved *qui prend en paramètre un tableau* t *tel que* t *représente des relations amoureuses comme décrit plus haut. Cette fonction renvoie le numéro de l'individu le plus aimé.*
   *Par exemple, si* t *vaut* {1,0,3,2,0,0} *alors* loved(t) *renvoie* 0.
3. *On appelle "triangle amoureux" un triplet d'individus distincts deux à deux* $(i, j, k)$ *tels que* $i$ *aime* $j$, $j$ *aime* $k$ *et* $k$ *aime* $i$. *Écrire une fonction* nbTriangles *qui prend en paramètre un tableau* t *tel que* t *représente des relations amoureuses comme décrit dans le paragraphe précédent. Cette fonction renvoie le nombre de triangles amoureux de* t.
   *Par exemple, si* t *vaut* {1,2,0,5,3,4,1,3} *alors l'appel* triangle (t) *renvoie* 2.

□

**Exercice 3**

*Un tableau d'entiers est* trié croissant *si chacun de ses éléments, mis à part le premier, est supérieur ou égal à l'élément qui le précède.*

*Un tableau d'entiers est* zig-zag *si chacun de ses éléments d'indice pair, mis à part le premier, est inférieur ou égal à l'élément qui le précède, et chacun de ses éléments d'indice impair est supérieur ou égal à l'élément qui le précède. Par exemple* {1,3,2,4} *est zig-zag, et* {1,2,3} *n'est pas zig-zag.*

1. *Écrire une fonction* estTri *qui prend en argument un tableau d'entiers et renvoie* true *s'il est trié croissant et* false *sinon.*
2. *Écrire une fonction* estZigZag *qui prend en argument un tableau d'entiers et renvoie* true *s'il est zig-zag et* false *sinon.*
3. *Écrire une fonction* minSuivant *qui prend en arguments un tableau d'entiers* tab *et un indice entier* ind *compris entre* 0 *et* tab.length−1 *et qui cherche l'indice du plus petit élément du tableau situé entre l'indice* ind *et* tab.length−1. *Par exemple pour le tableau* int []tab={1,4,3,2,5}, *on a* minSuivant(tab,1)=3 *car entre les positions* 1 *et* 4, *la plus petite valeur du tableau est* 2 *et elle se trouve à l'indice* 3.
4. *Écrire une fonction* maxSuivant *qui prend en arguments un tableau d'entiers* tab *et un indice entier* ind *compris entre* 0 *et* tab.length−1 *et qui cherche l'indice du plus grand élément du tableau situé entre l'indice* ind *et* tab.length−1. *Par exemple pour le tableau* int []tab={6,5,1,2,3}, *on a* maxSuivant(tab,1)=1 *car entre les positions* 1 *et* 4, *la plus grande valeur du tableau est* 5 *et elle se trouve à l'indice* 1.
5. *Écrire une fonction* echange *qui prend en arguments un tableau d'entiers* tab *et deux indices entiers* i *et* j *compris entre* 0 *et* tab.length−1 *et elle échange leurs valeurs dans le tableau. Cette fonction ne renvoie rien mais modifie le tableau. Par exemple, si l'on a* int []tab={1,4,3,2,5}, *après exécution de la fonction* echange(tab,2,4), *le tableau* tab *vaut* {1,4,5,2,3}.
6. *En vous servant des fonctions précédentes, écrire une fonction* faitZigZag *qui prend en argument un tableau d'entiers* tab *et le modifie de telle manière qu'à la fin de l'exécution de la fonction, le tableau* tab *est zig-zag et ses éléments sont les mêmes qu'avant l'exécution. Cette fonction ne renvoie rien. L'idée est la suivante, pour chaque indice* ind *(en commençant à l'indice 0) du tableau, si l'indice est pair on cherche le plus petit élément dans la tableau à partir de* ind *et on l'échange avec l'élément à l'indice* ind. *Si l'indice est impair, on fait la même chose en cherchant le plus grand élément.*

□

**Exercice 4** *On représente un dictionnaire des synonymes par un tableau de tableaux de chaînes de caractères. À chaque case de ce tableau, on trouve donc un tableau de chaînes de caractères dont le premier élément correspond à un mot et les autres éléments à ses synonymes. Par exemple le dictionnaire des synonymes suivant* String [][] dict={{"Football"},{" Frigidaire ","Glacière","Réfrigérateur"},{"Réfrigérateur"," Frigidaire ","Glacière"}} *nous dit que le mot Football n'a pas de synonyme, le mot Frigidaire a deux synonymes qui sont Glacière et Réfrigérateur et le mot Réfrigérateur a deux synonymes qui sont Frigidaire et Glacière. Pour les questions suivantes, vous pouvez utiliser la fonction : « 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* afficheDictionnaire *qui prend en argument un dictionnaire des synonymes et affiche ligne par ligne les mots, suivi de deux points, suivi par la liste des synomymes qui lui sont associés séparés par des virgules. Cette fonction ne renvoie rien. Sur l'exemple,* afficheDictionnaire (dict) *affichera :*

```
1 Football :
2 Frigidaire : Glacière, Réfrigérateur
3 Réfrigérateur : Frigidaire, Glacière
```

2. *Écrire une fonction* afficheSynonyme *qui prend en arguments un dictionnaire des synonymes et un mot et affiche sur une ligne ces synonymes séparés par des virgules. Si le mot n'a pas de synonyme, la fonction n'affiche rien. Sur l'exemple,* afficheSynonyme(dict,"Frigidaire") *affichera :*

```
Glacière, Réfrigérateur
```

3. *On veut vérifier qu'un dictionnaire des synonymes est symétrique, c'est à dire que si un mot mot2 est dans la liste des synonymes d'un mot mot1 alors mot2 est dans le dictionnaire et mot1 dans la liste des synonymes de mot2. Écrire une fonction* estSymetrique *qui prend comme argument un dictionnaire des synonymes et renvoie* true *si il est symétrique et* false *sinon. Si par exemple, on prend le dictionnaire des synonymes* String [][] dict2={{"Balle","Ballon","Boule"},{"Ballon","Balle","Boule"},{"Boule","Balle"}}, *il n'est pas symétrique car Boule est synonyme de Ballon, mais Ballon n'est pas dans la liste des synonymes de Boule. L'appel* estSymetrique(dict2) *renverra alors* false.

4. *On veut vérifier si le dictionnaire respecte l'ordre lexicographique (l'ordre classique des dictionnaires). Pour cela vous pourrez vous servir d'une fonction « boolean* estAvant (String s1, String s2) *» qui renvoie* true *si la chaîne* s1 *est avant la chaîne* s2 *dans l'ordre lexicographique et* false *sinon. Par exemple,* estAvant("Ballon","Balle") *renvoie* false *car Balle est avant Ballon. Écrire une fonction* estLexicographique *qui prend comme argument un dictionnaire de synonymes est renvoie* true *si les premiers mots de chaque tableau sont classés dans l'ordre lexicographique et* false *sinon (on ne vérifie pas que la liste des synonymes soit dans cet ordre) . Par exemple, si l'on a* String [][] dict3={{"Balle","Ballon","Boule"},{"Ballon","Balle","Boule"},{"Football"},{"Boule","Balle"}}, *l'appel à* estLexicographique(dict3) *renverra* false *car Football devrait venir après Boule.*

□

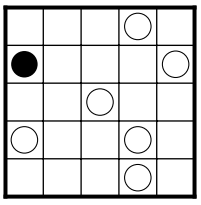
**Exercice 5** *L'objectif du puzzle du Masyu est de trouver un circuit qui passe par tous les cercles disposés sur une grille. Ce circuit doit vérifier les trois contraintes suivantes :*

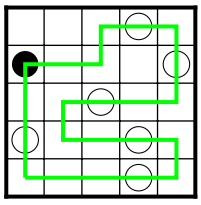
*(i) Quand il arrive sur un cercle noir, il faut que le chemin tourne d'un angle droit à gauche ou à droite.*

*(ii) Quand il arrive sur un cercle blanc, le circuit doit aller tout droit mais doit tourner à angle droit à gauche ou à droite dans au moins une des cases qui entoure ce cercle blanc.*

*(iii) En dehors des deux cas précédents, le chemin du circuit ne peut pas changer de direction. En d'autres termes, en dehors des cercles noirs et des cases qui entourent les cercles blancs, le circuit va tout droit.*

*Voici un exemple de grille (à gauche) et une solution pour cette grille (à droite) :*





*On représente une grille à l'aide d'un tableau de tableaux d'entiers* t. *Il y a un cercle blanc à la position* $(i, j)$ *(où* $i$ *est le numéro de ligne et* $j$ *le numéro de colonne) de la grille si* t[i][j] *vaut* $1$. *Il y a un cercle noir à la position* $(i, j)$ *de la grille si* t[i][j] *vaut* $2$. *Sinon,* t[i][j] *vaut* $0$ *s'il n'y a aucun cercle à la position* $(i, j)$. *Les lignes de la grille sont numérotées du haut vers le bas (la ligne la plus au nord est numérotée* $0$*). Sur chaque ligne, la case d'indice le plus petit se situe à gauche et celle d'indice le plus grand se situe à droite.*

*On représente une direction à l'aide d'un tableau de deux entiers. Le premier entier* $di$ *est pris dans l'ensemble* $\{-1, 0, 1\}$ *et il correspond à un déplacement sur les lignes. Le second entier* $dj$ *appartient aussi à l'ensemble* $\{-1, 0, 1\}$ *et il correspond à un déplacement sur les colonnes. Quand on est situé à la position* $(i, j)$ *et que l'on suit la direction* $(di, dj)$ *alors la prochaine position est* $(i + di, j + dj)$.

*Un circuit est représenté par un tableau de tableaux de deux entiers* c. *La première case de* c *est la coordonnée de départ du circuit et les cases suivantes sont les directions suivies par le circuit. Ainsi, le circuit de la figure est représenté par le tableau* { {0, 3}, {0, 1}, {1, 0}, {1, 0}, {0, −1}, {0, −1}, {0, −1}, {1, 0}, {0, 1}, {0, 1}, {0, 1}, {1, 0}, {0, −1}, {0, −1}, {0, −1}, {0, −1}, {−1, 0}, {−1, 0}, {−1, 0}, {0, 1}, {0, 1}, {−1, 0}, {0, 1} }.

1. *Écrire une fonction* circuitPositions *qui prend en paramètre un circuit comme décrit plus haut. Cette fonction renvoie un tableau de tableaux de deux entiers qui correspondent aux coordonnées des cases successives où passe le circuit en prenant son point de départ comme première de ces coordonnées. Remarque : Il n'est pas nécessaire de vérifier que le circuit contient des directions valides.*

2. *Écrire une fonction* isCircleCircuit *qui prend en paramètre un circuit* c *comme décrit plus haut ainsi qu'une grille* g *représentée comme décrit plus haut. Cette fonction renvoie un booléen qui vaut vrai si et seulement si la première et la dernière coordonnées du circuit sont égales, si le circuit ne sort pas de la grille et enfin si le circuit passe par tous les cercles de la grille.*

3. *Écrire une fonction* constraintsRespected *qui prend en paramètre un circuit comme décrit plus haut ainsi qu'une grille* g *représentée comme décrit plus haut. Cette fonction renvoie un booléen qui vaut vrai si et seulement si toutes les contraintes* $(i)$, $(ii)$ *et* $(iii)$ *sont vérifiées par le circuit.*
4. *Écrire une fonction* validAnswer *qui prend en paramètre un circuit* c *comme décrit plus haut ainsi qu'une grille* g *représentée comme décrit plus haut. Cette fonction renvoie le booléen vrai si et seulement si le circuit* c *est une solution valide pour la grille* g.

□