Scala

Sudoku : multi paradigme

Vérifiez les conventions à suivre pour l'écriture du code (cf Moodle).

Vous allez réaliser ce TD en Scala : dans un premier temps d'une manière hybride impérative et fonctionnelle et dans le TD suivant, d'une manière purement fonctionnelle.

Le TD peut tenir en un seul fichier : vous pouvez définir plusieurs classes et objets dans un même fichier scala. Faites un jar lançable par scala –jar (ne mettez pas les librairies scala dans un jar Java).

Vous indiquerez EXPLICITEMENT les types de retours des méthodes et des fonctions dans les déclarations systématiquement.

Vous allez réaliser un programme pour créer des grilles sudoku et/ou bien trouver toutes les solutions à un problème sudoku.

Rappel : le jeu de sudoku est représenté par une grille qu'il faut compléter de telle manière que les chiffres de 1 à 9 sont représentés une et une seule fois sur chaque ligne, chaque colonne et dans chaque cadre 3x3.

Exemple :

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Une solution :

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Vous allez fabriquer un générateur de grilles (il suffit de produire une ou plusieurs grilles, puis d'enlever de manière aléatoire des chiffres) ainsi qu'un système qui cherche toutes les solutions à une grille.

Dans un premier temps, vous allez coder de manière classique hybride impérative/fonctionnelle (multi paradigme):

Fabriquez une class Sudoku qui prend en paramètre un tableau d'entiers 9x9

class Sudoku(startConfig_ : Array[Array[Int]])

Fabriquez la méthode :

def solver(): List[Array[Array[Int]]]

qui résoud le problème représenté sous la forme d'une matrice d'entier : le résultat renvoyé est une matrice de solution Sudoku qui vérifie les contraintes indiquées.

En entrée, une valeur 0 dans **startConfig** indique que la case est vide. Créez une matrice locale **workGrid** à la méthode **solver** comme variable de travail (qui sert à envisager des remplissages)

La méthode **solver** lance le traitement suivant :

fillXY(0,0)

la méthode **fillXY** commence à remplir la matrice avec des chiffres à partir de la position (0,0). Elle utilise la méthode (que vous définirez) :

def isPossibleAt(number_ : Int, x_ : Int, y_ : Int): Boolean

qui renvoie vrai s'il est possible de placer le numero **number**_ à la position **x**_, **y**_ (c'est-à-dire s'il ce nombre n'est pas déjà dans la ligne x_ , dans la colonne y_ ou dans le quadrant 3x3 – vous pouvez créer des fonctions pour faire ces tests).

1. Résolvez la grille proposée dans l'énoncé.
2. Proposez une fonction capable de générer ce type de grilles.