# TP 1 : Solveur Sudoku

IFT2015 - Structures de données – A24

## Objectif

Développer un programme orienté objet pour résoudre un puzzle de Sudoku. Étant donné une grille de 9x9 qui peut avoir des cellules déjà remplies de chiffres entre 1 et 9, votre tâche consiste à compléter la grille de sorte que chaque ligne, chaque colonne et chacune des neuf sous-grilles 3x3 contiennent les chiffres de 1 à 9 exactement une fois.

## Description du problème

Vous allez implémenter un solveur de Sudoku qui lit un ou plusieurs puzzles 9x9 avec certaines cellules déjà remplies de chiffres de 1 et 9, tandis que d'autres sont vides (représentées par 0). Votre programme doit résoudre le Sudoku en remplissant les cellules vides et produire la solution du puzzle en sortie.

### *Règles du Sudoku*

- Chaque chiffre de 1 à 9 doit apparaître exactement une fois dans chaque ligne.
- Chaque chiffre de 1 à 9 doit apparaître exactement une fois dans chaque colonne.
- Chaque chiffre de 1 à 9 doit apparaître exactement une fois dans chaque sous-grille 3x3.

Nous vous fournissons une série de tests (**`SudokuApp`**) qui exécutent et vérifient la solution pour plusieurs puzzles. Vous devez évidemment réussir ces tests de la classe principale, **`SudokuApp.java`** , pour avoir tous vos points à l'exécution (voir barème).

## Formats d'entrée-sortie

Les tests sont composés de tableaux de `int` à 2 dimensions d'exactement 9 lignes par 9 colonnes (voir **`SudokuApp.java`**). Les entrées sont des chiffres de 0 à 9, 0 indiquant une case du puzzle non remplie.

En sortie, si le puzzle possède une solution, on imprime en **`stdout`** une ligne indiquant que le puzzle a été solutionné suivie de la solution de 9 lignes contenant 9 chiffres de 1 à 9 séparés par des espaces. Si le puzzle n'a pas de solution, on imprime un message indiquant cette situation, par exemple "No solution exists". Vous devez aussi détecter si la taille du puzzle est correct, 9x9. Si le puzzle n'est pas de la bonne taille, vous pouvez imprimer un message pour l'indiquer, par exemple "dimensions illégales".

## *Exemple d'entrée :*

```
Integer[][] puzzle = {
{5, 3, 0, 0, 7, 0, 0, 0, 0},
{6, 0, 0, 1, 9, 5, 0, 0, 0},
{0, 9, 8, 0, 0, 0, 0, 6, 0},
{8, 0, 0, 0, 6, 0, 0, 0, 3},
{4, 0, 0, 8, 0, 3, 0, 0, 1},
{7, 0, 0, 0, 2, 0, 0, 0, 6},
{0, 6, 0, 0, 0, 0, 2, 8, 0},
{0, 0, 0, 4, 1, 9, 0, 0, 5},
{0, 0, 0, 0, 8, 0, 0, 7, 9}
};
```

## *Exemple de sortie :*

```
Test Case 1: Sudoku Solved:
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
```

# Structure et exigences

Vous devez suivre les principes de la programmation orientée objet (POO) et des ADT (Abstract Data Types) pour ce projet. Les interfaces et classes ainsi que leurs contrats sont décrites ci-dessous :

## *Interfaces et classes obligatoires*

### ➡*interface GameBoard<T> :*

Représente l'abstraction d'une grille de jeu, une matrice avec des indices de types **`int`** et des valeurs de type générique **`<T>`**.

Exige les méthodes suivantes (voir **`GameBoard.java`**) :

```
T getCell( int x, int y )
void setCell( int x, int y, T value )
int getWidth()
int getHeight()
void display()
```

L'implémentation du **`GameBoard`** devra se nommer **`IntegerBoard`** car le jeu de Sudoku utilise une grille de chiffres qu'on implémente avec des **`Integer`** :

- Représente la grille de Sudoku 9x9
- Fournit les méthodes de l'interface **`GameBoard`** pour :
    - Obtenir et définir les dimensions de la grille
    - Obtenir et définir les valeurs dans les cellules individuelles
    - Afficher la grille
    - Cloner la grille actuelle pour faciliter l'encapsulation

### ➡*interface GameSolver :*

Représente l'abstraction d'un joueur capable de résoudre et d'imprimer la solution.

Exige les méthodes suivantes (voir **`GameSolver.java`**) :

```
boolean solve()
void printSolution()
```

L'implémentation du **`GameSolver`** s'appelle **`SudokuSolver`** et met en œuvre la logique de résolution du casse-tête Sudoku en implémentant les méthodes exigées par **`GameSolver`**. Il implémente l'interface **`GameSolver`** et implémente les méthodes **`solve()`** et **`printSolution()`**. Il crée la solution si elle existe et s'assure qu'elle respecte les règles du jeu. Votre programme doit résoudre tous les puzzles de Sudoku dans un fichier **`SudokuApp.java`**. Ce fichier sera modifier lors de la correction de vos codes.

### ➡*class SudokuApp :*

Implémente une suite de tests et vérifie les fonctionnalités du joueur.

# Makefile

Un fichier **`Makefile`** est fourni pour simplifier la compilation et l'exécution du programme et tests initiaux si vous utilisez Unix.

# Code et soumission

La date limite de soumission est le 17 novembre 2024 à 23h59. Soumettez votre solution sur StudiUM sous forme de fichier ZIP avec les considérations suivantes :

- Ce projet peut être réalisé en équipe de deux personnes maximum. Vous êtes encouragés à travailler ensemble pour concevoir, mettre en œuvre et tester la solution.
- Tous les fichiers de code source doivent être dans des fichiers **`.java`** séparés dans une archive zip :
  - **`GameBoard.java`**
  - **`IntegerBoard.java`**
  - **`GameSolver.java`**,
  - **`SudokuSolver.java`**
  - **`SudokuApp.java`**
- Soumettez votre archive et seuls les formats et noms de fichier suivants seront acceptés: `TP1_f1_f2.tgz` ou `TP1_f1_f2.zip`, où `f1` et `f2` sont les noms de famille du premier et du deuxième membre de l'équipe, ou si vous travaillez seul, utilisez `TP1_f.tgz` ou `TP1_f.zip`, où `f` est votre nom de famille.
- Les soumissions tardives entraîneront une pénalité de 20 % par période de 24, débutant à 00h00 le 18 novembre 2024.

# Suggestion d'étapes à suivre

Pour vous assurer que votre programme fonctionne correctement, suivez les étapes suivantes :

1. Naviguez jusqu'au répertoire contenant vos fichiers et assurez-vous qu'il contient le `Makefile` et les fichiers **`GameBoard.java`**, **`IntegerBoard.java`**, **`GameSolver.java`**, **`SudokuSolver.java`** et **`SudokuApp.java`**, ainsi que **`Position.java`**, **`Tree.java`** et **`AbstractTree.java`**.
2. Compiler les fichiers en exécutant la commande **`make`** dans le terminal. Ceci compilera tous les fichiers Java et générera les fichiers **`.class`**.
3. Exécuter le programme en tapant la commande **`make run`**. Ceci affichera les résultats dans le terminal (`stdout`).

# Cadre pour `SudokuSolver`

```
public class SudokuSolver implements GameSolver {

    IntegerBoard board;
    IntegerBoard solution;
    Tree<IntegerBoard> ...;

    public SudokuSolver( GameBoard board )...
    }

    // mandatory interface methods
    public boolean solve()...
    public void printSolution()...

    // validate an insertion in the board
    public boolean isValidPlacement( int row, int col, Integer value )...

    // actual solver
    private boolean solveBoard()...
}
```

Votre joueur doit utiliser une structure d'arbre général de **`IntegerBoard`** pour solution les puzzles, **`Tree<IntegerBoard>`**.

# Système de notation

Votre travail sera noté selon les critères suivants :

- Code correct (résout des puzzles) 20%
- Conception orientée objet 20%
- Réussit tous tests 50%
- Propreté et lisibilité 10%
- BONUS 10% - si vous généralisez votre programme pour résoudre des problèmes de taille quelconque qu'on appelle variants du Sudoku, par exemple 4x4, 16x16, vous aurez 10 points bonis. Attention, vous devrez retourner une solution plutôt que le message de dimensions illégales. Notez qu'il pourrait aussi y avoir des puzzles de dimensions illégales, par exemple si les dimensions ne sont pas des carrés parfaits car le nombre de cases sur chaque côté doit être la racine carrée d'un nombre entier.

## *Détails du barème de correction*

Code correct : Le programme résout tous les cas fournis mais ne trouvent pas nécessairement les bonnes réponses.

Conception orientée objet : Le programme doit suivre les principes de programmation orientés objet. Par exemple, l'encapsulation et la camouflage d'information des classes, le respect des interfaces, un couplage minimal entre les classes, etc.

Réussit tous les tests : Réussit tous les tests du `SudokuApp` fourni et celui qui sera utilisé pour la correction.

Propreté et lisibilité : Le code doit être bien structuré, lisible et correctement commenté.

# Questions

Si vous avez des questions, veuillez les poster sur le forum TP1 de StudiUM ou communiquer directement avec les assistants à l'enseignement ou le professeur :

- Francois Major : francois.major@umontreal.ca
- Simon Guy simon.guy@umontreal.ca
- Mohamed Elyes Kanoun mohamed.elyes.kanoun@umontreal.ca
- Morteza Mahdiani: Morteza.mahdiani@umontreal.com

## ***Game on!***