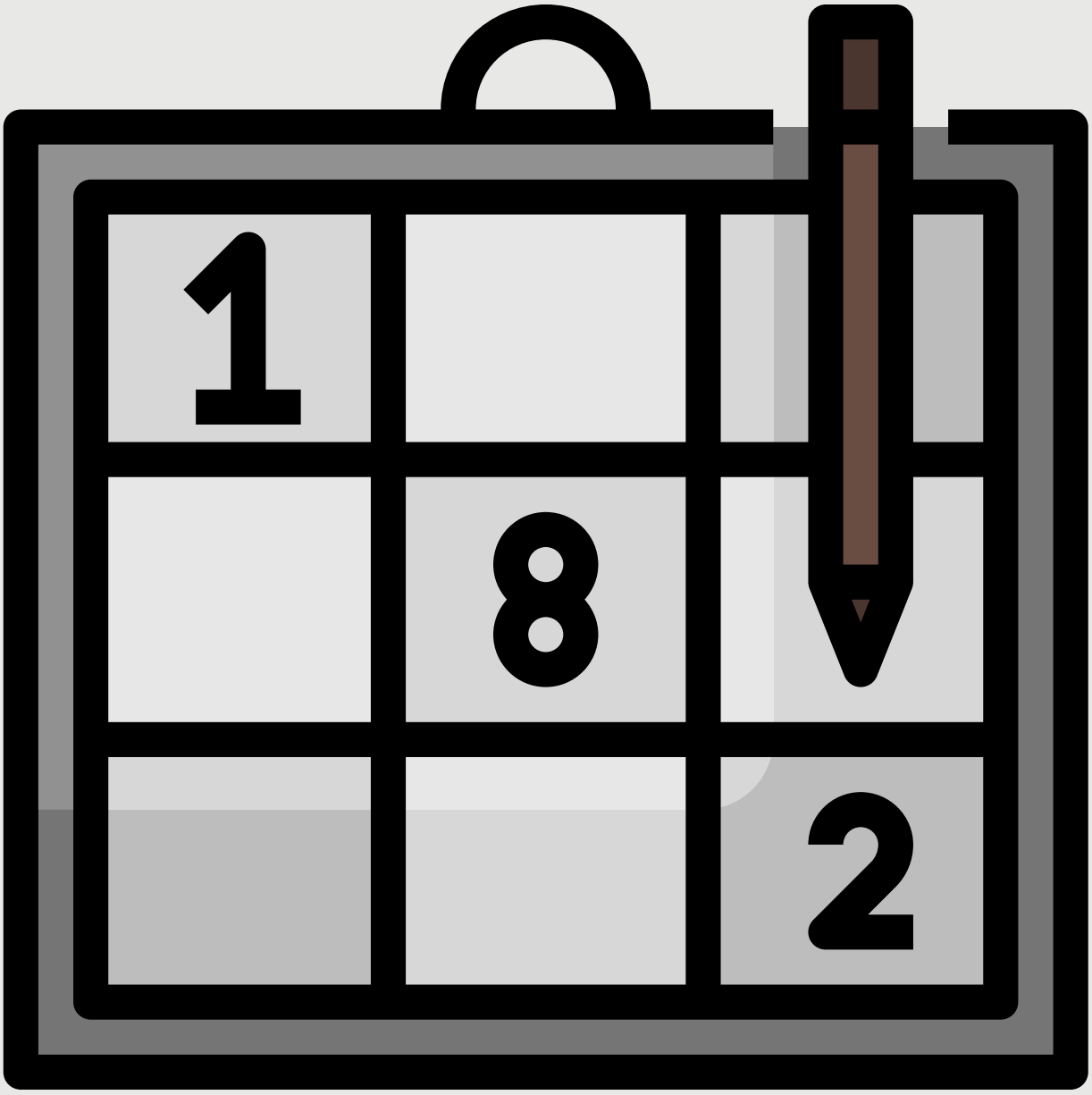
# Sudoku




Marwa Hafsati et Djedjiga Yahiaoui

# Sommaire

01

# Introduction

Le sudoku est un jeu omniprésent, que ce soit dans les journaux, les livres de jeux, ou même sur nos téléphones grâce à des applications dédiées ou des sites de jeu en ligne. L'objectif du jeu est de remplir une grille de 9x9 cases avec des chiffres de 1 à 9, sans qu'il y ait le même chiffre dans la même ligne, la même colonne ou la même région. Dans ce rapport, vous retrouverez comment nous avons programmé ce jeu en Java en deux partie une dédiée à un concepteur de grille et une dédiée à un joueur, à l'aide d'explications détaillées, de captures d'écran et d'un diagramme de classe. Il y a plusieurs classes que nous utilisons à la fois dans la partie concepteur et dans la partie joueur donc l'explication des programme débutera par les classes qui ne sont pas en communs

# 02 Premier programme et ses différentes classes (partie concepteur)

## Main.java

### Point d'entrée du concepreur

Nous avons développé une application en Java pour résoudre des grilles de Sudoku avec une interface graphique. Lorsque vous lancez l'application, une fenêtre intitulée "Joueur" s'ouvre, offrant trois options : "Résolution Automatique", "Résolution Manuelle" et "Charger Grille". Lorsque vous choisissez "Charger Grille", une boîte de dialogue s'affiche pour sélectionner un fichier contenant une grille de Sudoku. Une fois le fichier chargé, l'application lit son contenu pour afficher la grille. Ensuite, si vous sélectionnez "Résolution Automatique", l'application tente automatiquement de résoudre la grille chargée. Si la résolution est réussie, la grille résolue est affichée dans une nouvelle fenêtre. Sinon, un message d'erreur est affiché. Si vous préférez la "Résolution Manuelle", une interface s'ouvre vous permettant de saisir les chiffres pour résoudre la grille vous-même. Une fois la grille résolue, vous pouvez visualiser la solution dans une nouvelle fenêtre. Chaque case affiche le chiffre correspondant à la solution. Un bouton "Sauvegarder" est également disponible pour sauvegarder la grille résolue. En résumé, notre application offre une interface conviviale pour charger, résoudre et sauvegarder des grilles de Sudoku, vous permettant de profiter pleinement de ce jeu de réflexion classique.

```
public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Sudoku");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton grilleVideButton = new JButton("Partir d'une grille vide");
        JButton chargerGrilleButton = new JButton("Charger une grille");

        grilleVideButton.setBackground(new Color(50, 50, 200));
        grilleVideButton.setForeground(Color.WHITE);
        grilleVideButton.setFont(new Font("Arial", Font.BOLD, 14));
        grilleVideButton.setPreferredSize(new Dimension(180, 50));

        chargerGrilleButton.setBackground(new Color(50, 50, 200));
        chargerGrilleButton.setForeground(Color.WHITE);
        chargerGrilleButton.setFont(new Font("Arial", Font.BOLD, 14));
        chargerGrilleButton.setPreferredSize(new Dimension(180, 50));

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(2, 1, 10, 10));
        buttonPanel.setBackground(Color.WHITE);
        buttonPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

        buttonPanel.add(grilleVideButton);
        buttonPanel.add(chargerGrilleButton);

        frame.add(buttonPanel);

        frame.setSize(450, 600);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);

        grilleVideButton.addActionListener(new GrilleVideButtonListener(frame));
        chargerGrilleButton.addActionListener(new ChargerGrilleButtonListener(frame));
    }
    public static void afficherGrille(JFrame frame, GrilleVide grilleVide) {
        frame.getContentPane().removeAll();

        JPanel panel = new JPanel(new BorderLayout());
        panel.setBackground(Color.WHITE);

        JButton sauvegarderButton = new JButton("Sauvegarder");
        sauvegarderButton.addActionListener(new SauvegarderAction(frame, grillePanel));

        JPanel bottomButtonPanel = new JPanel();
        bottomButtonPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
        bottomButtonPanel.setBackground(Color.WHITE);
        bottomButtonPanel.add(sauvegarderButton);

        panel.add(grillePanel, BorderLayout.CENTER);
        panel.add(bottomButtonPanel, BorderLayout.SOUTH);

        frame.add(panel);

        frame.revalidate();
        frame.repaint();
    }
}
```

# 02 Premier programmme et ses différentes classes

**GrilleVideButtonListener.java**

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;

public class GrilleVideButtonListener implements ActionListener {
    private JFrame frame;

    public GrilleVideButtonListener(JFrame frame) {
        this.frame = frame;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        GrilleVide grillePanel = new GrilleVide(new int[9][9]);
        Main.afficherGrille(frame, grillePanel);
    }
}
```

Dans notre programme, nous avons créé une classe nommée `GrilleVideButtonListener` en Java pour gérer les événements associés au clic sur le bouton "Partir d'une grille vide" dans notre interface graphique Sudoku.

Nous avons importé les packages nécessaires pour travailler avec les événements d'action et les composants Swing, notamment `java.awt.event.ActionEvent`, `java.awt.event.ActionListener`, et `javax.swing.JFrame`.

Dans notre classe `GrilleVideButtonListener`, nous avons implémenté l'interface `ActionListener`, ce qui nous permet d'écouter les événements d'action, comme les clics de bouton. La classe contient un champ `frame` de type `JFrame`, qui représente la fenêtre principale de notre application.

Pour initialiser notre gestionnaire d'événements, nous avons écrit un constructeur qui prend un objet `JFrame` en paramètre et l'assigne au champ `frame`.

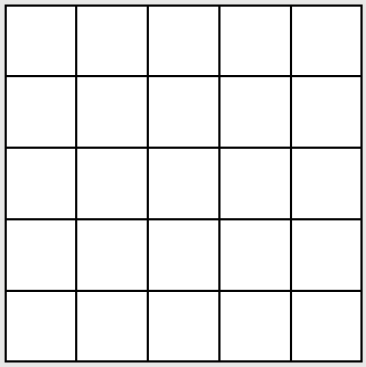
Lorsqu'un clic sur le bouton "Partir d'une grille vide" est détecté, la méthode `actionPerformed` est invoquée. À l'intérieur de cette méthode, nous créons une nouvelle instance de la classe `GrilleVide`, qui représente une grille de Sudoku vide de dimensions 9x9.

Ensuite, nous utilisons une méthode de la classe `Main` appelée `afficherGrille` pour afficher cette grille dans la fenêtre principale. Nous passons la `frame` actuelle et la nouvelle `grillePanel` en tant qu'arguments à cette méthode

# 02 Premier programmme et ses différentes classes

**GrilleVideAction.java**

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class GrilleVideAction implements ActionListener {
    private static final int GRID_SIZE = 9;
    private static final int CELL_SIZE = 50;
    private static final int GAP_SIZE = 5;

    private JFrame grilleFrame;

    @Override
    public void actionPerformed(ActionEvent e) {
        int[][] grilleVide = new int[GRID_SIZE][GRID_SIZE];
        GrilleVide grillePanel = new GrilleVide(grilleVide);

        grilleFrame = new JFrame("Grille Sudoku");
        grilleFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        grilleFrame.add(grillePanel); // Ajouter la grillePanel à la JFrame

        JButton sauvegarderButton = new JButton("Sauvegarder");
        sauvegarderButton.addActionListener(new SauvegarderActionListener(grilleVide, grilleFrame));

        JPanel buttonPanel = new JPanel(); // Créer un JPanel pour le bouton
        buttonPanel.add(sauvegarderButton); // Ajouter le bouton au JPanel

        grilleFrame.getContentPane().add(buttonPanel, BorderLayout.SOUTH); // Ajouter le JPanel au bas de la JFrame

        grilleFrame.pack();
        grilleFrame.setSize(CELL_SIZE * GRID_SIZE + GAP_SIZE * (GRID_SIZE + 1), CELL_SIZE * GRID_SIZE + GAP_SIZE * (GRID_SIZE + 1));
        grilleFrame.setVisible(true);
    }
```



Nous avons créé une classe Java appelée `GrilleVideAction` qui agit comme un gestionnaire d'événements lorsqu'un utilisateur souhaite créer une nouvelle grille de Sudoku vide.

Lorsqu'un événement est déclenché, la méthode `actionPerformed` est appelée. À l'intérieur de cette méthode, plusieurs étapes sont réalisées :

Nous avons initialisé une grille vide en créant un tableau 2D d'entiers de taille 9x9.

Nous avons créé une instance de la classe `GrilleVide` en lui passant la grille vide que nous venons de créer. Cette classe est responsable de l'affichage et de la gestion de la grille de Sudoku.

Une fenêtre (`JFrame`) a été créée pour afficher la grille de Sudoku. Le titre de la fenêtre a été défini sur "Grille Sudoku", et nous avons spécifié que la fermeture de la fenêtre ne devrait fermer que la fenêtre elle-même, pas toute l'application.

La grille de Sudoku a été ajoutée à la fenêtre en utilisant la méthode `add(grillePanel)`.

Un bouton "Sauvegarder" a été créé à l'aide de la classe `JButton`. Cela permettra à l'utilisateur de sauvegarder la grille. Nous avons ajouté un écouteur d'événements à ce bouton pour gérer les actions lorsque l'utilisateur clique dessus.

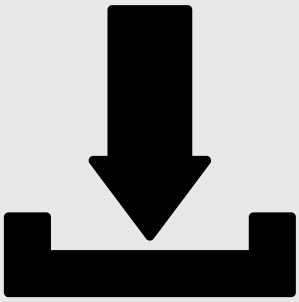
Un panneau (`JPanel`) a été créé pour contenir le bouton "Sauvegarder", et le bouton a été ajouté à ce panneau.

Ce panneau a été ajouté au bas de la fenêtre (`BorderLayout.SOUTH`) pour que le bouton soit affiché en dessous de la grille.

# 02 Premier programmme et ses différentes classes

**SauvegatderActionListener.java**

```java
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class SauvegarderActionListener implements ActionListener {
    private int[][] grille;
    private JFrame parentFrame;

    public SauvegarderActionListener(int[][] grille, JFrame parentFrame) {
        this.grille = grille;
        this.parentFrame = parentFrame;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        JFileChooser fileChooser = new JFileChooser();
        int result = fileChooser.showSaveDialog(parentFrame);
        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            sauvegarderGrille(selectedFile);
        }
    }

    private void sauvegarderGrille(File file) {
        try (PrintWriter writer = new PrintWriter(new FileWriter(file))) {
            for (int[] ligne : grille) {
                for (int chiffre : ligne) {
                    writer.print(chiffre);
                }
                writer.println();
            }
            JOptionPane.showMessageDialog(parentFrame, "Grille sauvegardée avec succès !", "Sauvegarde réussie", JOptionPane
.INFORMATION_MESSAGE);
        } catch (IOException ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(parentFrame, "Erreur lors de la sauvegarde de la grille", "Erreur", JOptionPan
E_MESSAGE);
        }
    }
```



Nous avons créé une classe Java nommée `SauvegarderActionListener` qui implémente l'interface `ActionListener`. Cette classe agit comme un gestionnaire d'événements lorsqu'un utilisateur souhaite sauvegarder la grille de Sudoku dans un fichier.

Dans le constructeur de la classe, nous avons passé la grille de Sudoku et le cadre parent (`JFrame`) en tant que paramètres. Cela nous permet d'accéder à la grille à sauvegarder et au cadre parent pour afficher les boîtes de dialogue associées.

Lorsque l'action est déclenchée, la méthode `actionPerformed` est appelée. À l'intérieur de cette méthode, nous avons créé une boîte de dialogue `JFileChooser` pour permettre à l'utilisateur de choisir l'emplacement et le nom du fichier dans lequel sauvegarder la grille.

Si l'utilisateur sélectionne un emplacement et un nom de fichier et clique sur "Enregistrer", la méthode `showSaveDialog` renvoie `JFileChooser.APPROVE_OPTION`. Dans ce cas, nous récupérons le fichier sélectionné à l'aide de `getSelectedFile()` et appelons la méthode `sauvegarderGrille` pour effectivement sauvegarder la grille dans ce fichier.

La méthode `sauvegarderGrille` prend le fichier sélectionné en paramètre et utilise un objet `PrintWriter` pour écrire les valeurs de la grille dans le fichier. En cas de succès, une boîte de dialogue d'information est affichée pour indiquer à l'utilisateur que la grille a été sauvegardée avec succès. En cas d'erreur lors de la sauvegarde, une boîte de dialogue d'erreur est affichée.

# 02 Premier programmme et ses différentes classes

## ChargerGrilleAction.java

Dans ce programme, nous avons créé une classe nommée `ChargerGrilleAction` en Java pour gérer les événements associés au chargement d'une grille depuis un fichier dans notre interface graphique Sudoku. Voici une explication détaillée du fonctionnement de ce programme :

Nous avons importé les packages nécessaires pour travailler avec les composants Swing, les événements d'action, les fichiers, et les exceptions, notamment `javax.swing.*`, `java.awt.event.ActionEvent`, `java.awt.event.ActionListener`, `java.io.File`, `java.io.IOException`, `java.nio.file.Files`, `java.util.ArrayList`, et `java.util.List`.

Dans notre classe `ChargerGrilleAction`, nous avons implémenté l'interface `ActionListener`, ce qui nous permet d'écouter les événements d'action, comme les clics de bouton.

Nous avons déclaré deux champs privés : `parentFrame`, qui est une référence à la fenêtre principale de notre application, et `gridPanel`, qui est une référence à la grille de Sudoku dans notre interface.

Nous avons écrit un constructeur qui prend en paramètres un objet `JFrame` et une instance de `GrilleVide` (la grille de Sudoku) et initialise les champs correspondants.

Nous avons écrit une méthode `chargerGrille` qui prend en paramètre un objet `File`, représentant le fichier à partir duquel charger la grille. À l'intérieur de cette méthode, nous lisons les lignes du fichier, convertissons les caractères en entiers pour obtenir les valeurs de la grille, et construisons un tableau 2D d'entiers représentant la grille.

Nous avons créé une nouvelle instance de `GrilleVide` avec les valeurs chargées à partir du fichier, puis nous avons retiré tous les composants actuels du conteneur principal (`parentFrame`), ajouté la nouvelle grille, et mis à jour l'affichage de la fenêtre.

Dans la méthode `actionPerformed`, qui est invoquée lorsque l'événement d'action se produit (c'est-à-dire lorsqu'un clic sur un bouton est détecté), nous avons créé un `JFileChooser` pour permettre à l'utilisateur de sélectionner un fichier à charger. Une fois le fichier sélectionné, nous avons appelé la méthode `chargerGrille` pour charger la grille à partir du fichier.

En cas d'erreur lors du chargement du fichier, une boîte de dialogue d'erreur est affichée pour informer l'utilisateur du problème.

## méthode actionperformerd

```
    public void actionPerformed(ActionEvent e) {
        JFileChooser fileChooser = new JFileChooser();
        int result = fileChooser.showOpenDialog(parentFrame);
        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            chargerGrille(selectedFile); // Appel de la méthode pour charger la grille
        }
    }
}
```

# 02 Premier programmme et ses différentes classes

**ChargerGrilleButtonListener.java**

Nous implémentons l'interface ActionListener dans cette classe, lui permettant ainsi d'écouter les événements d'action, notamment les clics de bouton.
La classe contient un champ frame de type JFrame, qui représente la fenêtre principale de l'application.
Le constructeur prend un objet JFrame en paramètre et l'assigne au champ frame.

Cette méthode est appelée lorsqu'un événement d'action se produit, en particulier lorsqu'un clic sur le bouton "Charger une grille" est détecté.
À l'intérieur de cette méthode, une instance de ChargerGrilleAction est créée en passant la frame actuelle et null (pour la grille, car elle n'est pas encore chargée).
Ensuite, la méthode actionPerformed de ChargerGrilleAction est appelée avec l'objet ActionEvent reçu en paramètre.

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;

public class ChargerGrilleButtonListener implements ActionListener {
    private JFrame frame;

    public ChargerGrilleButtonListener(JFrame frame) {
        this.frame = frame;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        ChargerGrilleAction chargerGrilleAction = new ChargerGrilleAction(frame, null);
        chargerGrilleAction.actionPerformed(e);
    }
}
```

# 03 Deuxièmes programmes et ses différentes classes(joueur)

**Menu.java**

Nous avons développé un programme Java qui propose une interface graphique pour résoudre des grilles de Sudoku. Notre application utilise la bibliothèque Swing pour créer une interface utilisateur conviviale. Lorsque nous lançons le programme, nous sommes accueillis par une fenêtre comportant trois boutons : "Résolution Automatique", "Résolution Manuelle" et "Charger Grille".
Lorsque nous cliquons sur "Charger Grille", une boîte de dialogue nous permet de sélectionner un fichier contenant une grille de Sudoku à résoudre. Une fois que nous avons choisi le fichier, le programme le lit et charge la grille dans l'application.
Ensuite, nous avons deux options pour résoudre la grille : automatiquement ou manuellement. Si nous optons pour la "Résolution Automatique" et que la grille peut être résolue, le programme affiche la grille résolue dans une nouvelle fenêtre. Sinon, un message d'erreur nous informe de l'impossibilité de résoudre la grille automatiquement.
Si nous préférons la "Résolution Manuelle", le programme ouvre une nouvelle interface où nous pouvons saisir les chiffres pour résoudre la grille nous-mêmes.
Une fois la grille résolue, nous pouvons visualiser la solution dans une nouvelle fenêtre, où chaque case affiche le chiffre correspondant à la solution. Un bouton "Sauvegarder" nous permet de sauvegarder la grille résolue.

# 03 Deuxièmes programmes et ses différentes classes

**SudokuSolver.java**

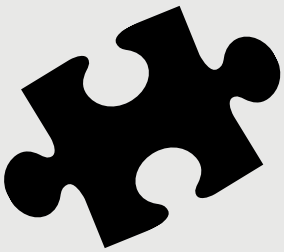
Nous avons ici une classe Java nommée SudokuSolver, conçue pour résoudre leq grille du Sudoku. Elle contient plusieurs méthodes pour effectuer cette tâche de manière récursive.

Latoute sécurité, la méthode revient en arrière pour essayer une autre valeur. Si la grille est entièrement remplie sans aucune contradiction, la méthode renvoie vrai, indiquant que le Sudoku a été résolu avec succès.

La méthode `estSecuritaire(int[][] grille, int ligne, int colonne, int num)` vérifie si le chiffre `num` peut être placé en toute sécurité dans la cellule spécifiée en vérifiant s'il n'est pas déjà présent dans la même ligne, colonne ou boîte 3x3.

Les méthodes `utiliseDansLigne()`, `utiliseDansColonne()` et `utiliseDansBoite()` sont des utilitaires utilisés par `estSecuritaire()` pour vérifier la présence d'un chiffre dans une ligne, une colonne ou une boîte 3x3 respectivement.

Ce programme utilise une approche de force brute récursive pour résoudre le Sudoku, en essayant toutes les combinaisons possibles jusqu'à ce qu'une solution soit trouvée.



# 03 Deuxièmes programmes et ses différentes classes(joueur)

**GrilleMain.java**

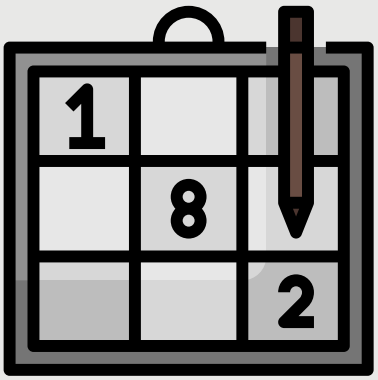
**Malgré que la classe s'appelle GrilleMain se n'est pas le main du programme**

Nous avons développé un programme Java pour résoudre des grilles de Sudoku de manière manuelle. Dans notre programme, la classe `GrilleMain` propose des méthodes pour choisir et afficher une grille de Sudoku, ainsi que pour résoudre manuellement la grille sélectionnée.

Lorsque nous utilisons la méthode `choisirGrille(int[][] grille)`, une nouvelle fenêtre s'ouvre avec le titre "Grille Sudoku", affichant la grille de Sudoku spécifiée. Cette grille est passée en paramètre à la méthode.

De même, lorsque nous optons pour la résolution manuelle avec la méthode `resolutionManuelle(int[][] grille)`, une nouvelle fenêtre apparaît, mais cette fois avec le titre "Résolution Manuelle du Sudoku". Encore une fois, la grille à résoudre est passée en paramètre.

Dans les deux cas, un bouton "Valider" est ajouté en haut de la fenêtre. Lorsque nous cliquons sur ce bouton, un ActionListener vérifie si la grille est complète. Si oui, une boîte de dialogue nous félicite pour avoir résolu la grille. Sinon, un message nous informe que la grille n'est pas encore complète.



# 03 Deuxièmes programmes et ses différentes classes(joueur)

## GrilleResolueInterface.java

Dans notre programme, la classe `GrilleResolueInterface` est utilisée pour afficher une grille de Sudoku résolue dans une nouvelle fenêtre graphique. Voici comment elle fonctionne :

Dans notre programme, la classe `GrilleResolueInterface` est utilisée pour afficher une grille de Sudoku résolue dans une nouvelle fenêtre graphique. Lorsque vous créez une instance de `GrilleResolueInterface` et que vous lui fournissez une grille résolue, elle crée une nouvelle fenêtre intitulée "Grille Résolue".

La disposition de la fenêtre est définie comme une grille 9x9 à l'aide de `GridLayout`, ce qui signifie que la grille sera affichée sous forme de grille 9x9 avec des cellules alignées.

Ensuite, pour chaque ligne de la grille résolue, et pour chaque valeur dans cette ligne, un objet `CaseGrille` est créé. La classe `CaseGrille` représente une cellule individuelle de la grille de Sudoku, affichant une valeur spécifique.

Ces objets `CaseGrille` sont ajoutés à la fenêtre `GrilleResolueInterface` à l'aide de la méthode `add()`, ce qui les place dans la disposition en grille.

Enfin, la fenêtre est ajustée à la taille de son contenu à l'aide de `pack()`, et elle est rendue visible avec `setVisible(true)`.

## CaseGrille.java

La classe `CaseGrille` est une sous-classe de `JTextField`, utilisée pour représenter une cellule individuelle dans une grille de Sudoku.
Lorsqu'une instance de `CaseGrille` est créée avec une valeur spécifique, cette valeur est utilisée pour initialiser le contenu de la cellule. Si la valeur est différente de zéro (ce qui signifie qu'il y a une valeur initiale dans la grille), le texte de la cellule est défini sur cette valeur à l'aide de `setText()`, et la cellule devient non éditable en utilisant `setEditable(false)`.

Cela signifie que lorsque la grille est affichée, les cellules contenant des valeurs initiales sont affichées avec leur valeur respective et sont verrouillées pour empêcher toute modification par l'utilisateur. Les cellules vides, représentées par des valeurs nulles, restent éditables pour permettre à l'utilisateur de remplir les cases vides lors de la résolution du Sudoku.

# 03 Deuxièmes programmes et ses différentes classes(joueur)

**MenuStarter.java**

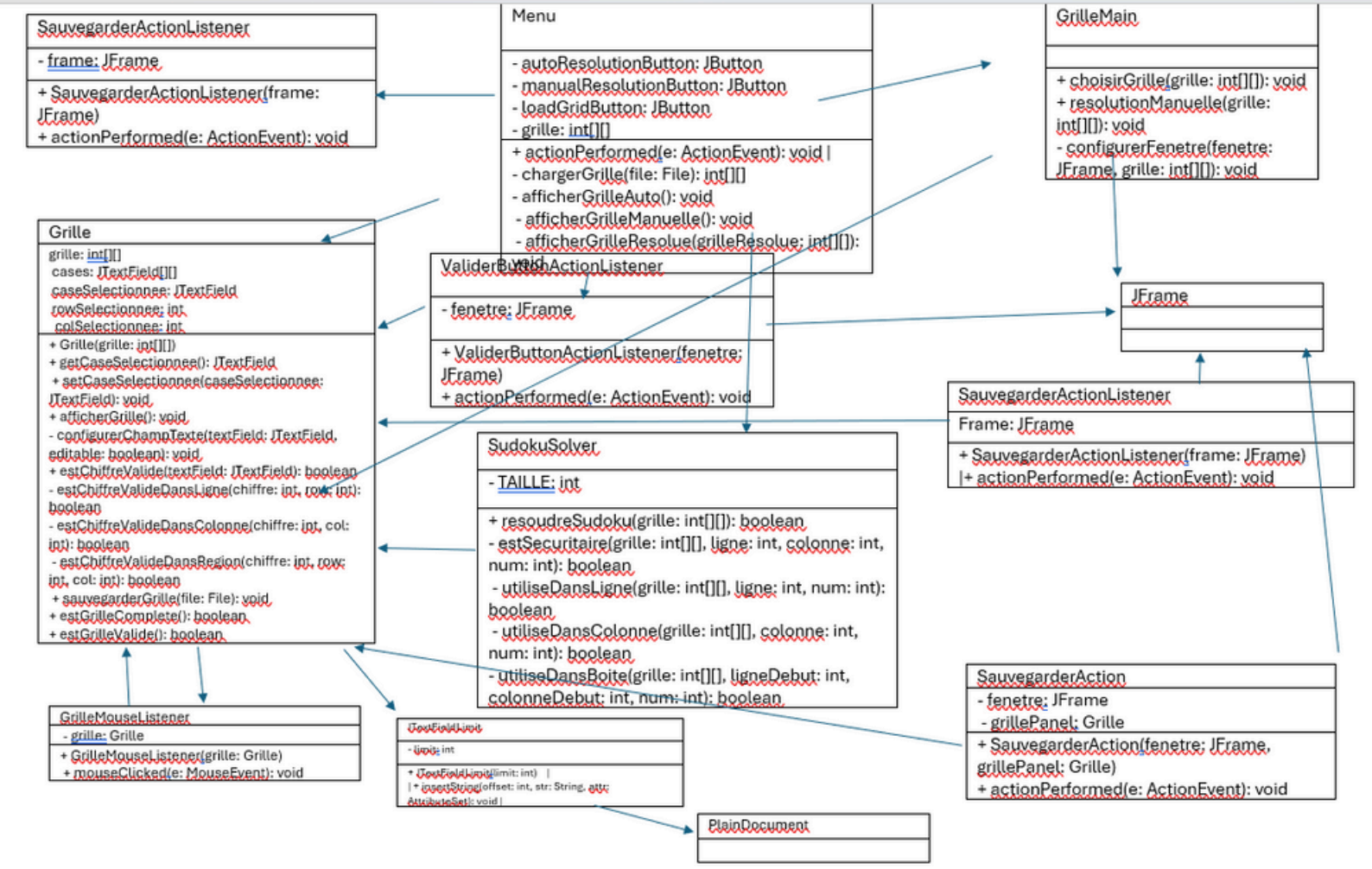
La classe `MenuStarter` implémente l'interface `Runnable`, ce qui nous permet de l'exécuter comme une tâche. La méthode `run()` crée une instance de la classe `Menu`, représentant l'interface utilisateur du menu, puis la rend visible.

En utilisant cette approche, lorsque nous exécutons `MenuStarter` en tant que tâche, elle démarre simplement l'interface utilisateur du menu. C'est une méthode standard pour démarrer une application graphique dans un environnement multithreadé.

```
public class MenuStarter implements Runnable {
    public void run() {
        Menu menu = new Menu();
        menu.setVisible(true);
    }
}
```

05

# Diagramme de classe de Joeur




Ce diagramme a été mit en tant que fichier pdf également.

06

# Conclusion

**Conclusion de Marwa**

En conclusion, ce projet m'a enseigné l'importance de la séparation maximale des classes dans des fichiers distincts. Cela facilite la manipulation du code et permet également de gagner du temps, car en séparant les classes, j'ai pu utiliser les classes de joueurs dans le programme du concepteur et inversement. Ce que j'ai particulièrement apprécié dans ce projet était la résolution manuelle du joueur, car cette partie du programme m'a semblé la plus simple par rapport aux autres parties du programme.

**Conclusion de Djedjiga**

En somme, ce projet m’a beaucoup appris sur la programmation en Java et bien plus que cela. En effet, l'exploration des algorithmes de résolution a renforcé ma compréhension de l'efficacité algorithmique, tandis que la conception de l'interface utilisateur graphique a mis en lumière l'importance de l'expérience utilisateur dans le développement logiciel. Ces aspects combinés ont enrichi mon expérience en développement Java, tout en m'apportant de précieuses leçons sur la conception logicielle et l'optimisation des performances