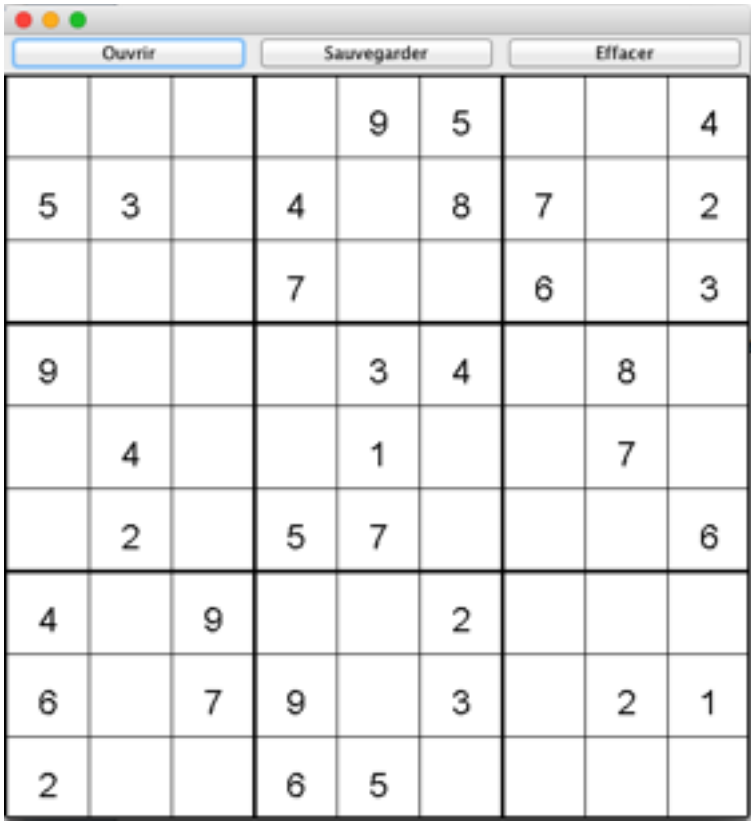
Austin Luke
Groupe 6
IS2
APL

Le Vendredi 13 mai, 2016

Compte Rendu du Projet Tutoré



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

Table de Matières

I. Introduction

Le projet consistait à créer en java deux interfaces avec lesquels un utilisateur peut éditer et/ou résoudre une grille du jeu Sudoku.

II. Fonctionnalités

La classe GridObject rassemble tout les valeurs utiles pour le projet:

```
public GridObject()
{
    this.tab = new int[9][9];
    this.textarray = new JTextField[9][9];
    this.valid = new boolean[9][9];
    this.clickable = new boolean[9][9];

    ButtonListener.emptyGrid(this);
}
```

- les valeurs de la grille,
- les champs de texte de la grille,
- si une valeur est valide, et
- si elle est clickable.

L'unicité est contrôlé a chaque changement de valeur. Ainsi une valeur est toujours dans l'état correcte. L'aspect visuelle est déterminé par sa validité et si elle est clickable. Ainsi l'utilisateur a un moyen simple et convivial de comprendre l'état d'une valeur. Par exemple une valeur est rouge si elle n'est pas valide.

Les deux classes PaintGrid et Fenetre déterminent l'aspect visuel des interfaces. "Fenetre" initialise tous les conteneurs (JComponents) des interfaces alors que PaintGrid trace la grille et les chiffres qui s'y trouvent.

Un utilisateur peut interagir avec le programme soit

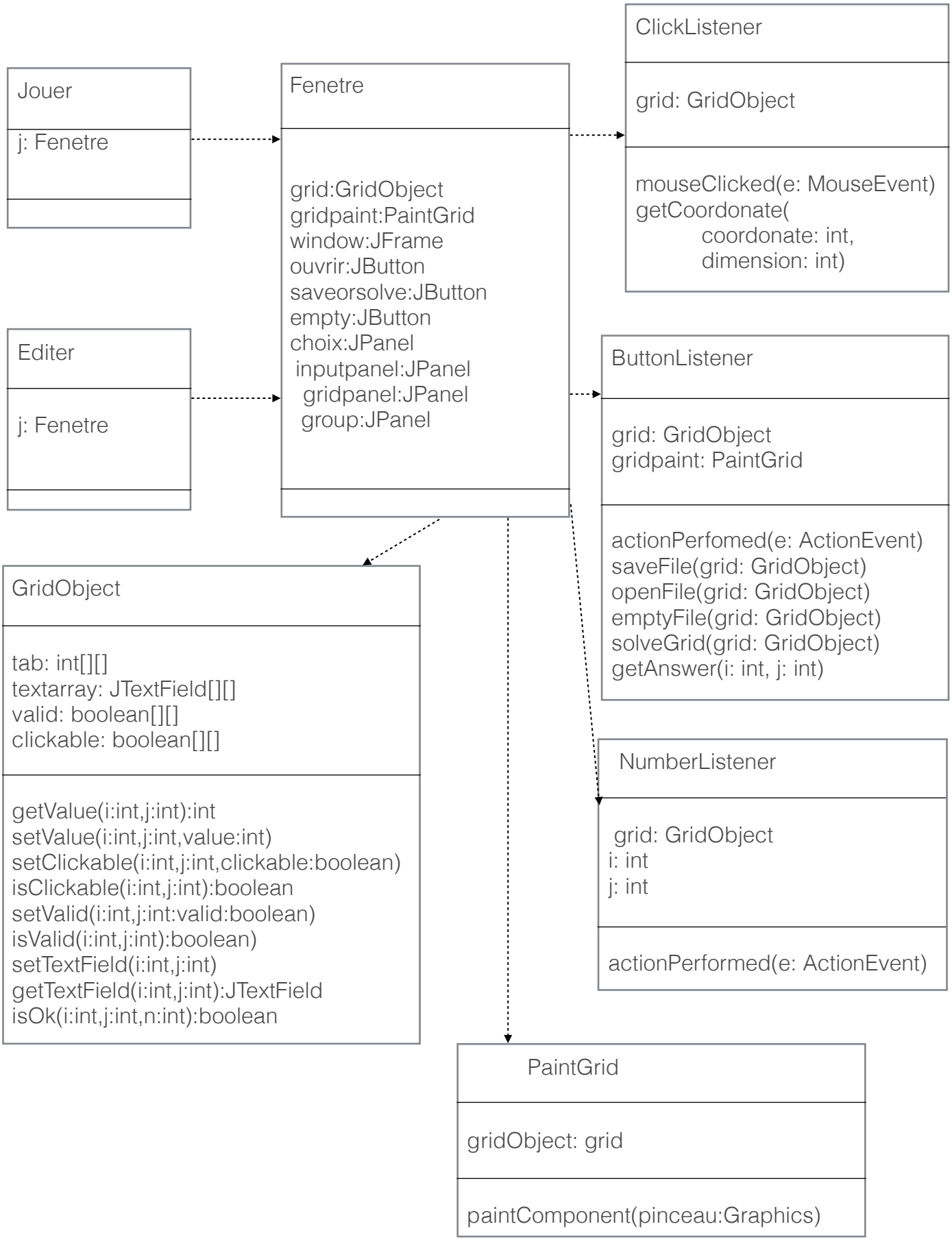
- en cliquant sur une case (ClickListener),
- en cliquant sur un bouton (ButtonListener),
- en appuyant sur la touche entrer (NumberListener).

ClickListener et NumberListener sont complémentaires et servent à entrer des valeurs manuellement. ButtonListener execute toutes les fonctionnalités de remplissage automatique comme par exemple effacer une grille ou la résoudre. Le code ci-dessous est extrait de la class ButtonListener.

```
public void actionPerformed(ActionEvent e)
{
    if(e.getActionCommand() == "Sauvegarder")
        saveFile(grid);
    else if(e.getActionCommand() == "Ouvrir")
    {
        emptyGrid(grid);
        openFile(grid);
    }
    else if(e.getActionCommand() == "Effacer")
        emptyGrid(grid);
    else if(e.getActionCommand() == "Resoudre")
        solveGrid(grid);

    gridpaint.repaint();
}
```

III. Structure



IV. Algorithme de Résolution

L'algorithme que j'ai trouvé est en fin de compte très simple. En tout elle ne prend qu'environ 50 lignes de code et deux fonctions. Elle se base sur l'hypothèse qu'il y a toujours au moins une case dans la grille qui n'a qu'une seule solution et que trouver ce nombre conduira a trouver le prochain et ainsi de suite jusqu'à ce que la grille soit entièrement résolue.

En essayant d'appliquer ce démarche ligne par ligne, j'ai découvert que vérifier zone par zone était plus efficace, c'est-à dire qu'il y à moins de solutions pour un nombre dans une zone (3x3) en général qu'en vérifiant les possibilités pour chaque ligne et donc c'est plus probable de trouver une solution unique.

```
for(int n=1;n<=9;n++)
{
    for(int i=0;i<9;i+=3)
    {
        for(int j=0;j<9;j+=3)
        {
            possible=0;
            for(int ini=i+2;ini>=i;ini--)
            {
                for(int inj=j+2;inj>=j;inj--)
                {
                    if(!(grid.isValid(ini,inj)))
                    {
                        grid.setValue(ini,inj,0);
                        grid.setValue(ini,inj,n);
                        if(grid.isValid(ini,inj))
                        {
                            x=ini;
                            y=inj;
                            possible++;
                        }
                        grid.setValue(ini,inj,0);
                    }
                }
            }
            if(possible==1) // Si il n'y a qu'une seul possibilité c'est surement la solution.
                grid.setValue(x,y,n);
        }
    }
}
```

V. Conclusion

En comparant ce projet avec ceux écrits en C, j'avais l'impression d'être guidé par les conventions du Java qui encouragent une certaine organisation du code dans des classes, alors que le C est beaucoup moins contraignant sur sa structure globale.

J'avais donc l'impression d'écrire du code plus organisé et aussi plus facile à utiliser, c'est à dire que l'organisation rend les parties du code plus isolés et donc je n'avais pas à réécrire tout le code a chaque ajout, suppression, ou modification.