

# RETOUR D'EXPERIENCE SUR LE PROJET « PETIT TABLEUR »

CHAUVEAU Arthur

NGWALA-NGWALA Arnaud

THURET Côme

# Table des matières

Table des matières .....	2
Présentation du projet.....	3
Introduction générale du sujet : .....	3
Fonctionnalités du programme .....	4
Structure du programme .....	6
Arbre Binaire .....	7
Algorithme de références circulaires .....	8
Énumération des structures de données.....	9
Conclusions personnelles.....	10
CHAUVEAU Arthur : .....	10
NGWALA-NGWALA Arnaud :.....	10
THURET Côme : .....	10

## Présentation du projet

### Introduction générale du sujet :

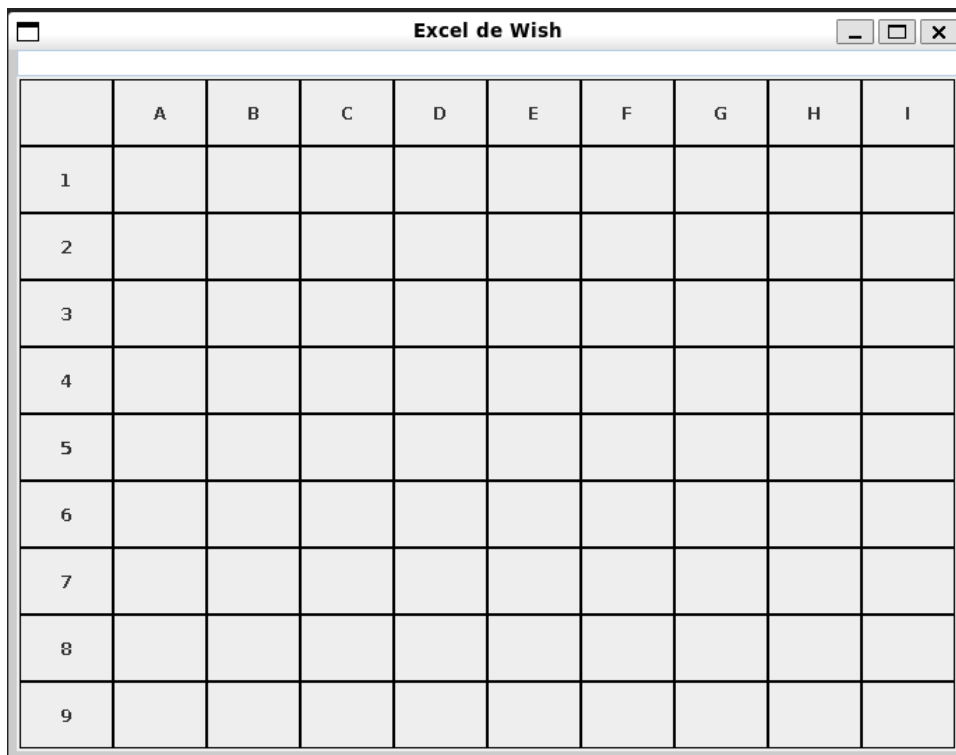
L'objectif du projet était en Java, de coder un tableur – comme Microsoft Excel, LibreOffice Calc et consorts – mais en introduisant pour les calculs un algorithme permettant de résoudre les opérations selon une notation préfixe et en simplifiant, nous cantonnant aux opérations de base que sont l'addition, la soustraction, la division et la multiplication et en imposant une grille de 9\*9 cellules.

Pour vérifier que les calculs étaient possibles il fallait aussi constamment vérifier l'état de la cellule pour signaler à l'utilisateur s'il y avait un quelconque problème de saisie empêchant le calcul.

## Fonctionnalités du programme

Notre programme de petit tableur propose une grille de 9 par 9 cellules.

Les utilisateurs peuvent modifier les valeurs en écrivant des formules dans la barre dédiée, et les résultats sont automatiquement affichés.



Chaque cellule peut être dans l'une des situations suivantes :

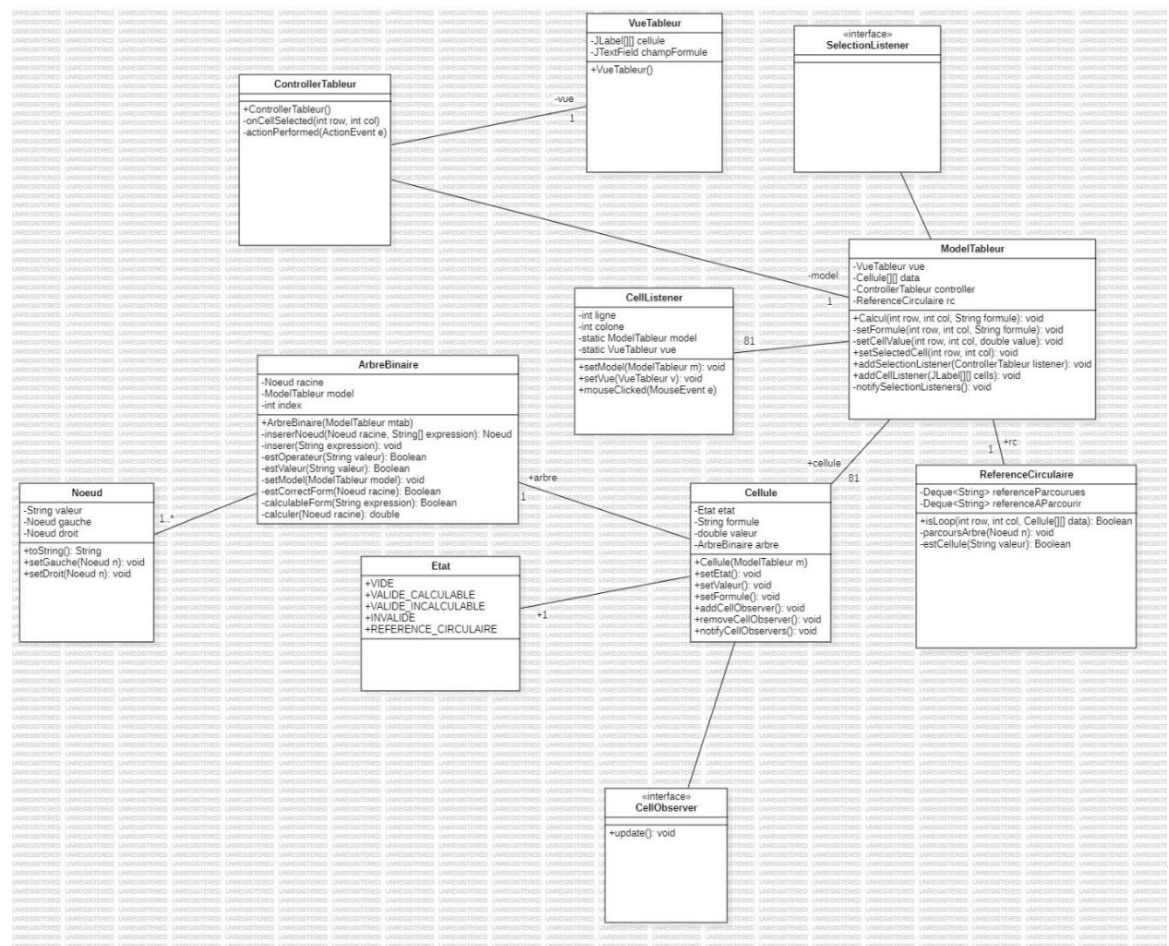
- **Vide** : En attente d'une formule. (Cellule en fond blanc)
- **Calculable** : Affiche le résultat si la formule est correcte. (Cellule avec une valeur dedans)
- **Incalculable** : Indique si une formule correcte ne peut pas être évaluée. (Cellule en fond orange)
- **Incorrecte** : Signale une erreur de syntaxe ou une formule mal écrite. (Cellule en fond rouge)
- Les utilisateurs peuvent voir la cellule sélectionnée (Cellule en fond jaune) et mettre à jour les valeurs qu'il y a dans les cellules.

Excel de Wish									
	A	B	C	D	E	F	G	H	I
1									
2				1.0					
3			45.0						
4									
5									
6									
7									
8									
9									

# Structure du programme

Le programme est composé d'un total de 12 classes :

- Le main, ne sert qu'à lancer le programme en instanciant un objet de la classe ControllerTableur.
- ControllerTableur, c'est donc la classe qui représente le controller du programme, il crée les objets de type VueTableur et ModelTableur, permettant d'initialiser la vue et le model du programme. La classe gère aussi les actions lors de la sélection d'une cellule.
- VueTableur, c'est la classe qui permet de créer la vue du tableau.
- ModelTableur, ModelTableur permet la création du modèle et donc du tableau de cellule correspondant à la vue ainsi que des méthodes pour interagir avec des objets de la classe Cellule.
- SelectionListener,
- ReferenceCirculaire,
- CellListener,
- Cellule, comme son nom l'indique cette classe permet la représentation d'une cellule et stocke des données tel que leur valeur, leur état, leur formule, le modèle, la liste de ses observer ainsi que l'arbre correspondant à leur formule.
- Etat, est un type énuméré représentant les différents états d'une cellule.
- ArbreBinaire, est la classe qui permet de représenter un arbre et qui possède des méthodes qui ont effet directement sur lui ou sur les valeurs qu'il contient.
- Noeud, cette classe qui permet de créer un Noeud ayant une valeur, un fils gauche et un fils droit.



## Arbre Binaire

L'arbre de calcul est composé de deux classes. La première est la classe `ArbreBinaire` qui sert à l'insertion des nœuds et aux différentes méthodes liées à l'arbre comme le calcul, les méthodes de vérification de formule, etc. La seconde classe utilisée dans la représentation de l'arbre de calcul est la classe `Noeud`, c'est cette classe qui permet de stocker les données de l'arbre et de parcourir l'arborescence.

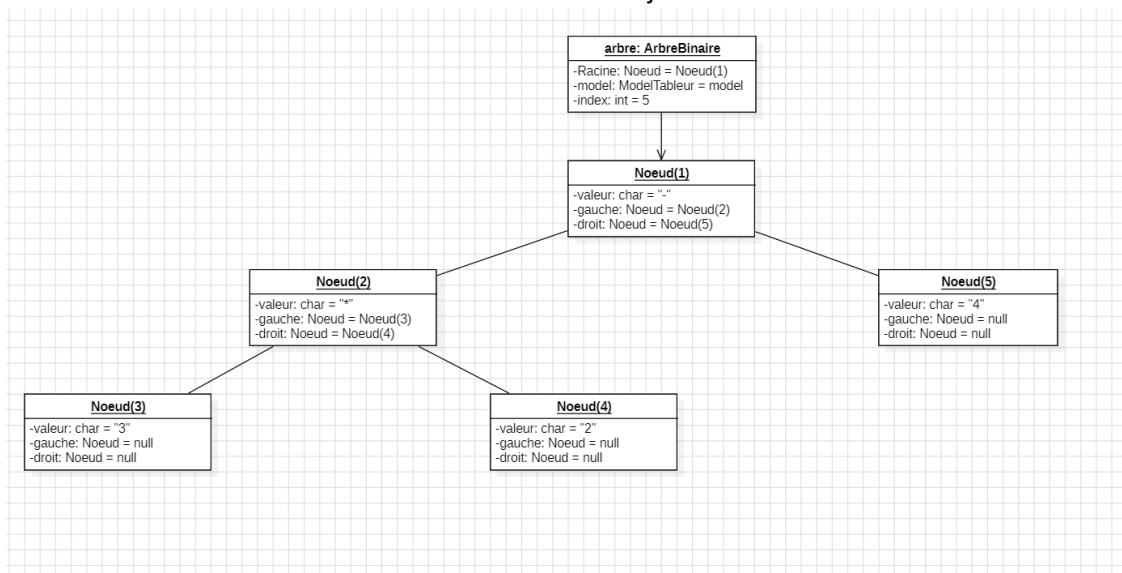
A titre de représentation, imaginons que nous créons l'arbre représentant le calcul suivant :

$(3 * 2) - 4$ , retranscrit en la formule en notation préfixe :  $- * 3 2 4$

L'arbre va être créé de la façon suivante :

- Création de l'objet arbre en initialisant l'attribut racine à null
- L'appel de la méthode `InsererNoeud` va permettre de remplir l'arbre selon la formule précédemment remplie de façon récursive. Le premier appel va créer la racine, ici `"-"` et l'index à 1.
- Le second appel de la fonction va donc créer le fils gauche de la racine qui sera égale à `"*"` et va instancier l'index à 2.
- Le troisième appel va lui instancier le fils gauche du nœud précédant et lui assigner la valeur de `"3"` et définir l'index à 3.
- Le quatrième appel va créer le fils droit du deuxième nœud et lui assigner la valeur de `"2"` et définir l'index à 4.
- Le cinquième appel va donc créer le fils droit de la racine en lui donnant la valeur de `"4"` et ajoute 1 à la valeur de l'index.
- L'index est maintenant supérieur au nombre d'opérande et d'opérateur. Nous sortons de la boucle et la méthode renvoie la racine

Nous obtenons donc les objets suivants :



## Algorithme de références circulaires

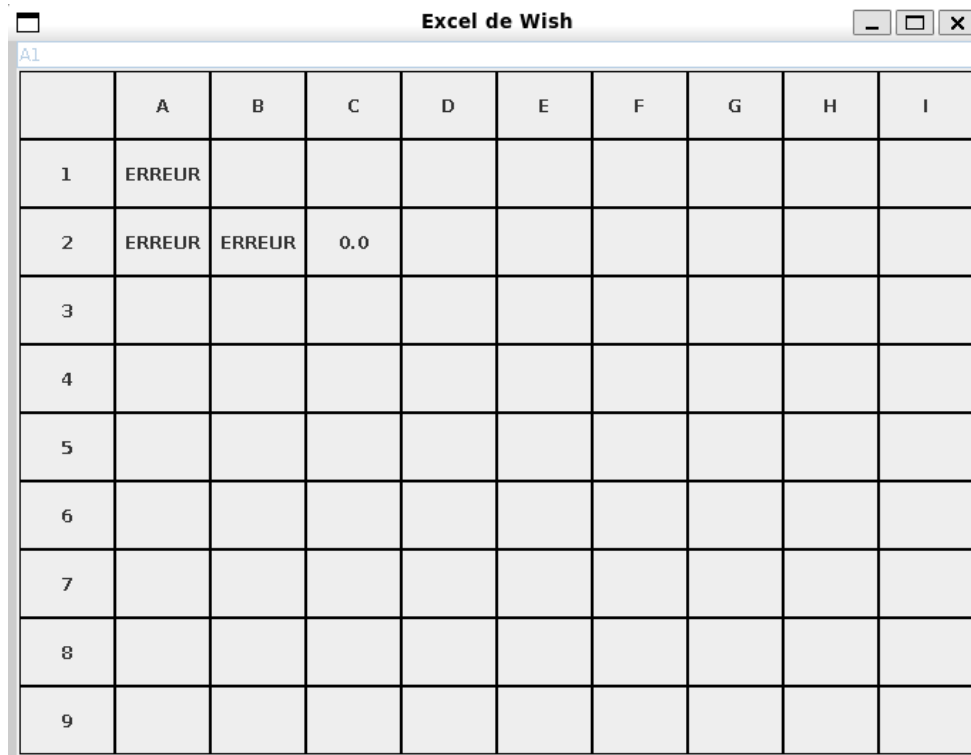
Afin de détecter les références circulaires, [une classe a été écrite](#) et est dédiée à toutes les opérations qui tournent autour de cette vérification. Mais deux méthodes vont vraiment nous intéresser pour les chercher : *isLoop* et *parcoursArbre*.

La première est la classe importante : Pour pouvoir déceler si oui ou non nous avons des références circulaires et s'il reste des cellules à visiter, deux piles sont instanciées : *référencesàParcourir*, *référencesParcourues*.

Pour donner suite à cela, on récupère la racine et on l'ajoute dans la première pile. On enchaîne sur une boucle qui tourne tant que la pile n'est pas vide, qui va appeler la deuxième méthode vue plus haut pour parcourir l'arbre étant associé à la cellule sur laquelle on est actuellement selon un parcours en profondeur et qui s'il trouve une cellule, l'ajoute à la deuxième pile.

Lorsque la méthode de parcours se termine, on vérifie si la cellule actuelle n'est pas déjà dans la pile des références parcourues, le cas échéant le booléen *true* est renvoyé et les cellules concernées ont un message d'erreur autrement on recommence jusqu'à ce qu'il n'y ait plus rien à parcourir.

En termes d'efficacité, le programme doit malheureusement parcourir à chaque fois un *ArrayDeque* ce qui ne doit pas être des plus économes en termes de ressources.



	A	B	C	D	E	F	G	H	I
1	ERREUR								
2	ERREUR	ERREUR	0.0						
3									
4									
5									
6									
7									
8									
9									



## Énumération des structures de données

Dans notre projet, nous avons utilisé trois structures de données abstraites principales :

- Les piles
- Les arbres
- Les listes

Les piles ont été utilisées dans l'algorithme des références circulaires.

Les arbres ont été nécessaires afin de faire les calculs.

Et les listes ont été utilisées dans l'algorithme de mise à jour.

## Conclusions personnelles

### CHAUVEAU Arthur :

Pour ma part, ce projet m'a permis de réaliser qu'il est essentiel de débiter un projet dès qu'on nous le confie, car celui-ci sera souvent très chronophage, comme ce fut le cas pour le petit tableur que nous avons dû réaliser. Même en ayant entamé le projet rapidement, nous avons constaté qu'il était plus exigeant en termes de temps que ce que nous avions initialement envisagé.

Ce projet m'a également permis d'améliorer mes compétences en java. En particulier, en élaborant la javadoc pour certains fichiers, j'ai compris toute son utilité en visualisant l'ensemble des méthodes présentes dans un seul fichier. Elle facilite la compréhension des fonctionnalités des méthodes pour toute personne consultant le code ultérieurement.

Enfin, cela m'a fait prendre conscience qu'un projet est souvent plus gérable en équipe ou en groupe, avec une bonne organisation. Bien que le démarrage puisse prendre du temps en raison des nécessaires concertations sur la répartition des tâches, cette phase s'avère indispensable.

### NGWALA-NGWALA Arnaud :

Ce projet était pour moi très enrichissant car il y avait une sorte de défi, certaines classes ont été plus difficiles à modéliser que d'autres mais quand tout fonctionne l'on a cette sensation de travail accompli et de satisfaction.

Il m'a également permis de renforcer mes acquis du langage et d'aborder d'une autre manière les travaux de programmation en Java. Pas que je n'aime pas le langage mais plutôt qu'il me frustre par moments et au cours de cette SAé je n'irai pas jusqu'à dire que c'était fluide mais c'était un plaisir d'écrire toutes ces classes, de réfléchir aux méthodes et à leur bonne implémentation.

### THURET Côme :

Ce projet a été encore pour moi une démonstration des marges de progression que j'ai en termes d'organisation. En effet le temps et les imprévus ont été ce qui a mis le plus en péril le projet selon moi, tout d'abords le temps car le projet s'est terminé proche de l'heure limite de rendu, rendant le travail difficile, mais les imprévus ont aussi joué dans les problèmes liés à l'organisation tel que le non-fonctionnement de GIT à certains moments. Pour conclure avec l'organisation j'ai appris que je dois me donner des horaires fixe de travail durant toute la durée du projet et je dois prévoir des imprévus tel que la dysfonction de GIT.

Pour ce qui est du côté technique, je n'ai pas eu de problème majeur durant la SAé. Je deviens de plus en plus à l'aise avec le code que je produis et suis plus à même de le modifier quand nécessaire, la théorie derrière le code fut aussi assez fluide dans l'acquisition.

Pour finir ce fut encore une fois pour moi une leçon sur mon organisation. Cependant j'ai pu prendre plus confiance en mes capacité technique et théoriques.