

SAÉ 3.2

RAPPORT

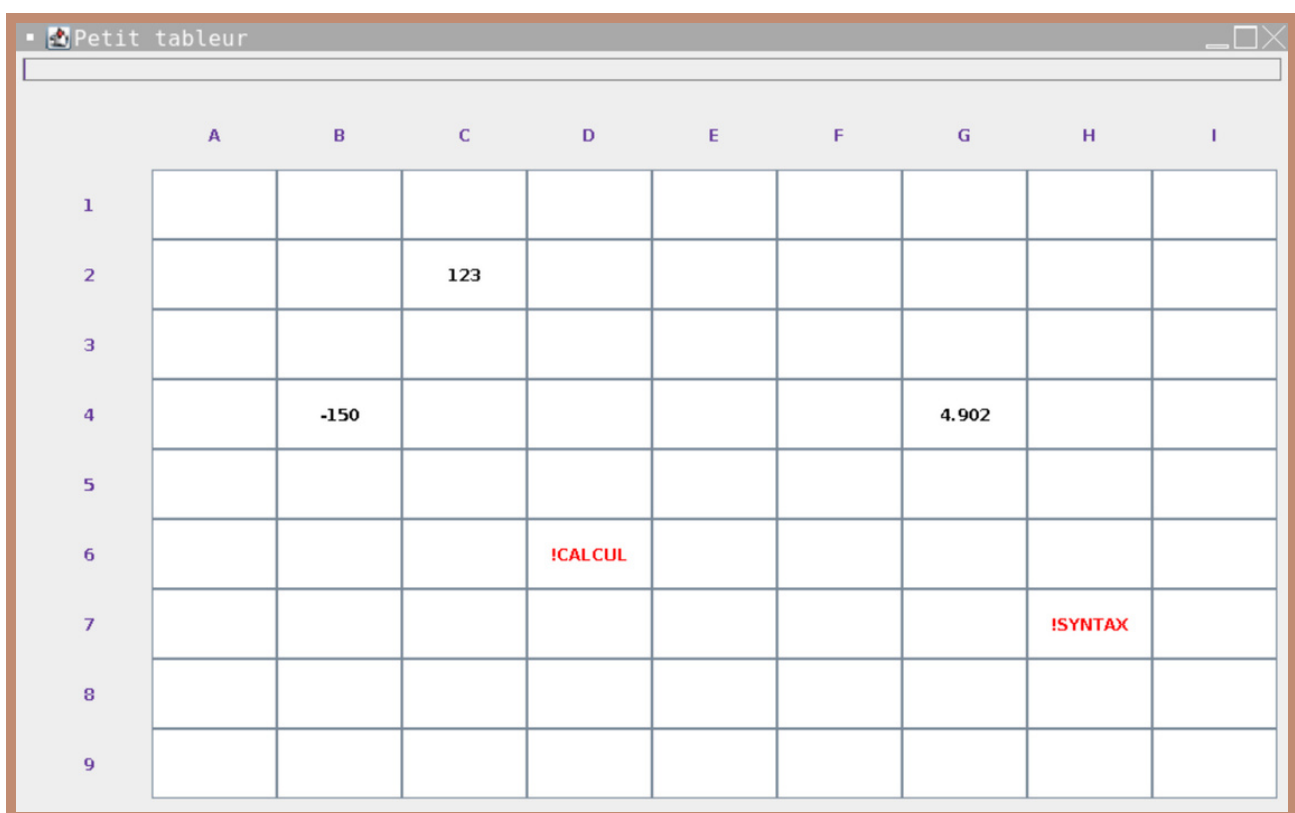
Nolan Toussaint
Lucile Pereira
Firmin Ndacleu

SOMMAIRE

Introduction	3
Fonctionnalités	4
Structure	5
AST	7
Références circulaires	9
Structures de données	10
Conclusion	11

INTRODUCTION

Dans le cadre de cette SAÉ, nous avons dû réaliser, en langage Java, un tableur qui permet de stocker et de calculer des formules en notation préfixe. Elles doivent être stockées dans des cellules, et doivent donc pouvoir faire référence au contenu d'autres cellules. Il doit aussi y avoir une gestion des erreurs de l'utilisateur, notamment la création de références circulaires.



The screenshot shows a window titled "Petit tableur" containing a spreadsheet with 9 columns (A-I) and 9 rows (1-9). The data is as follows:

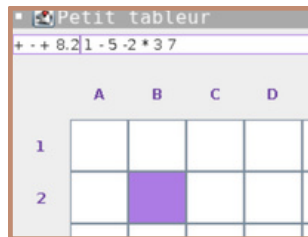
	A	B	C	D	E	F	G	H	I
1									
2			123						
3									
4		-150					4.902		
5									
6				!CALCUL					
7								!SYNTAX	
8									
9									

Un exemple d'utilisation de notre tableur

FONCTIONNALITÉS

On peut d'entrer une formule différente dans chacune des 81 cases. en cliquant sur la case et en entrant la formule dans un champ dédié.

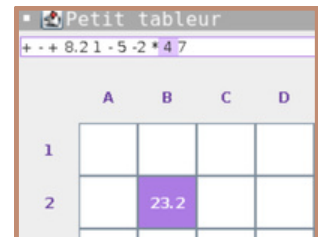
La formule en notation préfixe est enregistrée, et le résultat est affiché dans la case. On peut ensuite récupérer et modifier la formule en cliquant à nouveau sur la case.



On entre une formule



Le résultat s'affiche



On peut modifier la formule

On peut faire référence à d'autres cellules dans une formule, en inscrivant l'identifiant de la cellule. Lors du calcul, cette référence est remplacée par le résultat contenu dans la cellule.

Si le résultat d'une cellule change, le résultat de toutes les cellules qui y font référence est automatiquement mis à jour.



On fait référence à une cellule



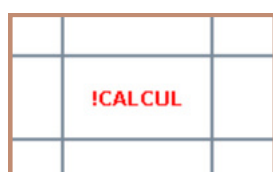
Le calcul est effectué



D2 est mise à jour après modification de B2

Si l'utilisateur entre une formule incorrecte (erreur de syntaxe ou introduction d'une référence circulaire) dans une cellule, elle affichera " !SYNTAX " en rouge au lieu d'afficher un résultat.

Si l'utilisateur entre une formule incalculable (division par 0 ou référence à une cellule invalide) dans une cellule, elle affichera " !CALCUL " en rouge au lieu d'afficher un résultat.



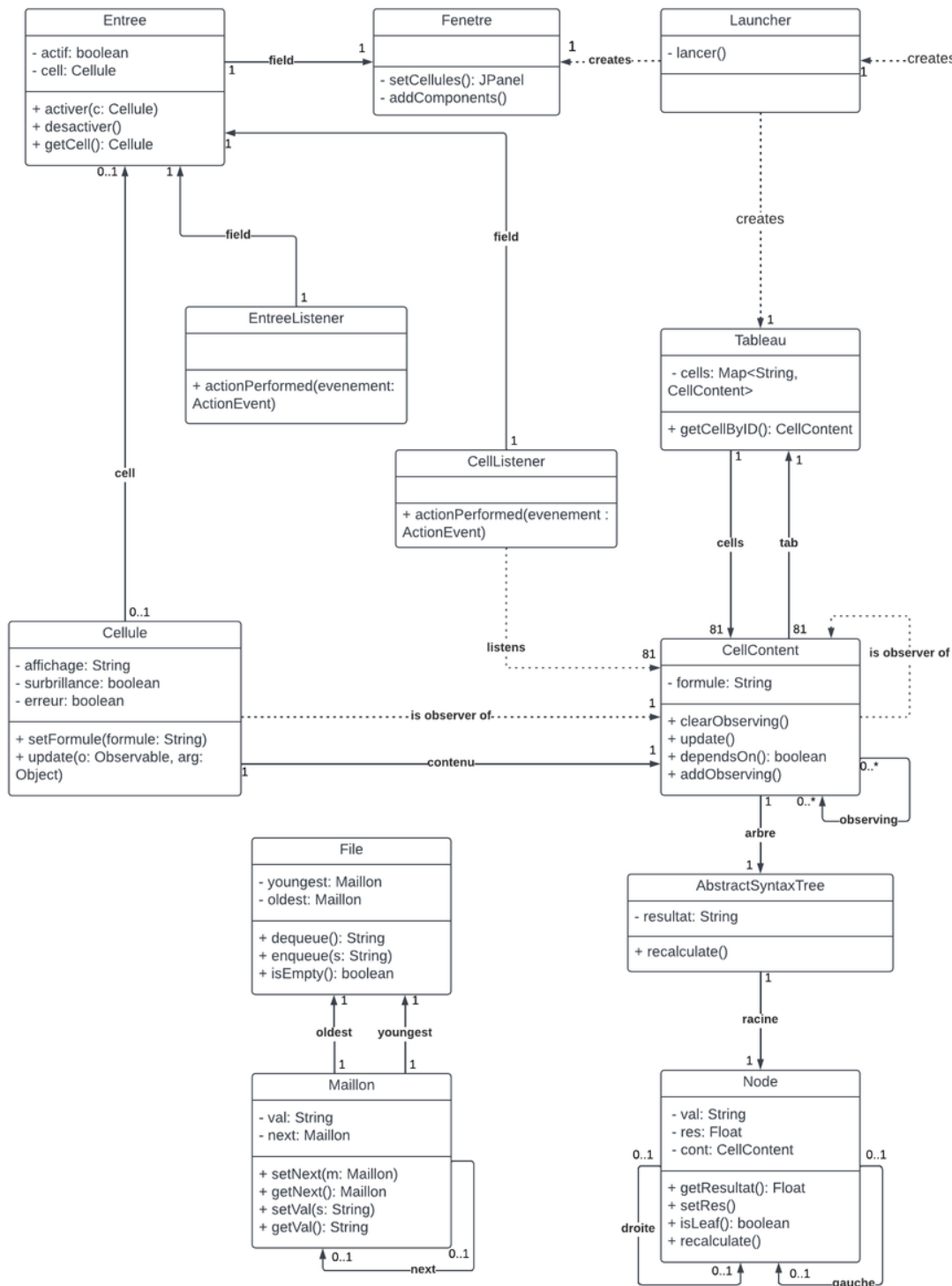
Une formule incalculable



Une formule incorrecte

STRUCTURE

DIAGRAMME DE CLASSE



EXPLICATION

Le programme suit le modèle MVC.

La classe **Main** lance le **Launcher** qui va créer un **Tableau** et une **Fenêtre**.

Côté visuel, la **Fenêtre** contient une **Entrée** et une grille de **Cellules**. L'**Entree** est écoutée par un **EntreeListener**, et chaque **Cellule** est écoutée par un même **CellListener**.

Côté données, le **Tableau** contient 81 **CellContents**, chacun représentant le contenu d'une **Cellule**, qui est un **Observer** du **CellContent** la représentant. Un **CellContent** contenant une formule contiendra un **AbstractSyntaxTree**, qui sera composé de plus ou moins de **Nodes**. Pour créer un **AbstractSyntaxTree**, on utilise une **File**, elle même composée de **Maillons**.

Chaque **CellContent** qui fait référence à un ou plusieurs autres **CellContents** dans sa formule est un **Observer** de ces **CellContents**.

Côté contrôleurs, le **CellListener** permet de détecter les clics sur une **Cellule** et d'activer l'**Entree**. L'**EntreeListener** permet de récupérer une formule entrée par l'utilisateur et d'en faire la formule d'un **CellContent**

ARBRE DE SYNTAXE ABSTRAITE

ABSTRACTSYNTAXTREE

La classe `AbstractSyntaxTree` lance la création récursive des nœuds de l'arbre. Elle récupère ensuite le résultat du calcul, ou, si la formule est incorrecte, elle catch l'exception produite lors de la détection de l'erreur et récupère le message associé. C'est ce résultat ou ce message qui est enregistré comme résultat de la formule.

NODE

La classe `Node` représente chaque nœud de l'arbre, c'est-à-dire chaque symbole ou terme de la formule. La création d'un nœud se fait à partir d'une `File`. Lorsque le nœud est créé, il récupère le plus ancien élément de la file. C'est cet élément que représente le nœud, c'est sa valeur.

Si l'élément est un symbole (+, -, *, /), alors on sait que le nœud doit avoir deux fils, et on les crée si c'est possible, sinon on renvoie une erreur de syntaxe. Sinon, le nœud est une feuille et on calcule son résultat.

Lors du calcul du résultat d'une feuille, on tente de convertir sa valeur en nombre. Si ça marche, ce nombre devient le résultat du nœud. Sinon, on essaye de récupérer une cellule du tableau grâce à la valeur de la feuille. Si ça marche, le résultat de la cellule appelée devient le résultat du nœud, sauf si il y a une référence circulaire. Si aucun de ces essais ne fonctionnent, on lève une exception qui s'affichera dans la cellule.

FILE

La classe `File` représente la formule qu'on lui donne, décomposée en séparant chaque symbole ou terme lors de sa création en différents maillons. Elle change également une virgule en point si c'est un décimal.

On peut ajouter un élément au début de la file ou en retirer un à la fin et le renvoyer de manière LIFO. On peut également savoir si elle est vide.

MAILLON

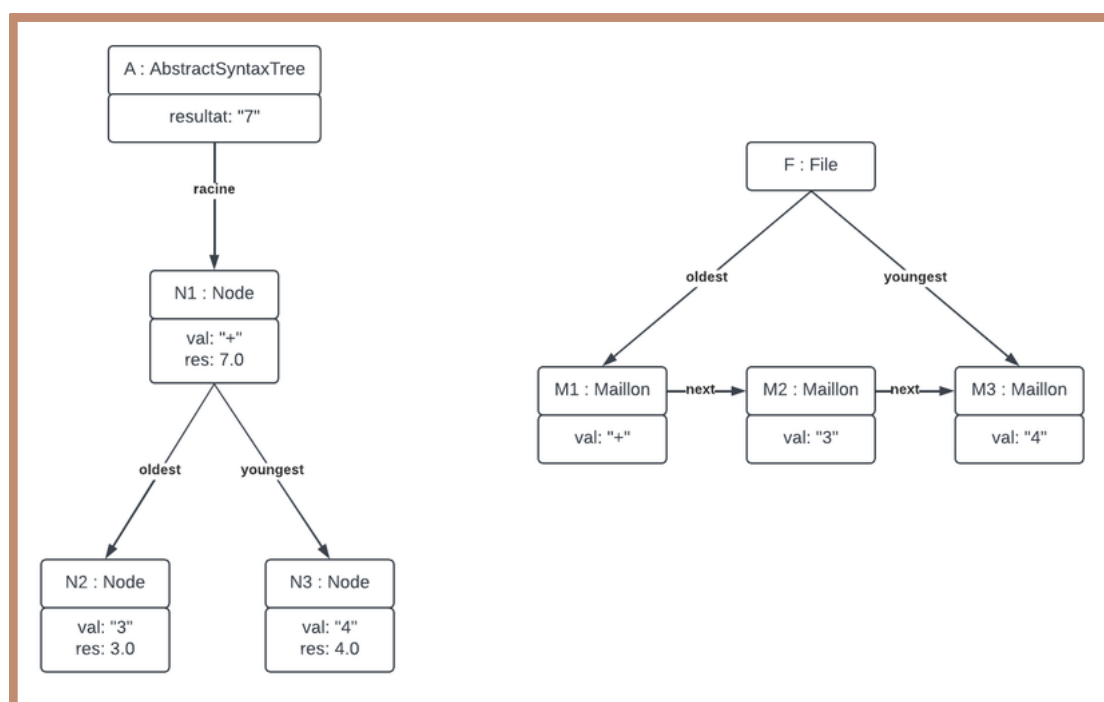
La classe Maillon est un élément d'une liste chaînée, il connaît uniquement le Maillon suivant et il est possible de récupérer sa propre valeur, ou celle du Maillon suivant.

EXEMPLE:

Nous avons une formule de base étant "+ 3 4" dans la cellule.

Le but va être de créer une file de la formule puis de l'utiliser afin de pouvoir utiliser efficacement l'arbre. On commence donc par créer une file qui récupère un à un les éléments pour en faire des maillons contenant chacun un terme ou symbole. On aura donc trois maillon au total contenant les valeurs "+", "3" et "4" dans cet ordre.

Ensuite, on crée l'arbre en lui donnant la file, qu'il va utiliser pour faire la racine uniquement. Lors de la création du nœud, on défile donc une première fois, puisque la valeur du maillon est un symbole, la valeur du nœud devient ce symbole. On lui crée également un fils gauche et un fils droit directement avec le même principe. Donc on défile une deuxième fois pour la création du fils gauche de la racine, et ce n'est pas un symbole donc la valeur du nœud devient la valeur du maillon, "3" dans notre cas. On fait de même avec le fils droit de la racine qui prend pour valeur "4". Pour la gestion des erreurs, tout se fera lors du calcul comme expliqué précédemment.



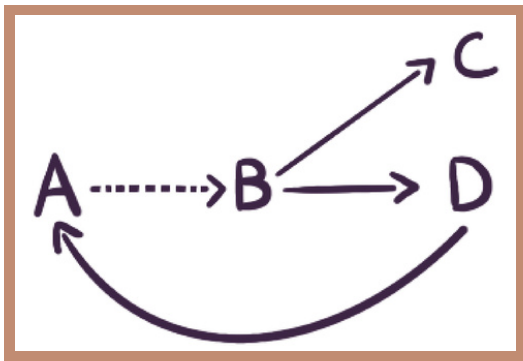
Le diagramme d'objet de la construction d'un arbre pour la formule "+ 3 4"

RÉFÉRENCES CIRCULAIRES

À chaque ajout d'une référence dans un CellContent, qu'on appellera A, on ajoute le CellContent auquel on fait référence à la liste **observing** de A.

On vérifie ensuite l'existence d'un cycle en parcourant la liste **observing** du CellContent auquel on veut faire référence. Si on y trouve A, on sait qu'il y a une référence circulaire.

Sinon, on fait la même vérification pour chacun des CellContent de la liste **observing** du CellContent auquel on veut faire référence, puis, pour ceux de leur liste **observing**, et ainsi de suite jusqu'à ce qu'on ait atteint uniquement des cellules sans référence, ou jusqu'à ce qu'on ait trouvé A dans une des listes **observing**.



On rajoute une référence A vers B.

B dépend de A ?

- > B ne dépend pas de A
- > C ne dépend pas de A
- > D dépend de A : il y a un cycle

STRUCTURES DE DONNÉES ABSTRAITES

ARBRE

Les classes **AbstractSyntaxTree** et **Node** servent à créer des arbres. Ils sont utilisés pour stocker et calculer les formules.

FILE

Les classes **File** et **Maillon** servent à créer des files, qui sont utilisées lors de la création des arbres.

LISTE CHAINÉE

Les listes chaînées sont utilisées pour construire des files, et pour stocker la liste des cellules dont dépend un **CellContent**.

DICTIONNAIRE

Un dictionnaire est utilisé dans la classe **Tableau** pour stocker l'ensemble des **CellContents** en les associant à un identifiant unique.

CONCLUSION

NOLAN

Ce projet m'a permis d'appliquer la majorité des apprentissages des TP de manière plus naturelle dans un environnement plus diversifié grâce au quel j'ai pu m'approprier plus efficacement les fonctionnements de chacun des éléments. Cela est aussi passé par la construction de notre structure de code où l'on a réfléchi aux différentes manières d'arriver à interpréter nos besoins. En effet, on y voit l'utilisation d'une file ou encore les séparations des classes avec par exemple la construction du tableau qui contient des cellules graphiques qui elles mêmes contiennent la formule finale.

Le travail de préparation était donc primordial, autant que celui de finalisation qui m'a permis de mieux comprendre efficacement l'entièreté du code grâce à la réalisation de la documentation ainsi que les interactions des classes entre elles.

Enfin, le fait de travailler avec une nouvelle équipe a été particulièrement intéressant puisque j'ai pu découvrir différentes méthodes de travail, autant dans la réalisation du code que dans la communication et l'établissement des délais. A l'avenir, j'envisage de continuer à utiliser ces différentes méthodes et ces outils de travail tel que de coder en appel vocal pour s'entraider et avancer ensemble ainsi qu'un gros travail en amont sur les bases du projet grâce à des diagrammes.

LUCILE

D'un point de vue technique, ce projet m'a permis de me concentrer sur le côté modélisation des données. Je comprends mieux ce à quoi une file ou un arbre peuvent servir en pratique. J'ai aussi fait de mon mieux pour rendre mes classes réutilisables dans d'autres cas de figures. Enfin, je pense maintenant avoir compris comment fonctionnait un Makefile et comment en réaliser un qui soit fonctionnel.

Le temps passé sur l'algorithme de détection des références circulaires a aussi été très intéressant et m'a poussée à vraiment me pencher sur les interactions entre les classes et sur la façon dont le programme fonctionne.

Au niveau de la coopération, j'ai trouvé la communication au sein de mon groupe parfois compliquée, et j'ai parfois noté un manque de réactivité de la part de mes coéquipiers. Cependant, le travail a tout de même été agréable et, une fois le travail commencé, j'ai beaucoup apprécié travailler avec eux.

FIRMIN

Pour ma part ce projet m'a permis de mieux m'intégrer à mon nouvel environnement en comblant un grand nombre de mes lacunes. J'ai été réellement aidé par les autres membres du groupe qui ont fait preuve de patience et de rigueur Grace à eux je comprends mieux les notions JAVA (Classes, diagrammes....) et je manipule un peu mieux le GIT. J'ai vraiment apprécié ce travail de groupe et je les remercie énormément pour tout.