

SAE3.2 : Tableur

par

Maxime Lelong,

Kohsey Dufour,

Axel Pietrois

Sommaire :

1. Introduction (p.2)
2. Description des fonctionnalités (p.2)
3. Présentation de la structure du programme (p.5)
4. Arbre de syntaxe (p.6)
5. Algorithme de références circulaires (p.8)
6. Structures de données abstraites (p.8)
7. Conclusions (p.9)

1. Introduction

Le but de cette SAÉ est de créer une application Java qui est une version simplifiée d'un tableur. Ce tableur organise les données en une grille de cellules. Les formules de chaque cellule sont entrées avec la notation préfixe et sont stockées sous forme d'arbres binaires.

2. Description des fonctionnalités

Sur le tableur on peut :

- Sélectionner une cellule, ici D6 est sélectionnée



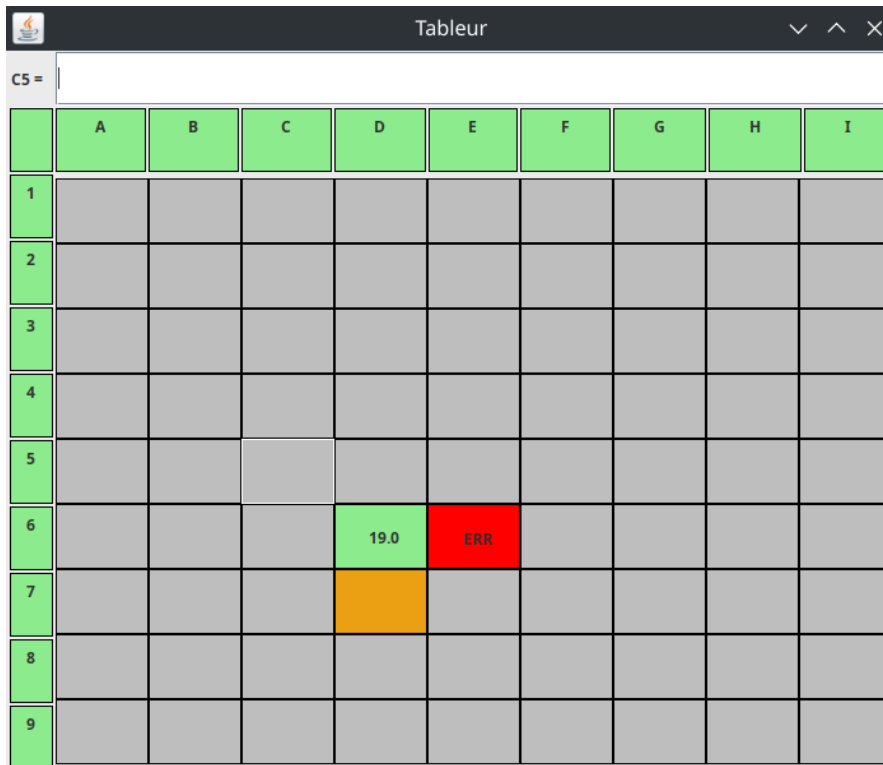
- Entrer une formule en notation préfixe, appuyer sur ENTRÉE pour valider

Tableur									
D6 = + 12 5									
	A	B	C	D	E	F	G	H	I
1									
2									
3									
4									
5									
6				17.0					
7									
8									
9									

- Mettre à jour la formule, appuyer sur ENTRÉE pour valider

Tableur									
D6 = + 12 7									
	A	B	C	D	E	F	G	H	I
1									
2									
3									
4									
5									
6				19.0					
7									
8									
9									

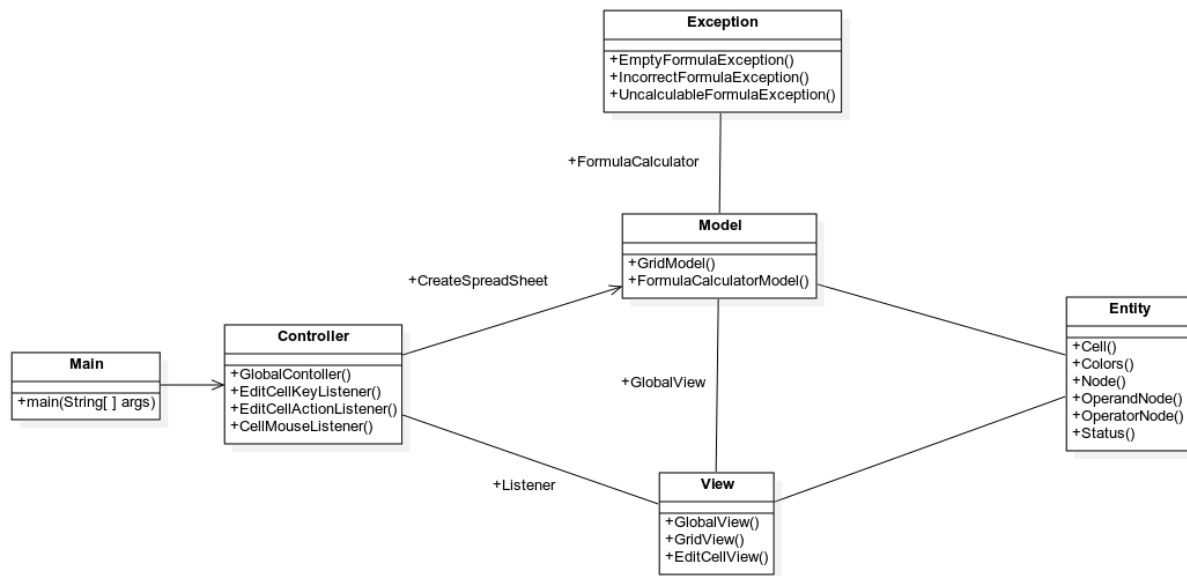
- Voir le statut d'une formule,



	A	B	C	D	E	F	G	H	I
1									
2									
3									
4									
5									
6				19.0	ERR				
7									
8									
9									

- Les statuts possibles pour une cellule sont :
 - Gris = Vide
 - Vert = Formule correcte et calculable
 - Orange = Formule correcte mais incalculable
 - Rouge = Formule incorrecte

3. Présentation de la structure du programme



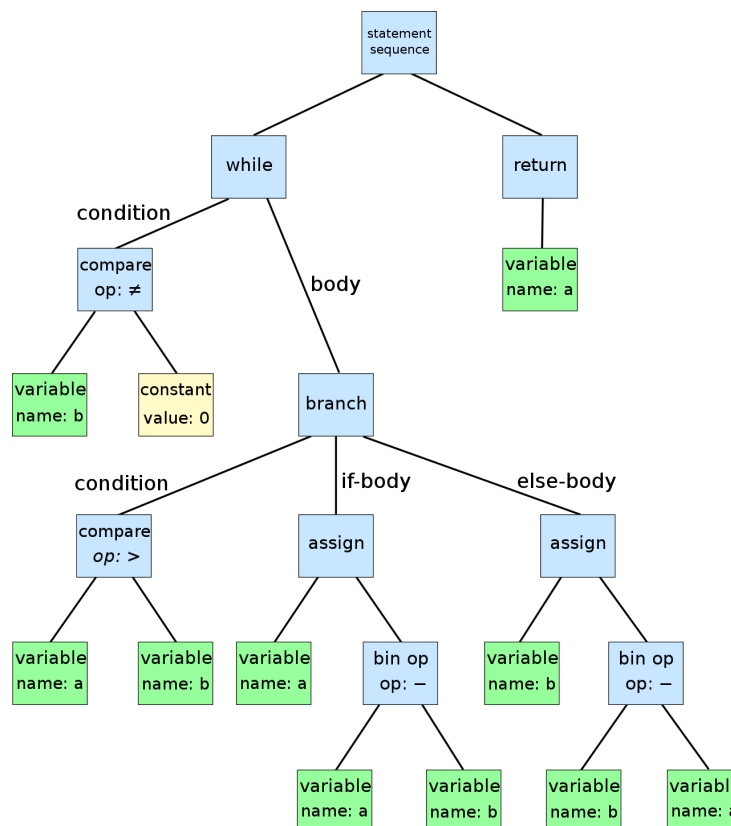
Notre application est décomposé en six packages :

- Main
- Controller
- Model
- Exception
- View
- Entity

La classe Main est utilisée pour lancer le programme et appelle GlobalController. Ce controller appelle le modèle qui va créer le tableau de cases puis créer la vue globale. Une fois la vue créée avec tous les éléments (légende, grille, éditeur de cases...), les Listeners entrent en jeu et sont à l'écoute d'un clic pour sélectionner une case ou mettre à jour celle-ci via la touche entrée dans l'éditeur de formule.

4. Arbre de la syntaxe abstraite

L'arbre de syntaxe abstraite permet de structurer de manière hiérarchique les éléments d'une expression (comme les opérandes et les opérateurs) en fonction de leur relation et de leur ordre d'évaluation. L'arbre de la syntaxe abstraite est un sous-ensemble d'arbre enraciné qui sont des arbres ayant une unique racine. Cette dernière est elle-même un sous-ensemble d'arbre binaire qui est un arbre à deux branches.



Arbre syntaxique abstrait (source : https://fr.wikipedia.org/wiki/Arbre_de_la_syntaxe_abstraite)

Les classes Node, OperandNode, et OperatorNode constituent la structure de base pour construire un arbre de syntaxe abstraite :

Node : Classe abstraite représentant un nœud général de l'AST. Elle définit la méthode `evaluate()`, qui sera implémentée par les sous-classes pour effectuer l'évaluation spécifique du nœud.

OperandNode : Dérivée de Node, cette classe représente un opérande dans l'expression, pouvant être soit une valeur numérique directe, soit une référence à une autre cellule. La méthode `evaluate()` retourne la valeur de l'opérande ou évalue la formule de la cellule référencée.

OperatorNode : Aussi dérivée de Node, il représente un opérateur (par exemple, +, -, *, /) dans l'expression. Il contient des références aux nœuds opérands gauche et droit. La méthode `evaluate()` effectue l'opération spécifiée sur les valeurs évaluées de ces nœuds opérands.

Pour construire l'arbre syntaxique abstraite, l'algorithme suit ces étapes :

Il divise la formule de la cellule en tokens (nombres, opérateurs, références de cellules) en utilisant des espaces comme séparateurs.

Pour chaque token lu de droite à gauche (pour respecter la notation préfixe), il crée un `OperandNode` pour les nombres ou les références de cellules et un `OperatorNode` pour les opérateurs.

Dans le cas d'un `OperatorNode`, il retire les deux nœuds supérieurs de la pile et les assigne comme opérands gauche et droit du nœud opérateur, puis repousse ce nœud opérateur dans la pile.

Ce processus crée un arbre où chaque nœud opérateur relie correctement ses opérands, respectant ainsi la structure et la hiérarchie de l'expression originale.

Le nœud restant dans la pile à la fin du processus est la racine de l'arbre de syntaxe abstraite, qui peut alors être évaluée pour obtenir le résultat de la formule.

Exemple concret de la construction d'un arbre avec l'expression $(3+5)*2$ ou $* + 3 5 2$ en notation préfixe avec la méthode `buildASTForFormula` de la classe `FormulaCalculatorModel`.

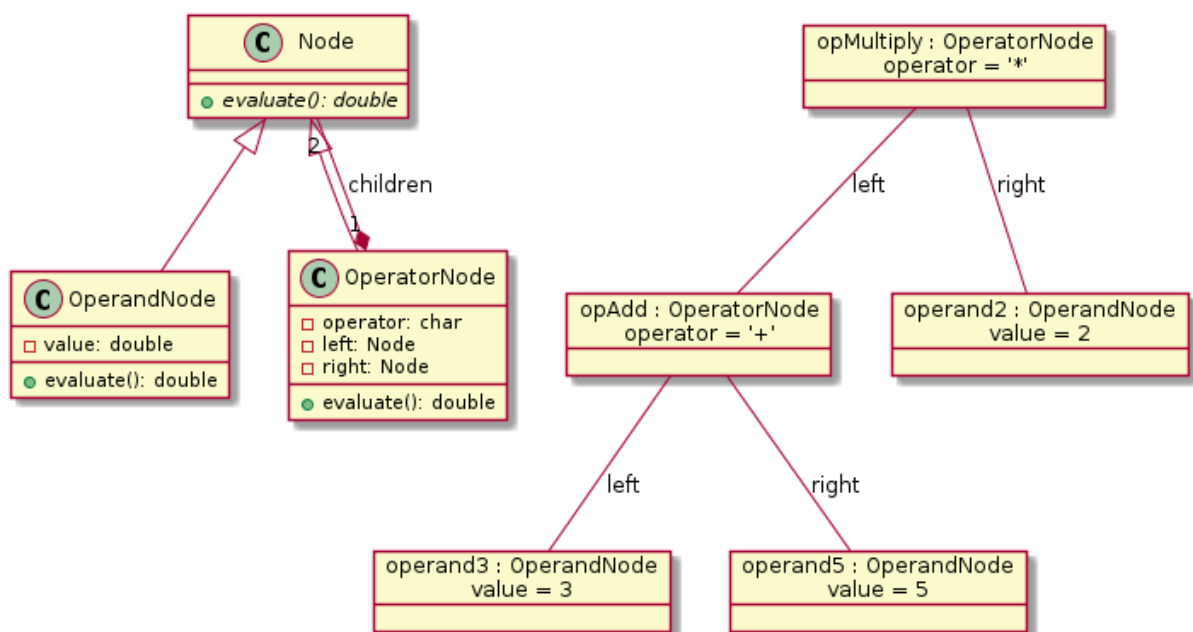


Diagramme de Classe et Diagramme objet d'un arbre à partir de l'expression $(3+5)*2$

5. Algorithme de références circulaires

Cet algorithme détecte les références circulaires dans une feuille de calcul représentée. Chaque cellule peut contenir une formule faisant référence à d'autres cellules.

La fonction commence par vérifier si la cellule courante est nulle, auquel cas il n'y a pas de référence circulaire.

Elle extrait la formule de la cellule et recherche des références à d'autres cellules en utilisant une expression régulière.

Pour chaque référence trouvée dans la formule :

- Si cette référence a déjà été visitée (contenue dans `visitedReferences`), une référence circulaire est détectée.
- Sinon, la référence est ajoutée à l'ensemble des références visitées.

Ensuite, l'algorithme récupère la cellule référencée et rappelle récursivement la fonction pour vérifier la présence de références circulaires dans cette nouvelle cellule.

Après avoir vérifié une branche de références, la référence courante est retirée de l'ensemble des références visitées pour permettre la vérification d'autres chemins potentiels sans fausses détections de référence circulaire.

La fonction renvoie `true` si une référence circulaire est détectée à n'importe quel niveau de récursivité, sinon elle continue jusqu'à ce que toutes les références aient été vérifiées sans détecter de circularité, auquel cas elle renvoie `false`.

6. Structures de données abstraites

Dans le projet, on a utilisé un arbre syntaxique abstrait (abstract syntax tree, AST en anglais) pour stocker les formules. Pour aider à la création de la formule, on a aussi utilisé une Pile.

7. Conclusions

Maxime

Tout d'abord, j'ai trouvé le sujet intéressant. Ce projet m'a permis d'apprendre de nouvelles fonctionnalités du java. Et contrairement aux autres projets, nous avons bien organisé les tâches de chacun, tout le monde a participé au projet et produit des fonctionnalités et grâce à ça nous avons pu finir à l'heure le projet. Pour cela on a utilisé des branches (feature/Prénom) pour que chacun puisse avancer à son rythme sans empiéter sur le travail des autres. Personnellement, je m'occupais de mettre à jour la branche master en implémentant les fonctionnalités de chacun et j'ai aussi aidé pour la vue et la structuration du code ainsi que les initialisations. Durant le projet, nous avons réfléchi aussi à des fonctionnalités supplémentaires comme la sauvegarde du tableur et l'ouverture d'un tableau et nous voulions aussi implémenter la sélection de cases dans les formules comme dans excel. Malheureusement, nous n'avons pas eu le temps de mettre en place ces fonctionnalités et nous avons préféré nous concentrer sur les fonctionnalités obligatoires.

Kohsey

Ce projet a été une expérience enrichissante et pleine d'apprentissage pour moi. J'ai été chargé de gérer le calcul préfixe, de concevoir l'arbre syntaxique abstrait et de détecter les références circulaires, ce qui m'a permis d'approfondir mes connaissances en algorithmie et de mettre en pratique les théories apprises en cours. L'organisation du travail en tâches distinctes au sein de notre équipe s'est avérée efficace, permettant une progression rapide et une intégration harmonieuse des différentes composantes du projet. La clarté du sujet y a contribué. Cette méthodologie de travail claire et structurée a non seulement facilité notre collaboration, mais a également contribué au succès du projet. Cette expérience a renforcé mes compétences en développement logiciel et a souligné l'importance du travail d'équipe dans la réalisation d'objectifs communs. Maxime, Axel et moi formons une bonne symbiose.

Axel

Ce projet m'a permis d'approfondir mes compétences d'utilisation de git avec l'utilisation des branches. La séparation des tâches m'a permis de travailler de façon assez régulière sur l'affichage sans forcément devoir attendre l'avancée des autres membres du groupe, tout en devant avancer assez vite pour pouvoir permettre aux autres de voir visuellement leurs avancées.