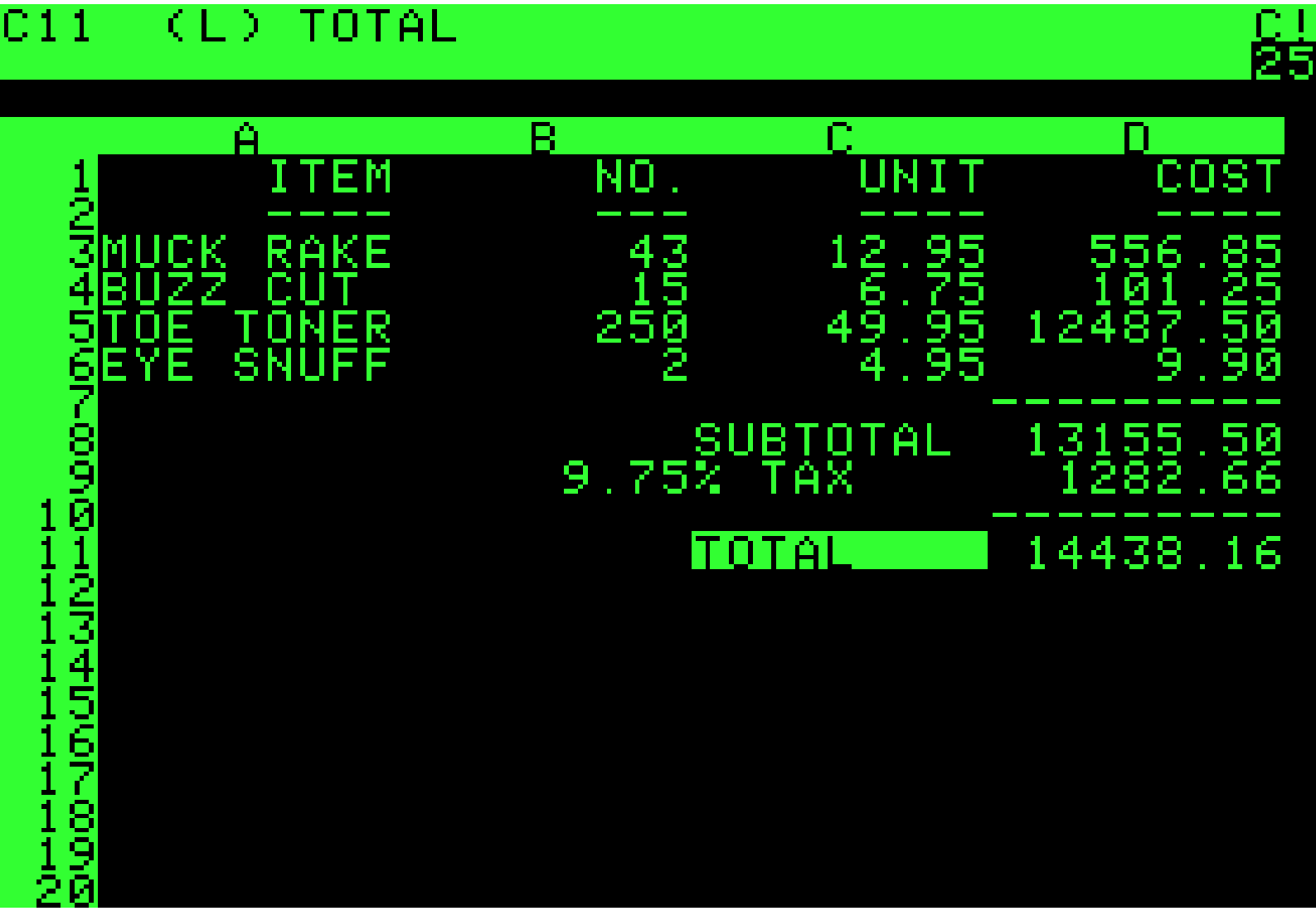
SAE32_2023

PETIT TABLEUR EN JAVA



Baptiste Blanchon, Ayoub AMMARA & Samet DURAN

SOMMAIRE

INTRODUCTION

Le projet consiste à créer un tableur simplifié en Java. Un tableur est un outil utilisé pour organiser des données sous forme de grille. Notre version sera une grille de 9 lignes et 9 colonnes, où chaque cellule peut contenir soit une valeur soit une formule mathématique.

L'objectif est de permettre à l'utilisateur de saisir des formules en utilisant des nombres réels, des références de cellules et des opérateurs mathématiques de base (+, -, *, /). Ces formules doivent respecter une syntaxe préfixée et seront évaluées pour afficher les résultats dans les cellules correspondantes.

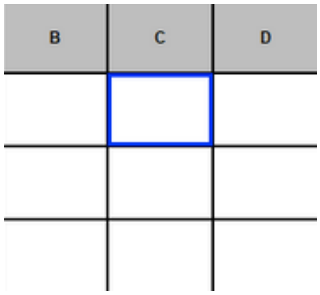
Le programme doit être capable de distinguer visuellement quatre états différents pour chaque cellule en fonction de la validité syntaxique et du calcul de la formule. De plus, toute modification dans une cellule entraînera la mise à jour des cellules dépendantes dans un ordre approprié pour refléter les changements.

Pour optimiser les performances lors des calculs répétés, nous utiliserons une représentation en arbre de syntaxe abstraite (AST) pour chaque formule valide. Cette approche permettra une évaluation efficace des formules sans avoir à les retraiter à chaque modification.

FONCTIONNALITÉS

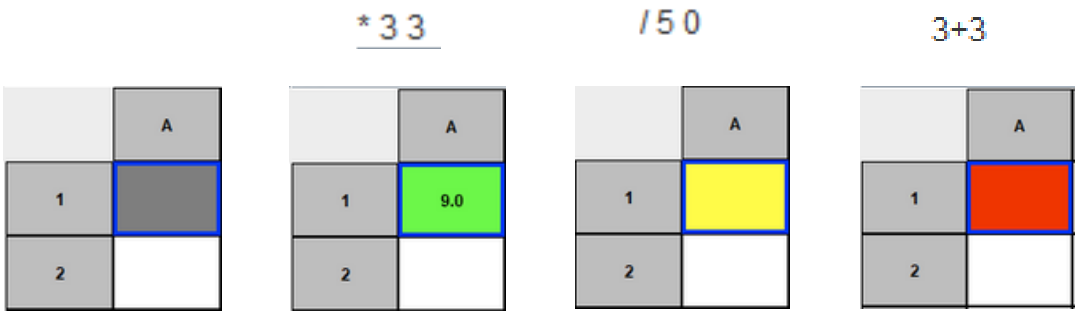
Notre tableur dispose des fonctionnalités suivantes :

- Lorsque l'on clique sur une cellule du tableur, une bordure délimite la cellule actuellement sélectionnée, pour gagner en visibilité :

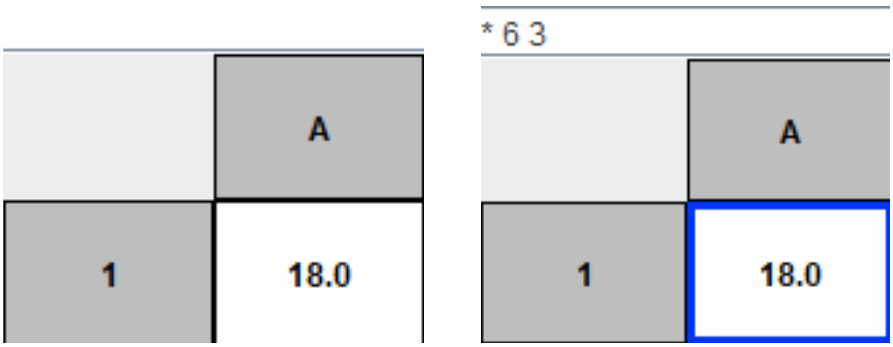


Chaque cellule peut se trouver quatre situations possibles visuellement distinctes :

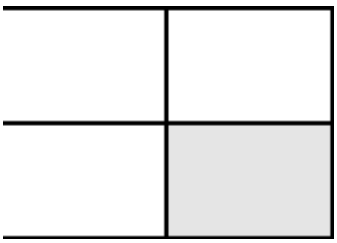
- La cellule contient une formule vide (on clique sur la cellule et on fait Entrée (gris)
- La cellule contient une formule correcte et calculable (notation préfixée) (vert)
- La cellule contient une formule correcte mais incalculable (division par 0) (jaune)
- La cellule contient une formule incorrecte (notation normale au lieu de préfixée (rouge)



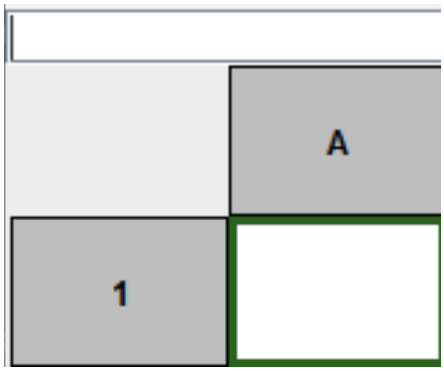
- Lorsque l'on sélectionne une cellule dans la feuille de calcul, celle-ci est visuellement mise en avant et sa formule apparaît dans la barre de formules. Ici, l'utilisateur n'a pas entré l'entier 18 dans la cellule A1, mais plutôt la formule "* 6 3" que l'on retrouve en haut :



- Lorsque l'on passe la souris sur les cellules du tableur, la couleur change pour améliorer le côté esthétique. Ici on peut voir dans le coin inférieur droit une couleur proche du gris qui nous montre ce changement :



- A chaque clic sur une cellule, le focus se fait automatiquement sur le barre de formule, pour ne pas avoir à cliquer sur la cellule puis ensuite sur la barre de formule. Comme on peut le voir, le curseur est présent après avoir cliqué sur A1.



STRUCTURE DU PROGRAMME

La création du Makefile a été le premier point abordé lors de la réalisation du projet. Il permet de réaliser plusieurs actions :

- compiler le programme (make build),
- compiler et lancer le programme (make run),
- supprimer les fichiers générés lors de la compilation (make clean).
- créer une archive jar du projet et permettre son exécution (make jar puis make run-jar).

Avant de commencer le projet, une réflexion sur la structure du code a été essentielle. Nous avons choisi de diviser le programme en plusieurs classes, chacune ayant un rôle spécifique, pour faciliter la compréhension et l'organisation de notre code.

AbstractSyntaxTree s'occupe de la construction et de l'évaluation des arbres syntaxiques abstraits pour les formules des cellules. Elle contient des méthodes pour valider et traiter les expressions préfixées, construire l'arbre à partir de la formule, et évaluer l'arbre syntaxique. Elle joue un rôle crucial dans l'interprétation et le calcul des formules insérées dans les cellules.

Cell représente une cellule individuelle dans la feuille de calcul. Elle stocke des informations telles que sa position (ligne et colonne), sa référence de cellule, et la formule qu'elle contient. Cette classe étend Observable pour notifier les changements dans la cellule à d'autres composants qui pourraient en dépendre.

CellLabel est conçue pour afficher le contenu d'une cellule dans une feuille de calcul, ce qui correspond à sa description. Elle gère l'apparence visuelle de la cellule et peut mettre en évidence son état en fonction des opérations..

CellMouseListener gère les événements de clic de souris sur les cellules. Cette classe met à jour l'apparence de la cellule sélectionnée et interagit avec la barre de formules

CellNode est un type de nœud dans l'AbstractSyntaxTree, représentant une cellule. Elle évalue la formule de la cellule référencée, en gérant les exceptions comme les formules incalculables ou incorrectes.

CellOperations fournit des méthodes statiques pour convertir entre les références de cellules et leurs coordonnées.

ExitButtonListener, FormulaBarListener et FormulaBarComponentAdapter gèrent les interactions de l'utilisateur avec les éléments de l'interface, comme la barre de formules et le bouton “quitter”.

HomePage est la fenêtre principale de l'application. Elle organise les différents JPanel et gère la navigation dans l'application.

IncalculableFormulaException et IncorrectFormulaException définissent des exceptions personnalisées pour gérer les erreurs spécifiques liées aux formules des cellules.

Main est la classe principale du projet, elle initialise et affiche ce qu’on a appelé HomePage (la page principale / d’accueil pour faire simple).

Node, NumberNode et OperatorNode sont abstraites (Node) et concrètes et définissent les différents types de nœuds dans l'AbstractSyntaxTree, permettant l'évaluation des formules.

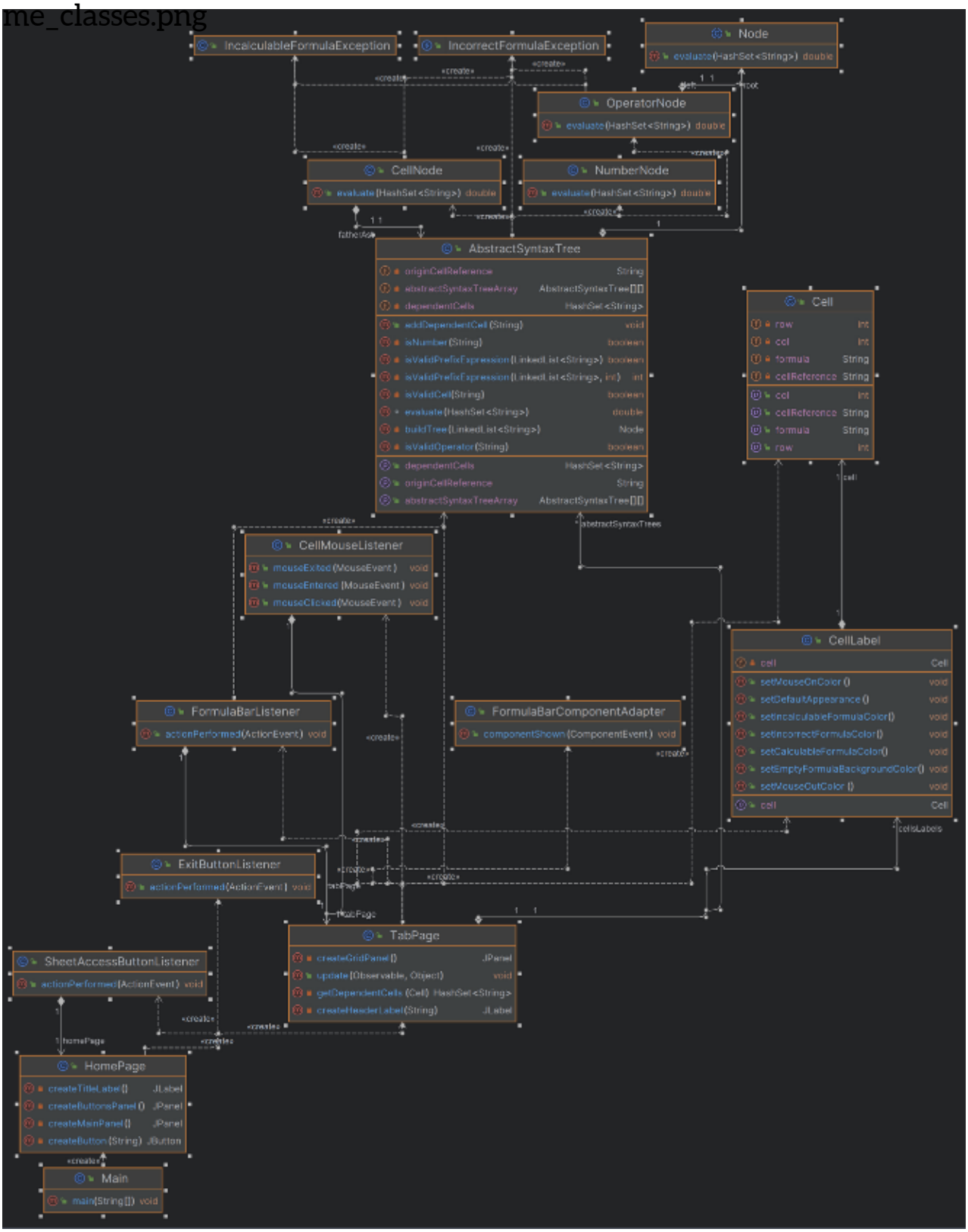
Observable et Observer sont des interfaces définissant les méthodes pour le modèle observateur-observé, utilisé dans la gestion des formules de cellules.

OperatorNode est un type de nœud dans l'AbstractSyntaxTree, représentant un opérateur. Il évalue l'expression en fonction de l'opérateur et des nœuds enfants.

SheetAccessButtonListener gère les actions sur le bouton d'accès au tableur.

TabPage organise l'affichage des cellules et gère les interactions au sein de la grille du tableur.

Cliquer ici pour voir le diagramme de classes de notre projet. Il est recommandé de le consulter via l'adresse suivante pour plus de lisibilité : https://dwarves.iut-fbleau.fr/gitiut/duran/SAE32_2023/src/branch/master/uml/diagramme_classes.png



ARBRE DE SYNTAXE ABSTRAIT

Principales classes de l'AST

Il n'y a que quelques classes impliquées dans l'arbre de syntaxe abstraite.

AbstractSyntaxTree représente l'arbre de syntaxe abstraite pour une formule donnée, voici ses propriétés principales :

- Node root : c'est la racine de l'AST
- String [] validOperators : ce sont les opérateurs valides ("+","-","*","/")
- evaluate() : c'est une méthode qui évalue l'expression représentée par l'AST

Node représente un nœud dans l'AST. C'est une classe abstraite qui a différentes sous-classes pour différents types de nœuds :

- OperatorNode représente un opérateur
- NumberNode représente un nombre
- CellNode représente une référence à une autre cellule dans le tableur

Cell représente une cellule dans le tableur, voici ses propriétés principales :

- int, row, col : coordonnées de la cellule
- String formula : formule de la cellule
- setFormula() définit la formule de la cellule et notifie les observateurs

CellOperations fournit des méthodes statiques pour des opérations liées aux cellules, comme la conversion des références de cellules en coordonnées.

Exemple concret

Dans cet exemple, nous alons détailler le processus d'analyse et d'évaluation d'une formule en notation préfixe, par exemple "+ 2,66 * B7 0,33" à l'aide de l'arbre de syntaxe abstraite (AST) de notre tableur.

La formule "+ 2,66 * B7 0,33" est sous la forme préfixée où l'opérateur précède les opérandes. Dans ce cas, l'opérateur principal est "+", et il a pour opérandes "2,66" et le résultat de la multiplication de “B7” par “0,33”. La première étape est la construction de l’arbre de syntaxe abstraite.

1. Initialisation de l’AST :

- Un AbstractSyntaxTree est créé pour représenter la structure de la formule.
- Le nœud racine est un OperatorNode correspondant à l'opérateur "+".

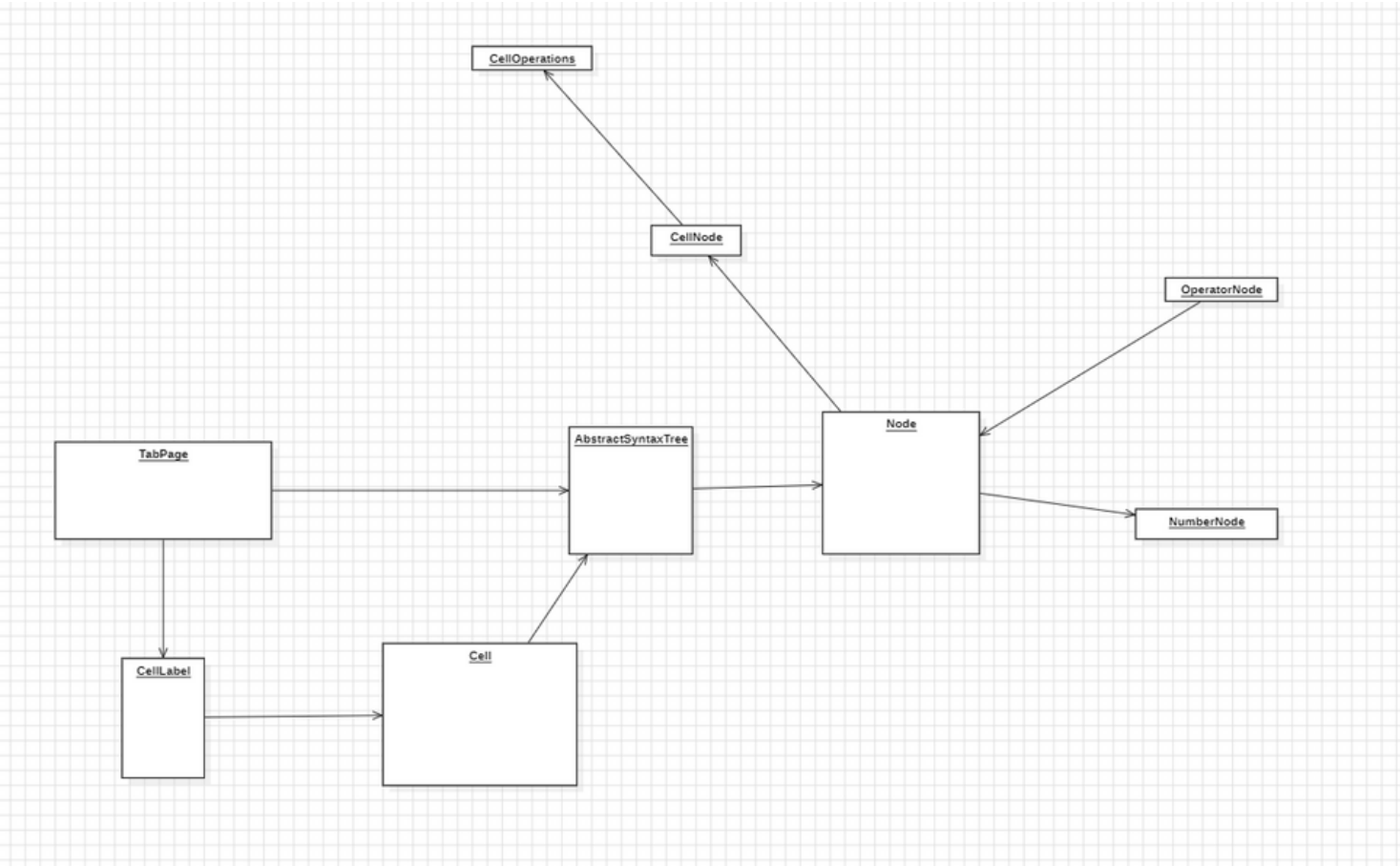
2. Construction des Sous-Arbres :

- Le premier enfant du nœud root est un NumberNode pour "2,66".
- Le second enfant est un OperatorNode pour "*", qui représente la multiplication.
- Ce nœud représentant la multiplication a lui même deux enfants :
  - Un CellNode pour "B7", qui fait référence à la valeur contenue dans la cellule B7 du tableur.
  - Un NumberNode pour "0,33".

Après avoir crée l'AST, on l'évalue en commençant au root (à la racine).

- L'OperatorNode de "+" appelle la méthode evaluate() sur chacun de ses enfants.
- Le NumberNode de "2,66" retourne simplement sa valeur.
- L'OperatorNode de "*" évalue ses enfants (la valeur de la cellule "B7" et "0.33"), effectue la multiplication, et donne le résultat.
- Le nœud racine additionne ensuite 2,66 au résultat de la multiplication pour produire la valeur finale.

Diagramme d'objets



Cliquer ici pour voir le diagramme d'objets de notre projet. Il est disponible au format PNG et SVG et il est recommandé de le consulter via l'adresse suivante pour plus de lisibilité : https://dwarves.iut-fbleau.fr/gitiut/duran/SAE32_2023/src/branch/master/uml/diagramme_classes.png

ALGORITHME DE REFERENCES CIRCULAIRES

Notre algorithme vise à identifier les cas où une formule dans une cellule fait référence, directement ou indirectement, à elle-même, créant ainsi une boucle infinie. Cette détection se fait durant l'évaluation de l'AST de chaque formule.

1. Initialisation :

Chaque fois qu'une formule doit être évaluée, un HashSet<String> nommé visited est créé. Ce HashSet va garder la trace des références de cellules déjà visitées durant l'évaluation.

2. Visite de chaque nœud de l'AST :

- Lors de l'évaluation d'un nœud de l'AST, l'algorithme vérifie si ce nœud est un CellNode (c'est-à-dire une référence à une autre cellule).
- Si c'est le cas, l'algorithme récupère la référence de la cellule ciblée.

3. Vérification des références circulaires :

- Avant de poursuivre l'évaluation de la cellule ciblée, l'algorithme vérifie si la référence de cette cellule existe déjà dans le HashSet visited.

- Si la référence est déjà présente dans visited, cela signifie que la formule courante a déjà accédé à cette cellule auparavant, indiquant ainsi une référence circulaire. Dans ce cas, une exception IncorrectFormulaException est lancée, signalant une référence circulaire.

4. Ajout de références à “visited” :

- Si aucune référence circulaire n'est détectée, la référence de la cellule actuelle est ajoutée à visited.
- L'algorithme continue ensuite l'évaluation de l'AST.

5. Propagation de “visited” :

- Lorsque l'AST contient des CellNode, l'algorithme évalue récursivement les formules des cellules référencées, en passant le HashSet visited à chaque appel récursif.
- Cela garantit que toutes les cellules visitées dans le cadre de l'évaluation d'une formule spécifique sont prises en compte pour la détection des références circulaires.

STRUCTURE DE DONNÉES ABSTRAITES

Notre projet utilise plusieurs structures de données abstraites, comme par exemple LinkedList<> que nous avons vu en cours, pour ne citer qu'elle. Voici la liste de ces structures abstraites :

l'Arbre de Syntaxe Abstraite (AbstractSyntaxTree) :

- Vu pendant les cours sur les arbres, il représente la structure hiérarchique d'une formule. Il permet d'analyser, de représenter et d'évaluer les formules saisies dans les cellules du tableur.

Le nœud (Node) et ses sous-classes (OperatorNode, NumberNode et CellNode) :

- ce sont les composants de base de l'AST représentant les différents éléments (opérateurs, nombres, références de cellules) d'une formule. Ils sont utilisés pour construire l'AST et évaluer les formules.

HashSet<String> :

- c'est une collection d'objets qui ne permet pas de doublons, utilisée pour stocker des éléments uniques. Le HashSet empêche les références circulaires dans les formules et gère les dépendances entre cellules.

LinkedList<String> :

- c'est une liste chaînée qui permet un accès séquentiel aux éléments. Elle est utilisée pour stocker et manipuler les composants d'une formule pendant son analyse syntaxique.

Tableau en deux dimensions (AbstractSyntaxTree[][]) :

- c'est une structure qui permet de stocker des éléments dans une grille à deux dimensions. On l'utilise pour représenter les AST associés à chaque cellule dans le tableur, facilitant l'accès et la gestion des formules.

Observable :

- c'est un patron de conception observateur qui sert à surveiller les changements d'état. Cela permet aux cellules du tableur d'être observées pour des changements, comme la mise à jour des formules.

CONCLUSIONS PERSONNELLES

Baptiste: Ce projet a été enrichissant pour moi, de par la découverte en profondeur des arbres en Java, qui n'a pas été évident au premier abord. Le plus compliqué dans ce projet était de travailler avec Observable et de gérer les références circulaires. Cela a nécessité plus de temps que prévu pour garantir la fiabilité et le bon fonctionnement de notre tableur. Cependant, malgré ces difficultés, j'ai apprécié concevoir l'aspect visuel du tableur. La réalisation de l'arbre de syntaxe abstraite a également été une partie très enrichissante du projet, étant elle aussi une des plus difficiles. Nous espérons, en rendant ce projet, avoir pris compte de nos erreurs passés, du moins nous en avons tiré le meilleur.

Samet: Ce projet, sélectionné pour sa simplicité, a constitué une transition efficace avant le début du second semestre. Il s'est distingué par sa mise en œuvre fluide, avec peu de difficultés. Le principal défi a été de rendre les cellules observables, un problème que nous avons réussi à résoudre sans trop de complications. D'autres problèmes mineurs ont également été rapidement identifiés et corrigés. Ce projet m'a permis d'améliorer mes compétences en Java. Nous avons tiré des leçons de nos erreurs passées et avons bénéficié d'une communication améliorée au sein du groupe.

Ayoub: Ce projet de développement d'un tableur en Java a été une expérience formatrice et révélatrice. En concevant un tableur simplifié, j'ai pu explorer les subtilités de la programmation orientée objet et de l'interface utilisateur. La création de classes telles que Cell et AbstractSyntaxTree a été particulièrement instructive, m'enseignant l'importance d'une architecture logicielle claire et modulaire.

La manipulation des formules dans le tableur, notamment la gestion des opérateurs arithmétiques et des références de cellules, a renforcé mes compétences en parsing et en évaluation d'expressions. De plus, travailler en équipe et utiliser le serveur Gitea pour un développement collaboratif a souligné l'importance de la communication et de la documentation dans les projets logiciels.

En résumé, ce projet a été un défi stimulant qui a enrichi mes compétences en programmation et en conception de logiciels, tout en me préparant pour des projets futurs plus complexes.