SAE 3.2 – RAPPORT DE PROJET

Souidi Lyanis – Moguljak Tom – Dimitrijevic Hugo

PETIT TABLEUR

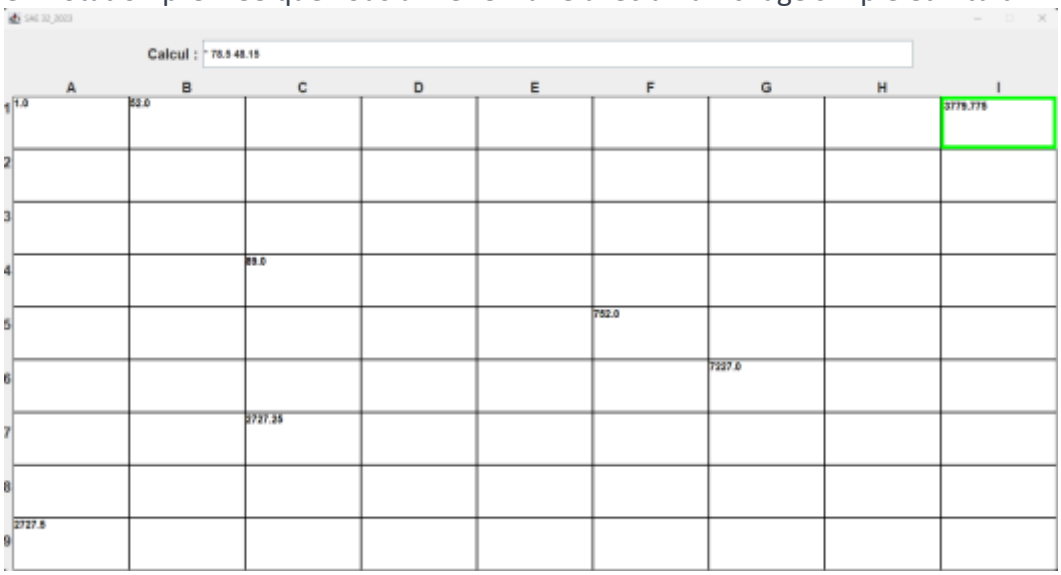
Introduction :

Le projet consiste à développer un tableur simplifié en Java, une application capable de gérer des données numériques dans une grille de 9 lignes et 9 colonnes. Chaque cellule de cette grille peut contenir une formule, initialement vide, basée sur des constantes réelles, des références à d'autres cellules, ou des opérateurs arithmétiques (+, -, *, /) en notation préfixe. L'application permet aux utilisateurs de saisir et de modifier ces formules, offrant une interaction directe et intuitive avec les données.

Les Fonctionnalités :

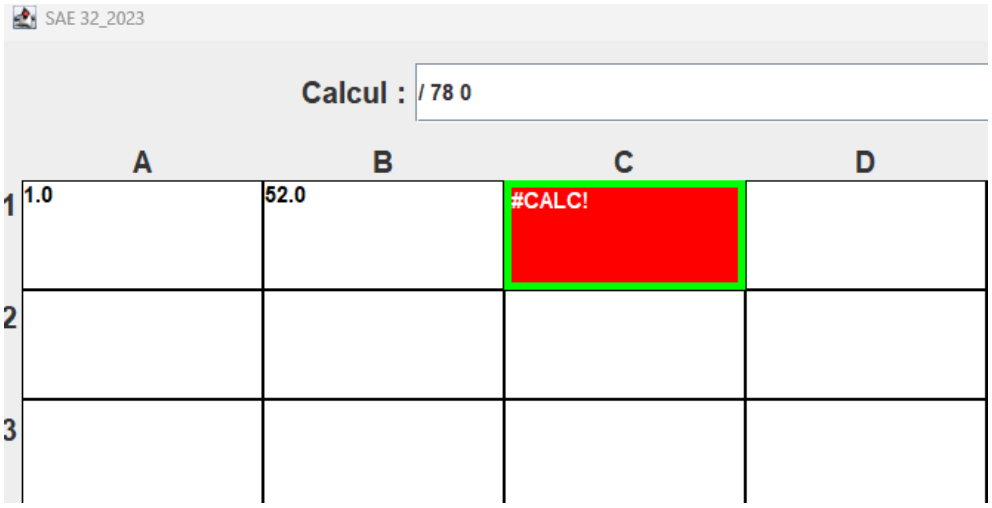
- Une grille de 9x9 fonctionnelle

  Cette grille de 9x9 est capable de stocker et vous donner les résultats de tous les calculs en notation préfixée que vous aimeriez faire avec un affichage simple et intuitif



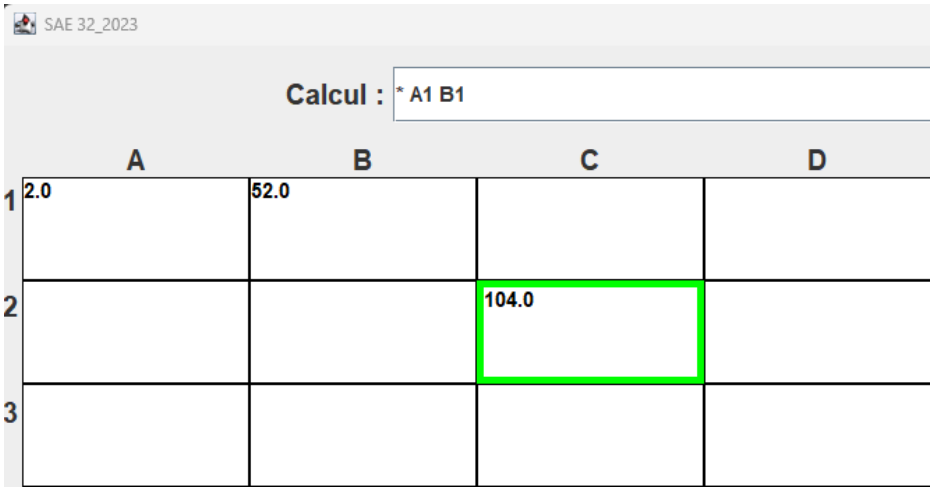
- Une gestion des erreurs

  Une faute de frappe dans un calcul est vite arrivée, et aucun risque de casser votre tableau si vous entrez une division par 0. L'application vous le diras sous la forme d'une erreur « CALC ».



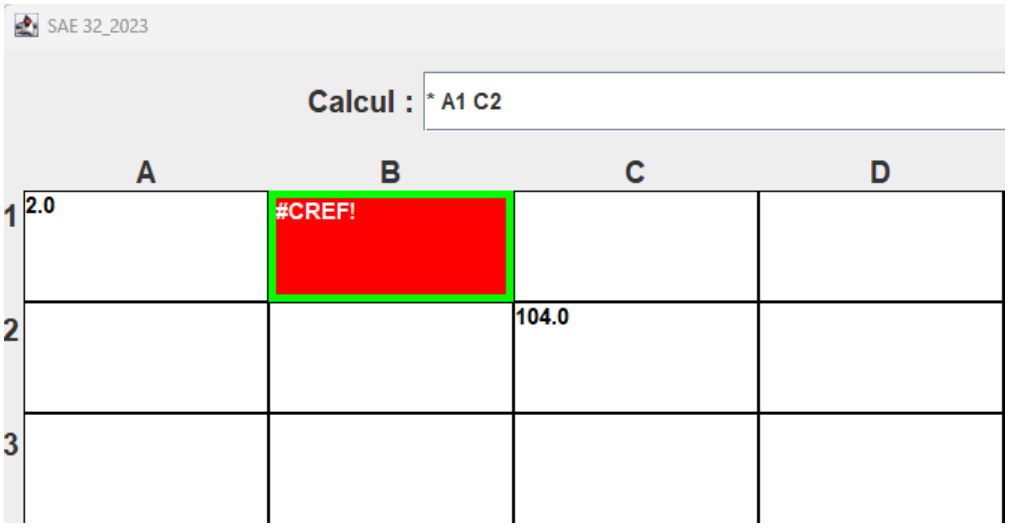
- Utiliser les cellules pour faire des calculs

  Vous souhaitez faire un tableau complexe avec des cellules qui interagissent entres elles et qui se mettent à jour automatiquement.
  Alors vous êtes sur la bonne application vous pouvez utiliser des cellules comme valeurs.



- Une gestion des références circulaire

  Votre tableau devient de plus en plus complexe et vous entrez la mauvaise cellule en entré pour votre calcul, pas de panique l'application détecte les références circulaires vous empêchant de vous retrouver avec des boucles sans fins avec l'erreur « CREF ».



Structure du programme :

Voici l'organisation des différentes parties de l'application :

1. **Gestion des Formules avec les Nœuds (TreeNode et ses dérivés)** :
   Au cœur de l'application, on trouve une série de classes pour gérer les formules. **TreeNode** est la classe de base pour représenter les éléments d'une formule. De là, on a plusieurs sous-classes comme **AdditionTreeNode**, **SubtractionTreeNode**, **MultiplicationTreeNode**, **DivisionTreeNode**, **NumberTreeNode**, et **ReferenceTreeNode**. Chacune de ces classes représente une opération ou un élément spécifique dans une formule. Cela permet de décomposer une formule complexe en parties gérables.

2. **Architecture MVC pour les Cellules et les Feuilles de Calcul** :
   Pour les cellules, nous avons **Cell** (le modèle), **CellController** (le contrôleur), et **CellView** (la vue). La même structure est appliquée aux feuilles de calcul avec les classes **Worksheet**, **WorksheetController**, et **WorksheetView**. Cela sépare bien la logique des données (comme le stockage des valeurs des cellules) de la logique de l'interface utilisateur (comme l'affichage et la mise à jour des cellules).

3. **Interface Utilisateur et Contrôle** :
   WindowController et **WindowView** gèrent l'interface utilisateur de l'application. Ces classes sont responsables de la fenêtre principale, des menus, et d'autres éléments d'interface, fournissant ainsi l'interaction avec l'utilisateur.

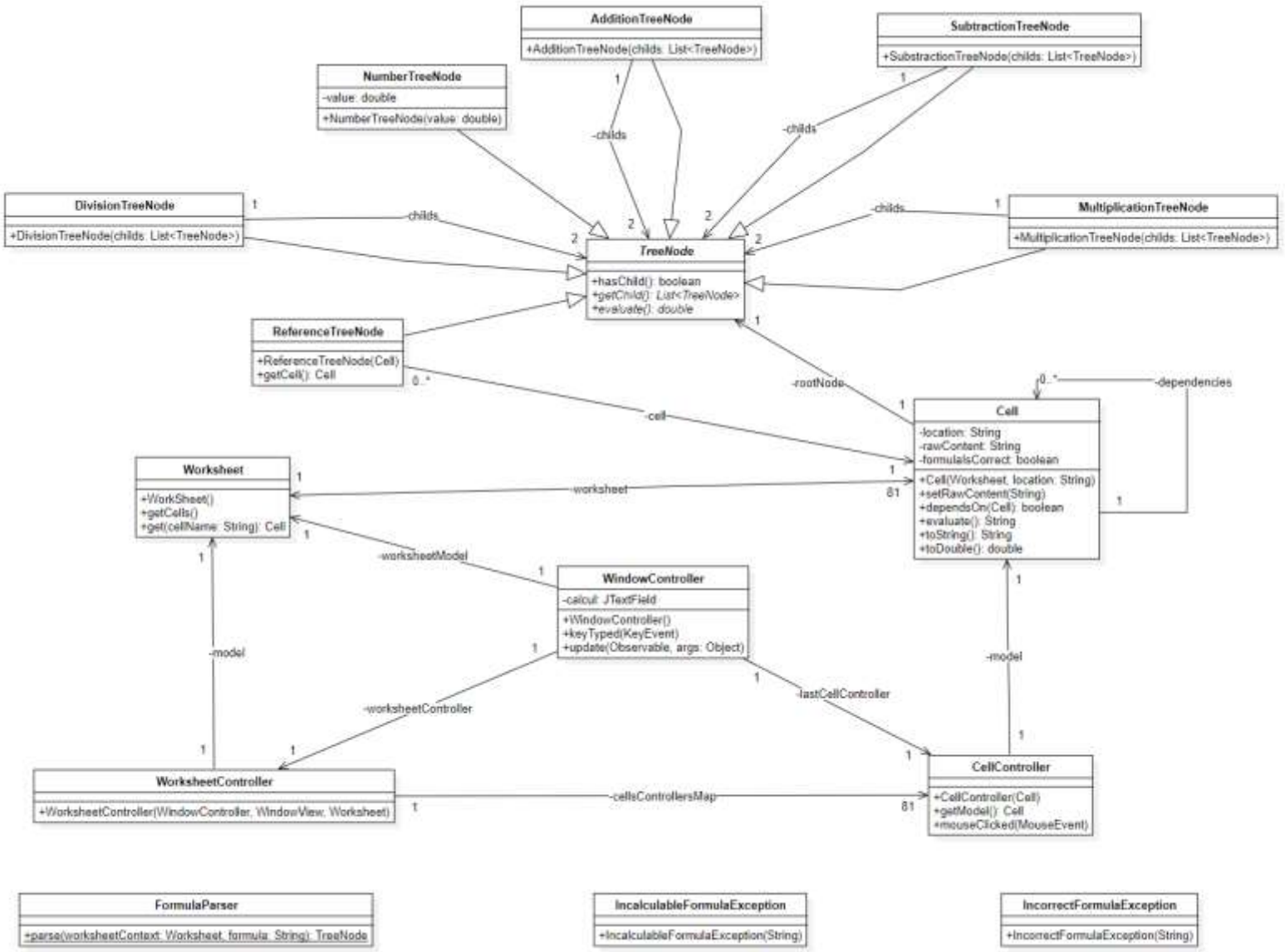
4. **Traitement des Formules** :
   La classe **FormulaParser** joue un rôle crucial. Elle analyse et interprète les formules entrées dans les cellules, transformant une chaîne de texte en une structure exploitable pour les calculs.

5. **Gestion des Erreurs** :

Avec **IncalculableFormulaException** et **IncorrectFormulaException**, l'application peut gérer des erreurs spécifiques liées aux formules, comme une formule incorrecte ou un calcul qui ne peut pas être effectué.

6. **Démarrage de l'Application** :
   Enfin, la classe **Main** est le point de départ de l'application, où tout le processus démarre.

Voici le diagramme de classe simplifié :



Classes de l'Arbre de Syntaxe Abstraite :

Exemple concret :

Considérons la formule + 2.66 * B7 0.33. Cette formule effectue une addition entre 2.66 et le produit de la valeur dans la cellule B7 et 0.33. (Pour faire simple on considère que B7 contient « 1 »).

Voici le cheminement :

- **La Racine (AdditionTreeNode)** :
  Ce nœud représente l'opération principale de la formule, qui est l'addition. Il indique que nous allons effectuer une addition entre deux éléments.

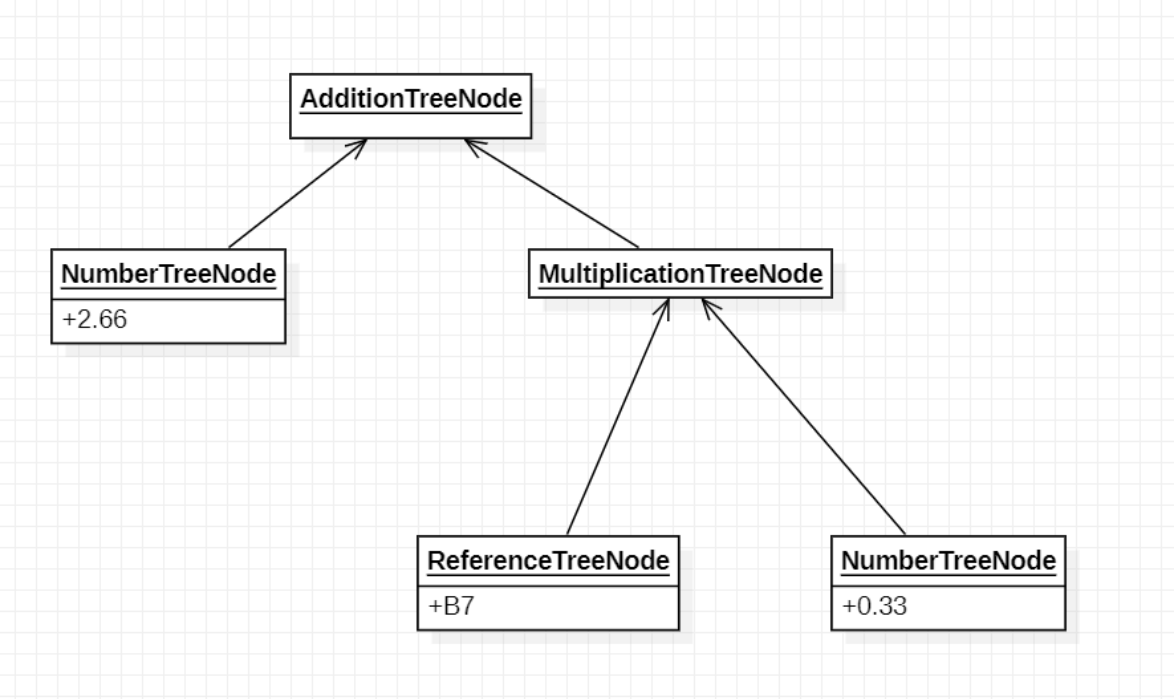
- **Premier Enfant (NumberTreeNode avec 2.66)** :
  Ce nœud est un opérande de l'addition et contient la valeur numérique 2.66. Il représente la première valeur à ajouter.

- **Deuxième Enfant (MultiplicationTreeNode)** :
  Ce nœud gère une sous-opération de la formule, en l'occurrence une multiplication. Il s'agit d'une opération intermédiaire nécessaire pour obtenir le second opérande de l'addition.

- **Premier Sous-Enfant (ReferenceTreeNode pour B7)** :
  Ce nœud fait référence à la cellule B7 du tableur, dont la valeur sera utilisée dans notre opération de multiplication.

- **Deuxième Sous-Enfant (NumberTreeNode avec 0.33)** :
  Il fournit la deuxième valeur numérique pour la multiplication, qui est 0.33.

Voici le schéma correspondant pour simplifier la compréhension :



Algorithme de Détection des Références Circulaires :

L'algorithme que nous avons implémenté pour détecter les références circulaires dans notre tableur fonctionne en suivant les étapes suivantes :

- **Analyse de la Formule** :
  À partir d'une cellule donnée, l'algorithme examine sa formule, extraite à l'aide de classes comme **FormulaParser**.

- **Parcours des Références** :
  Si la formule contient des références à d'autres cellules (gérées par **ReferenceTreeNode** dans l'AST), ces références sont parcourues récursivement.

- **Vérification des Boucles** :
  Pendant le parcours, on maintient un ensemble des cellules déjà visitées. Si on revisite une cellule, cela indique une boucle, signalant une référence circulaire.

- **Optimisation et Performance** :
  L'implémentation tient compte des performances, en évitant les vérifications redondantes, particulièrement pertinentes pour les formules complexes référençant plusieurs cellules.

- **Rétroaction Utilisateur** :
  En cas de détection d'une référence circulaire, l'utilisateur est informé, potentiellement via des mécanismes impliqués dans **CellView** ou **WorksheetView**.

Cette approche assure que les formules entrées dans le tableur ne créent pas de références circulaires, garantissant ainsi la stabilité des calculs dans l'application. L'algorithme s'appuie sur la structure et les relations définies dans l'AST pour effectuer cette vérification de manière efficace.

Structures de données abstraites :

Parmi les structures de données abstraites que nous avons utilisé dans notre projet mais aussi vu durant les cours, on peut lister :

- Des listes
- Des dictionnaires
- Des stacks
- Une classe abstraite (TreeNode)

Ceci nous a permis d'avoir une un programme fonctionnel qui répond à la consigne demandée.

Conclusion personnelle :

- Lyanis Souidi :

Ce projet fut intéressant à réaliser, j'ai aimé développer le parser car c'était un des éléments les plus complexes et importants de ce projet. Il y a pleins de possibilités d'évolutions donc on a essayé de faire en sorte de pouvoir ajouter des fonctionnalités facilement.

- Tom Moguljak :

En conclusion, la conception d'un tableur simplifié a été l'occasion pour moi d'explorer des concepts avancés tels que la gestion des formules, la validation syntaxique, et la manipulation efficace de structures de données vues en cours.

- Hugo Dimitrijevic :

Personnellement ce projet a été intéressant car le concept de refaire un Excel est assez amusant, et contrairement au projet précédent cela était plus simple d'avoir une idée du programme final. Je pense que nous avons fait quelque chose de simple à l'image de la consigne et que le tout est plutôt réussit.