NOM : MAHAMAT GARBA HADJE ACHE et SIDI SYLLA

L2 MIAGE

TD7 JAVA

1. LA CLASSE CALCULATRICE :

Cette classe ‘Calculatrice’ est une implémentation d'une calculatrice qui prend une expression mathématique en chaîne de caractères, la valide, la transforme en notation postfixée (notation polonaise inversée) et enfin évalue cette expression postfixée pour donner un résultat. La méthode ‘calculer(String equation)’ est la méthode principale de la classe. Elle prend en entrée une chaîne représentant une expression mathématique, la valide d'abord à l'aide du validateur syntaxique. Si l'expression est valide, elle la transforme ensuite en notation postfixée. Enfin, elle évalue cette expression postfixée à l'aide de l'évaluateur postfixe pour obtenir le résultat final. Si l'expression est invalide du point de vue de la syntaxe, elle lance une exception **‘ExceptionArithmetique’** avec le message **"Syntaxe invalide".**

2. EvaluateurPostfixe: Cette classe évalue l'expression en notation postfixée pour obtenir le résultat numérique final. Elle utilise une pile pour stocker les opérandes temporaires lors de l'évaluation de l'expression.

3. ExceptionArithmetique : La classe appelée **ExceptionArithmetique** éxtend la classe **RuntimeException**. Cette classe est utilisée pour représenter une exception spécifique liée à des erreurs arithmétiques qui peuvent survenir lors de l'exécution de certaines opérations mathématiques dans le contexte de la calculatrice.

4. L ‘Interface Expression : Ce code définit une interface nommée **‘Expression’** ; l'interface ‘**Expression**’ déclare une seule méthode abstraite nommée **‘evaluer()’** qui retourne un double. Cette interface est conçue pour être implémentée par des classes qui représentent des expressions mathématiques. La méthode **‘evaluer()’** est destinée à être mise en œuvre par ces classes pour évaluer une expression mathématique et retourner un résultat numérique sous forme de double.

Ainsi, toute classe qui implémente l'interface ‘**Expression**’ doit fournir une implémentation concrète de la méthode ‘**evaluer()’,** qui effectue le calcul spécifique à cette expression et retourne le résultat. Cela permet de créer un ensemble de classes qui représentent différentes expressions mathématiques et de les évaluer de manière cohérente à travers une interface commune.

5.OPERATEUR : Cette classe Operateur est une énumération qui représente différents opérateurs mathématiques utilisés dans une calculatrice ; Elle définit des constantes énumérées pour différents opérateurs mathématiques tels que l'addition, la soustraction, la multiplication, la division, et les parenthèses ouvrantes et fermantes. Elle a un constructeur privé qui prend en paramètres le symbole de l'opérateur et sa priorité. Elle fournit des méthodes pour obtenir le symbole et la priorité de chaque opérateur. Elle fournit également une méthode statique **fromSymbole** qui permet d'obtenir l'opérateur correspondant à un symbole donné. Elle a une méthode statique **estOperateurValide** qui vérifie si un caractère donné est un opérateur valide. Elle contient une méthode **appliquer** (actuellement non implémentée) qui est destinée à être utilisée pour appliquer l'opérateur à deux opérandes. Cette classe énumération fournit une représentation structurée des opérateurs mathématiques avec leurs symboles et priorités associés, ainsi que des méthodes pour accéder à ces informations et vérifier la validité des opérateurs.

6.OPERATION : Cette classe représente une opération mathématique, telle que l'addition, la soustraction, la multiplication ou la division, entre deux expressions. La classe contient trois champs privés **: gauche, droite** et **operateur**, qui représentent respectivement l'expression de gauche, l'expression de droite et l'opérateur de l'opération mathématique. Elle a un constructeur qui prend trois paramètres : les expressions de gauche et de droite, ainsi que l'opérateur. Ces valeurs sont utilisées pour initialiser les champs correspondants de l'objet **Operation.** La classe implémente l'interface **Expression, donc** elle doit fournir une implémentation pour la méthode **evaluer()** définie dans cette interface. Dans la méthode **evaluer(),** l'opération mathématique est effectuée en évaluant d'abord les expressions de gauche et de droite, puis en appliquant l'opérateur approprié. La méthode utilise une instruction **switch** pour sélectionner l'opération à effectuer en fonction de l'opérateur spécifié. Si l'opérateur est une division et que le diviseur (la partie droite de l'opération) est égal à zéro, une exception **ArithmeticException** est lancée pour éviter une division par zéro. Si l'opérateur spécifié n'est pas pris en charge (par exemple, un opérateur inconnu), une exception **IllegalArgumentException** est lancée pour signaler une erreur d'opérateur inconnu.

En résumé, cette classe **Operation** représente une opération mathématique entre deux expressions, avec une méthode **evaluer()** qui retourne le résultat de cette opération.

7.TransformateurExpression : "**TransformateurExpression**". Cette interface comporte une méthode appelée "**transformer**" qui prend en paramètre une chaîne de caractères représentant une équation mathématique et retourne une chaîne de caractères. L'objectif de cette interface est de définir un contrat pour les classes qui implémentent cette interface. Ces classes devront fournir une implémentation de la méthode "**transformer**" qui prend une équation mathématique dans un certain format et la transforme en une autre forme, par exemple, en convertissant une équation infixée en une équation postfixée ou en effectuant d'autres transformations nécessaires pour l'évaluation de l'expression.

Bref, cette interface fournit un moyen standardisé pour les différentes classes de transformation d'expressions d'indiquer comment elles effectuent la transformation des équations mathématiques.

8.TransformateurInfixeVersPostfixe : est une classe concrète implémentant l'interface "**TransformateurExpression**". Son rôle est de convertir une expression mathématique en notation infixée en notation postfixée. Ce qu'il fait :

La méthode "**transformer**" prend une chaîne de caractères représentant une équation mathématique en notation infixée et retourne une chaîne de caractères représentant la même équation en notation postfixée. Elle utilise une pile d'opérateurs pour stocker les opérateurs rencontrés lors de la conversion. Elle parcourt chaque caractère de l'équation en entrée. Si le caractère est un espace blanc, il est ignoré, si le caractère est un chiffre, l'algorithme extrait le nombre complet et l'ajoute à la chaîne postfixée. Si le caractère est une parenthèse ouvrante '(', elle est ajoutée à la pile d'opérateurs. Si le caractère est une parenthèse fermante ')', l'algorithme vide la pile d'opérateurs jusqu'à trouver une parenthèse ouvrante correspondante, ajoutant les opérateurs retirés à la chaîne postfixée. Si le caractère est un opérateur, l'algorithme compare sa priorité avec celle de l'opérateur en haut de la pile d'opérateurs. Les opérateurs ayant une priorité plus élevée sont retirés de la pile et ajoutés à la chaîne postfixée.

Une fois que tous les caractères de l'équation ont été traités, l'algorithme vide la pile d'opérateurs et ajoute les opérateurs restants à la chaîne postfixée. Enfin, la chaîne postfixée est retournée.

En gros, cette classe permet de transformer une expression mathématique de notation infixée en notation postfixée, ce qui facilite son évaluation ultérieure.

9.ValidateurExpression : "**ValidateurExpression**". Cette interface comporte une méthode appelée "**valider**" qui prend en paramètre une chaîne de caractères représentant une équation mathématique et retourne un booléen indiquant si l'équation est valide ou non. Elle vérifie si l'expression est écrite correctement du point de vue de la syntaxe mathématique (par exemple, si les parenthèses sont correctement équilibrées, si les opérateurs sont correctement utilisés, etc.).

Bref, cette interface fournit un moyen standardisé pour les différentes classes de validation d'expressions d'indiquer comment elles valident les équations mathématiques.

10.ValidateurOperandes : Ce classe représente une classe nommée "**ValidateurOperandes**" qui implémente l'interface "**ValidateurExpression**". Son rôle est de valider les opérandes dans une équation mathématique. La méthode "**valider**" prend une chaîne de caractères représentant une équation mathématique en entrée et retourne un booléen indiquant si les opérandes dans l'équation sont corrects ou non. La méthode parcourt chaque caractère de l'équation en entrée. Si le caractère est un espace blanc, il est ignoré. Si le caractère est une parenthèse ouvrante '(' ou fermante ')', cela est traité comme une séquence d'opérandes valides. Si le caractère est un chiffre, cela indique un début ou une continuation d'un opérande. Si le caractère est un point décimal '.', il est vérifié s'il est déjà présent dans l'opérande ou s'il suit directement un opérateur. Si le caractère n'est ni un chiffre ni un point décimal, il est considéré comme un opérateur. Dans ce cas, il est vérifié s'il y a un opérande

valide précédent. Le processus continue jusqu'à la fin de l'équation. Enfin, la méthode retourne vrai si la dernière expression de l'équation est un opérande valide, sinon faux.

En gros, cette classe valide les opérandes dans une équation mathématique en vérifiant leur structure et leur placement par rapport aux opérateurs.

11.ValidateurOperateurs :

"**ValidateurOperateurs**" implémente l'interface "**ValidateurExpression**". Son rôle est de valider la présence d'opérateurs valides dans une équation mathématique. La méthode "**valider**" prend une chaîne de caractères représentant une équation mathématique en entrée et retourne un booléen indiquant si tous les opérateurs dans l'équation sont valides ou non. La méthode parcourt chaque caractère de l'équation en entrée. Si le caractère n'est pas un espace blanc, une parenthèse ouvrante '(' ou fermante ')', alors il est considéré comme un opérateur potentiel. Si le caractère n'est ni un chiffre ni un opérateur valide, la méthode retourne faux, indiquant qu'un opérateur invalide a été trouvé. Si tous les caractères de l'équation sont vérifiés et aucun opérateur invalide n'est trouvé, la méthode retourne vrai, indiquant que tous les opérateurs sont valides.

En résumé, cette classe valide la présence d'opérateurs valides dans une équation mathématique en vérifiant chaque caractère pour s'assurer qu'il représente soit un chiffre, soit un espace blanc, soit une parenthèse, soit un opérateur valide défini par une autre classe appelée "Operateur".

12.ValidateurParantheses :

"**ValidateurParentheses**" i implémente l'interface "**ValidateurExpression**". Son rôle est de valider l'équilibre des parenthèses dans une équation mathématique. La méthode "**valider**" prend une chaîne de caractères représentant une équation mathématique en entrée et retourne un booléen indiquant si toutes les parenthèses dans l'équation sont correctement appariées ou non. La méthode parcourt chaque caractère de l'équation en entrée. Si le caractère est une parenthèse ouvrante '(', le compteur est incrémenté. Si le caractère est une parenthèse fermante ')', le compteur est décrémenté. Si à un moment donné le compteur devient négatif, cela signifie qu'une parenthèse fermante a été trouvée sans qu'il y ait une ou plusieurs parenthèses ouvrantes correspondantes précédentes, ce qui rend l'équation invalide. Une fois que tous les caractères de l'équation ont été parcourus, la méthode vérifie si le compteur est égal à zéro. Si c'est le cas, cela signifie que toutes les parenthèses ouvrantes ont été appariées avec des parenthèses fermantes, sinon, l'équation contient des parenthèses mal appariées et la méthode retourne faux.

En résumé, cette classe valide l'équilibre des parenthèses dans une équation mathématique en comptant le nombre de parenthèses ouvrantes et fermantes et en vérifiant s'il y a le même nombre de chacune. Si le nombre de parenthèses fermantes dépasse le nombre de parenthèses ouvrantes à un moment donné, l'équation est considérée comme invalide.

13.ValidateurSyntaxe :

"**ValidateurSyntaxe**" implémente l'interface "**ValidateurExpression**". Son rôle est de valider la syntaxe d'une équation mathématique en utilisant une liste de validateurs spécifiques. La classe "**ValidateurSyntaxe**" prend en entrée une liste d'objets **ValidateurExpression** lors de son initialisation. La méthode "**valider**" parcourt cette liste de validateurs. Pour chaque validateur dans la liste, la méthode appelle sa méthode "**valider**" avec l'équation en entrée. Si un des validateurs renvoie faux, cela signifie

qu'une règle de syntaxe spécifique n'est pas respectée dans l'équation, donc la méthode "**valider**" retourne faux. Si tous les validateurs renvoient vrai, cela signifie que toutes les règles de syntaxe spécifiques sont respectées dans l'équation, donc la méthode "valider" retourne vrai.

En résumé, cette classe permet de valider la syntaxe d'une équation mathématique en utilisant une liste de validateurs spécifiques, ce qui permet de séparer la logique de validation en plusieurs sous-composants spécialisés.

14.MAIN : C'est la classe main. Ce que fais ce code :

a. Il importe les classes nécessaires, y compris ArrayList pour stocker les validateurs d'expression et différentes classes liées à la calculatrice.

b. Crée une liste appelée **"validateurs"** pour stocker des objets de type **ValidateurExpression.**

c. Ajout de différents validateurs à la liste **"validateurs"**, notamment **ValidateurParentheses, ValidateurOperateurs** et **ValidateurOperandes.**

d. Crée un objet **ValidateurExpression** nommé **"validateurSyntaxe"** en utilisant un **ValidateurSyntaxe**, en lui passant la liste **"validateurs**".

e. Crée un objet **TransformateurExpression** nommé "transformateur" en utilisant un **TransformateurInfixeVersPostfixe.**

f. Crée un objet **EvaluateurPostfixe** nommé **"evaluateur".**

g. Crée un objet **Calculatrice** nommé **"calculatrice"**, en lui passant les objets **"validateurSyntaxe", "transformateur"** et **"evaluateur".**

h. Définition d'un tableau de chaînes **"expressionsValides"** contenant des expressions arithmétiques valides.

i. Définition d'un tableau de chaînes "**expressionsInvalides**" contenant des expressions arithmétiques invalides.

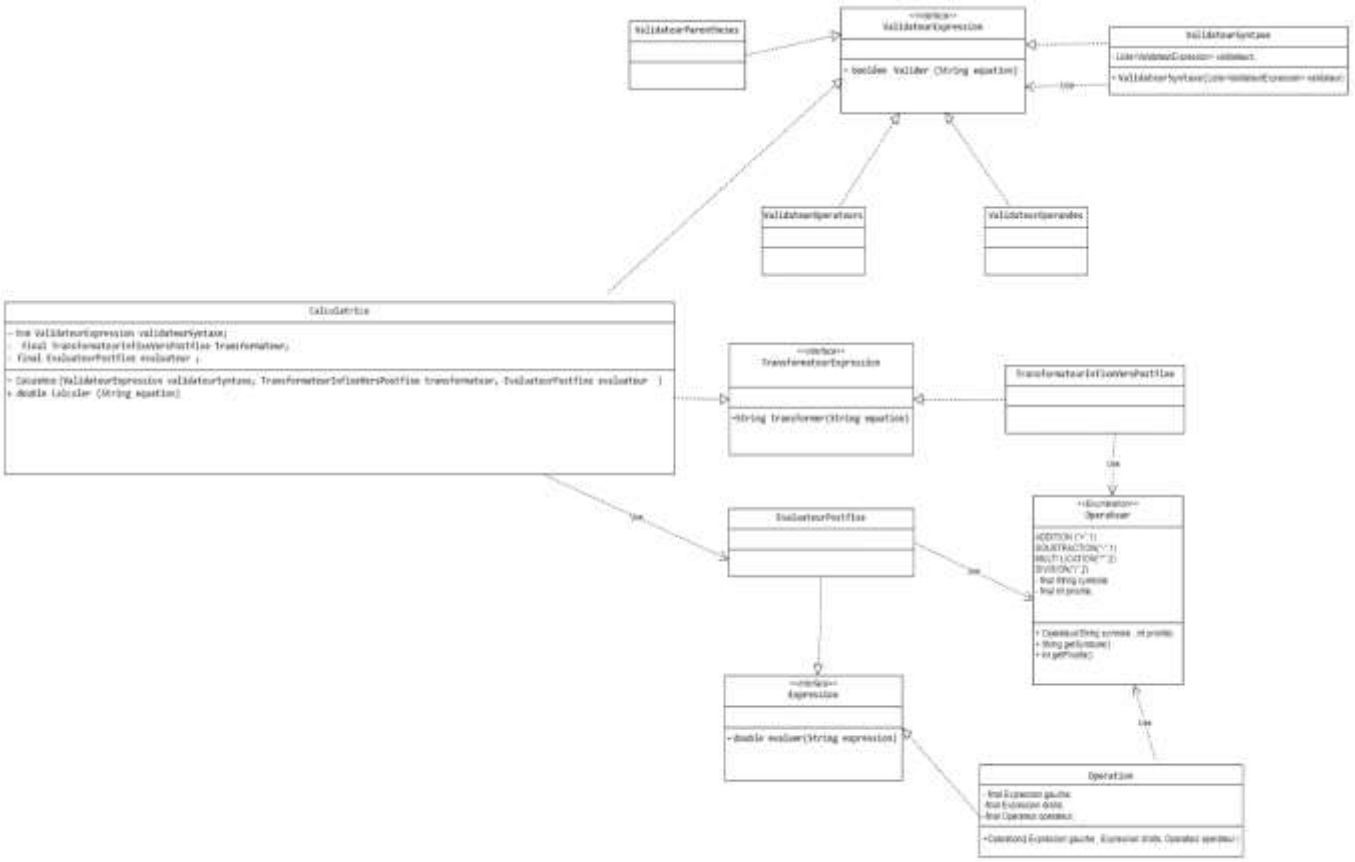
j. Itération à travers chaque expression valide dans "**expressionsValides**", tentant de calculer le résultat à l'aide de la calculatrice. Si une exception arithmétique est levée, elle est capturée et affichée.

k. Itération à travers chaque expression invalide dans "**expressionsInvalides**", tentant également de calculer le résultat à l'aide de la calculatrice. Si une exception arithmétique est levée, elle est affichée.

l. Les résultats ou les erreurs sont affichés à la console.

En résumé, ce programme de test crée une calculatrice avec différents validateurs d'expression, puis teste son fonctionnement en évaluant une série d'expressions arithmétiques valides et invalides, affichant les résultats ou les erreurs associées.

Diagramme de classe



Difficultés rencontrées :

1. **Gestion des exceptions :** Nous avons rencontré des défis lors de la gestion des exceptions, des parenthèses non équilibrées des operateur composés ou un operateur à la fin d'une expression et ls espaces.

2. **Validation des expressions :** La mise en place des validateurs pour vérifier la syntaxe et la validité des expressions mathématiques a été un processus complexe

Nous avons effectué des tests approfondis de notre projet en utilisant un ensemble diversifié d'expressions mathématiques. Jusqu'à présent, notre application a fonctionné correctement pour toutes les expressions testées.

Il est difficile de prévoir toutes les combinaisons d'expressions possibles, il est possible qu'une expression spécifique puisse ne pas être correctement évaluée. Nous aimerons bien de nous fournir des retours d'informations sur toute expression qui ne semble pas être correctement traitée.