|  |
| --- |
| Haute Ecole d’Ingénierie et de Gestion |
| POO Laboratoire 7 |
| Calculatrice |

|  |
| --- |
| Sebastian Diaz & Dunant Guillaume  04/12/2023 |

## Introduction

Le laboratoire vise à élaborer une calculatrice fonctionnant selon la notation polonaise inverse (RPN) en utilisant des classes Java distinctes. Le fichier JCalculator.java, fourni par l'assistant et l'enseignant, établit l'interface graphique d'une calculatrice encore incomplète. Pour achever sa mise en œuvre, il est nécessaire d'implémenter la classe Operator et ses sous-classes. L'accent sera mis sur la maximisation de la factorisation des opérations au sein de la hiérarchie de classes, éliminant ainsi les constructions conditionnelles explicites. La notation polonaise inverse sera basée sur une pile (Stack) que nous devrons créer. Cette approche permet des calculs sans avoir recours aux parenthèses, et la pile sera conçue de manière modulaire pour assurer une structure flexible. Enfin, la classe State gérera les états internes de la calculatrice, la représentation de la pile, ainsi que la synchronisation avec l'interface graphique.

## Classe Stack

La classe Stack représente une implémentation de la structure de données de pile. Elle offre les fonctionnalités classiques d'une pile, permettant l'ajout (push) et la suppression (pop) d'éléments selon le principe LIFO. La pile est générique, ce qui signifie qu'elle peut stocker des éléments de n'importe quel type. La classe propose également des méthodes pour obtenir la taille de la pile (size), vider la pile (emptyStack), et récupérer ses éléments sous forme de tableau ou de chaîne de caractères. Un itérateur est également disponible pour parcourir les éléments de la pile.

La classe Stack est conçu de manière modulaire avec un nœud (Node) interne qui représente chaque élément de la pile. Les opérations sur la pile sont effectuées en manipulant les liens entre ces nœuds.

## Classe Node

La classe Node représente les éléments individuels d'une pile. Chaque nœud contient une valeur de type générique et une référence vers le nœud précédent dans la pile. Les nœuds sont utilisés pour construire la structure de la pile, permettant ainsi une gestion efficace des éléments selon le principe Last In, First Out.

Il y a deux constructeurs qui permet de créer le nœud. Le premier en spécifiant la valeur uniquement, et l’autre en indiquant également son précédant. La classe offre des méthodes pour accéder et modifier la valeur du nœud (getValue, setValue), ainsi que pour obtenir et définir le nœud précédent (getPrevious, setPrevious).

## Classe Iterator

La classe Iterator implémente un itérateur simple permettant de parcourir les nœuds d'une pile. L'itérateur est initialisé avec un nœud de départ, généralement le sommet de la pile. La méthode next permet de récupérer la prochaine valeur non encore retournée dans le parcours, en se déplaçant vers le nœud précédent. La méthode hasNext indique si une valeur suivante est disponible en vérifiant si le nœud précédent existe.

Cet itérateur offre une manière pratique de traverser les éléments de la pile sans avoir à manipuler directement la structure interne des nœuds.

## Classe Operator

La classe Operator, en tant que classe abstraite centrale, offre une structure modulaire pour diverses opérations spécifiques. On distingue deux catégories principales : UnaryOperation, regroupant les opérations à une seule opérande telles que la racine carrée, la puissance et l'inverse. Puis nous avons DoubleOperation, dédiée aux opérations à deux opérandes (Addition, Subtraction, Multiplication, Division). En plus de ces catégories, d'autres sous-classes d'Operator sont présentes pour des opérations spécifiques comme le stockage en mémoire, l'effacement de caractères ou la réinitialisation de la pile de mémoire. La classe Operator a accès à l'état interne de la calculatrice (State), ce qui lui permet d'appliquer les opérations de manière cohérente et de maintenir la synchronisation avec les données en cours de calcul.

## Classe State

La classe State assume le rôle central de la gestion des états internes, intervenant à chaque sollicitation d'opération. Elle est conçue pour maintenir une représentation précise de l'état courant de la calculatrice. À l'intérieur de cette classe, on observe la présence de constantes telles que "ERROR" pour indiquer un état d'erreur, "DEFAULTVAL" pour la valeur par défaut, "NEGATE" pour le signe négatif, et "DOT" pour le point décimal. Ces constantes contribuent à une gestion uniforme des états et des opérations au sein de la classe.

La pile est l'élément fondamental pour stocker temporairement les valeurs pendant les calculs. Les opérations de la pile, telles que l'ajout et la récupération de valeurs, sont orchestrées par des méthodes telles que "pushCurrent" et "getStackValue".

L'utilisation de la mémoire offre la possibilité de stocker une valeur pour une utilisation ultérieure, illustrée par les fonctions MS (Memory Store) et MR (Memory Recall) de la JCalculator. Enfin, la classe est également équipée de méthodes pour gérer le mode d'erreur, l'entrée utilisateur, la modification de la valeur courante, et d'autres opérations spécifiques. Ces méthodes incluent "setError", "setUserInput", "appendToCurrent", et "negateCurrent".

## Classe JCalculator

La classe JCalculator constitue l'interface graphique d'une calculatrice fonctionnant en notation polonaise inverse. Héritant de la classe JFrame, elle offre une mise en page structurée comprenant un champ de texte pour afficher la valeur courante et une liste pour représenter la pile de calcul. Les composants graphiques sont soigneusement positionnés grâce à la gestion des contraintes GridBagConstraints. La méthode « update » assure une mise à jour cohérente de l'interface à chaque opération, en utilisant les données provenant de la classe State. Les boutons numériques et opérateurs sont liés à des instances spécifiques d'Operator, mettant en œuvre des fonctionnalités telles que l'ajout de chiffres à la valeur courante, les opérations arithmétiques, et le changement de signe. En somme, la classe JCalculator agit comme l'interface visuelle intuitive et interactive qui permet à l'utilisateur d'interagir avec la calculatrice RPN, orchestrant les opérations à travers les mécanismes élaborés dans les autres classes.

## Classe Calculator

La classe Calculator contient l’implémentation de la calculatrice en mode console. Pour l’utiliser, on peut soit entrer un nombre (décimal ou entier) pour l’ajouter sur la pile ou bien une opération (liste des opérations supportées obtenue grâce à la commande *HELP*). Pour faire le lien entre l’entrée de l’utilisateur et l’opération à exécuter, nous avons utilisé une HashMap qui permet de facilement récupérer le bon Operator grâce à un String sans utiliser un switch – case ou des if – else à la suite. Après chaque entrée de l’utilisateur (sauf *HELP*), le contenu de la pile puis la valeur courante sont affichés.

Malheureusement, une petite erreur se produit lors de l’appel des fonctions *C* ou *CE*. En effet, après avoir réinitialisé la valeur courante, elle y mette la valeur par défaut, qui est 0. Ceci est tout à fait adapter pour la version graphique de la calculatrice car l’utilisateur peut changer la valeur courante mais ce n’est pas le cas en console, ce qui a pour conséquence qu’un 0.0 apparaît sur la pile. Il faudrait modifier le comportement des Operator Clear et ClearError pour réparer cela mais nous manquons malheureusement de temps.

## Classe TestStack

Le fichier “TestStack.java” est une classe de test visant à évaluer notre implémentation de notre pile. Les tests comprennent le push, le pop, si la pile est vide et l’utilisation d’un itérateur pour parcourir les éléments. Le code utilise des méthodes pour comparer le résultat attendu avec celui obtenu, affichant les succès en vert et les cas d’échec en rouge dans la console.

## Test des programmes

Pour tester le bon fonctionnement de la calculatrice, en mode graphique ou en mode console, nous avons essayé chacune des fonctions possibles en lançant les programmes afin de vérifier que leur comportement est bien celui attendu.

Pour tester le fonctionnement de la pile, le fichier TestStack.java possède sa propre méthode main qui lancera différent test de la pile et de son itérateur.

## Annexes

* Diagramme de classe
* Code source