Documentation des choix et hypothèses de travail

- Auteurs : Bleuer Rémy, Leyre Arnaut

- Date : 28.11.24
- Classe : POO-A

Introduction

Ce document présente les choix de conception et les hypothèses de travail réalisés dans le cadre de la modélisation UML et de l'implémentation Java du laboratoire `Calculatrice`. L'objectif est de fournir une base de code extensible et robuste pour gérer les opérations d'une calculatrice fonctionnant en notation polonaise inverse, tout en respectant les contraintes de la donnée.

Choix d'implémentation

1. Stack :

- Au-delà des classes requises par la consigne, un des premiers choix effectués était de déclarer la classe 'Node' comme une classe interne privée à la classe 'Stack'. Cette encapsulation permet à 'Stack' d'itérer sur ses propres données sans perdre le haut de la pile et factorise le code de ses fonctions.
- La classe 'Stack' bénéficie d'un deuxième classe interne 'StackIterator' public cette fois pour permettre à d'autres classes de parcourir la stack par itération si besoin. 'StackIterator' implémente l'interface 'java.util.Iterator' pour respecter la convention Java et permettre une utilisation plus souple des StackIterators.

2. State (gestion des données) :

La classe 'State' sert à gérer les données, le stack, la mémoire courante et le slot de mémoire. Les attributs sont privés et ne peuvent être interagis que par le biais de metteur secteur pour éviter les manipulations bizarres.L'ajout le plus notable à 'State' est la variable isOperationPerformed qui est cruciale pour différencier un 0 en mémoire courante d'un zéro de l'état de défaut.

3. Factorisation des opérations :

Afin de factoriser un maximum le code et de respecter le design pattern Modèle-Vue-Contrôleur, il a été décidé de répartir les différentes opérations dans diverses sous-classes qui héritent de 'Operator'. Ces sous-classes sont réparties dans trois groupes en fonction de la classe dont elle hérite ou non. On a donc:

- Les 'UnaryOperator' pour les sous-classes requièrent comme paramètre un double.

- Les 'BinaryOperator' pour les sous-classes requièrent comme paramètre deux doubles.
- Ainsi que quelques sous-classes qui ne rentrent dans aucune des deux classes précédentes.

4. Calculator :

Dans la calculatrice du terminal, il y a quelques changements dans le comportement. Notamment, toutes les opérations d'édition de valeur ne sont pas directement accessibles, car les valeurs sont entrées sous la forme d'une ligne de commande. Ainsi, il n'y a pas de commande pour 'Backspace', 'Enter' et 'ClearEntry' qui sont par défaut faisables dans le terminal. La seule vraie différence, c'est donc que la valeur doit attendre la prochaine opération pour entrer dans la stack. Exemple : pour faire 2 + 0 par exemple, il faut entrer "0" dans le terminal pour stacker 2, mais on peut faire "enter" sur l'application pour stacker 2 et remplacer le currentValue par 0. Ces deux cas achèvent le même résultat avec le même nombre d'input et ne justifient donc pas l'ajout de 'Enter' dans les opérations disponibles à 'Calculator'.

Modélisation UML

1. Cardinalités :

- Il y a une cardinalité 0..1 à 0..1 de `Node<T>` vers `Node<T>` pour représenter la variable next, top n'a pas de nodes précédentes et Bottom a null pour next.
- Il y a une cardinalité * à 1 de `StackIterator<T>` vers `Node<T>` car tout stackItérator doit tracker une node mais une node n'est pas forcément traquée.

2. CI :

- Une division ne peux pas diviser par 0.
- Une inversion ne peut pas inverser 0.
- Une racine carrée pour x négatif n'a pas de solution dans R.
- Une node ne doit pas être next d'une node en dessous d'elle dans la pile
- Un string digit passée à 'NumberOperator' doit être parsable en double.
- Une commande dans la Hash map de 'Calculator' ne peut pas être liée à deux opérations.

Tests unitaires :

1. Addition :

- basic addition : 4.0 [OK]
- default state addition : Error [OK]
- default state 2.0 addition : Error [OK]

2. Subtraction :

- basic subtraction : -1.0 [OK]
- default state subtraction : Error [OK]
- default stacked 2.0 subtraction : Error [OK]

3. Multiplication :

- basic Multiplication : 2.0 [OK]
- Multiplication by float : 0.375 [OK]
- Multiplication reasonable large float handling(no overflow) : 0.12522222222222223 [OK]

4. Division :

- basic Division : 1.0 [OK]
- Division by float : 24.0 [OK]
- Division by 0 : Error [OK]

5. SquareRoot :

- basic SquareRoot : 2.0 [OK]
- SquareRoot of négative : [OK]

6. Reciprocal :

- basic Reciprocal : 0.25 [OK]
- Reciprocal of 0 : Error [OK]
- Reciprocal of float : 4.0 [OK]

7. Square :

- basic Square : 4.0 [OK]
- Square of negative : 4.0 [OK]

8. ChangeSign :

- basic ChangeSign : 2.0 [OK]

9. misc :

- memory swaperou : 5.0 [OK]

10. pop :

- pop empty : Stack is already empty. [OK]

11. pushAndArray :

- get array returned : {2.0, 2.0} [OK]
- get array in place : {2.0, 2.0} [OK]

12. StackIterator :

- iterrate from start : 3.0 [OK]
- last has next: false [OK]

13. StackToString:

- basic string : 3.0 2.0 1.0 [OK]

Conclusion :

Cette implémentation répond aux demandes de la donnée de manière modulaire et maintenable. Le choix d’utiliser des interfaces pour les opérations permet une extension facile des opérations. Les tests unitaires couvrent l'ensemble des cas d'usage, et des CIs. L’architecture proposée offre une grande flexibilité.