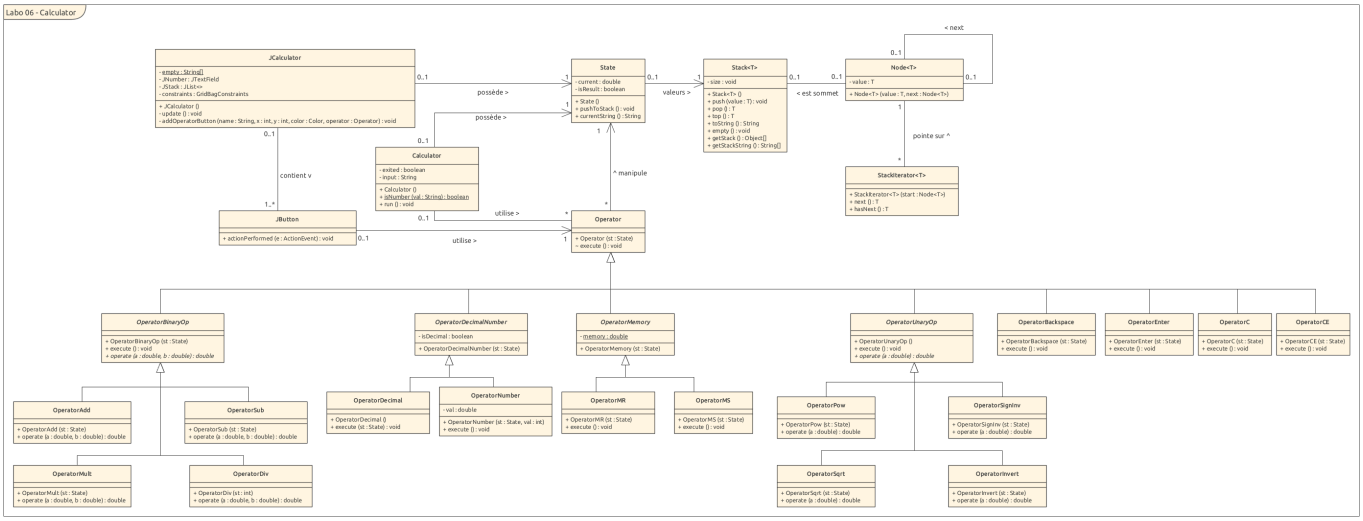
POO Rapport Lab06

Auteurs : Sara Camassa & David Berger

Introduction

Ce laboratoire a pour but d'implémenter les opérations d'une calculatrice. Un document nous avait déjà été fourni avec une interface graphique fonctionnelle. Il était demandé de faire fonctionner cette calculatrice graphiquement et par terminal.

Modélisation



Choix d'implémentation

Stack et State

Une `State` définit l'était actuel de notre calculatrice. Chaque `State` possède une pile qui stocke toutes les valeurs précédentes ainsi qu'une variable `current` gère le nombre affiché dans l'interface. Elle possède également un booléen `isResult` pour gérer le stockage dans la pile.

Nous avons créé notre propre stack générique, `Stack<T>`, qui possède toutes les opérations classiques d'une pile. En haut de cette pile se trouve un objet `Node<T>`, qui stocke la dernière valeur rentrée.

Chaque `Node<T>` possède une valeur et est lié à son précédent au moyen d'un attribut `Node<T> next`.

Un objet `StackIterator<T>` nous donne un itérateur sur notre `Stack<T>`. Cet itérateur nous permet de parcourir notre pile entièrement, grâce aux méthodes `next()` et `hasNext()`, pour les méthodes d'affichage et pour transformer la pile en un tableau d'`Object`.

Operator

Il nous a été demandé d'implémenter toutes les opérations d'une calculatrice simple. Pour ce faire, nous avons utilisé `Operator` comme "super classe" abstraite.

Pour éviter la duplication de code, nous avons regroupé les opérations qui se ressemblaient en les faisant sous-classes d'une autre classe abstraite, elle-même sous-classe de `Operator`

- `OperatorBinaryOp` : regroupe toutes les opérations binaires, c'est à dire l'addition, la soustraction, la multiplication et la division.
- `OperatorUnaryOp` : regroupe toutes les opérations unaires, c'est à dire la mise au carré, la racine carrée, l'inversement et le changement de signe.
- `OperatorDecimalNumber` : regroupe la gestion des nombres entiers ou à virgules.
- `OperatorMemory` : regroupe la gestion des opérations de mémoires, MemoryStore (MS) et MemoryRecall (MR).
  Les opération `OperatorBackspace`, `OperatorEnter`, `OperatorC` et `OperatorCE` sont directement sous-classes de `Operator`.

Chaque sous-classe directement reliée à `Operator` redéfinit la méthode `execute()`, et les sous-classes de calcul redéfinissent la méthode `operate()`.

Calculator

`Calculator` nous permet d'utiliser toutes les options implémentées dans notre calculatrice directement dans la console. C'est également cette classe qui va nous permettre de faire les tests.

Bien évidemment, toutes les opérations sur l'interface graphique ne sont pas disponibles dans `Calculator`, comme notamment :

- `OperatorBackspace` : l'effacement de la valeur courante peut s'effectuer directement au clavier.
- `OperatorCE` : l'effacement de la valeur courante peut s'effectuer directement au clavier.
- `OperatorDecimal` : on peut entrer un chiffre à virgule directement au clavier.
- `OperatorMS` et `OperatorMR` : la valeur courante est modifiée manuellement et n'est pas affichée.

On utilise une `HashMap<String, Operator>` pour lister et appeler toutes les opérations possibles, une `State` pour sauvegarder l'état de la stack, un String pour stocker l'input

utilisateur et un booléen pour vérifier si le programme doit être arrêté.

Tests effectués

Les tests sont effectués à la main. Nous utilisions l'extension `Calculator` sur le terminal, sauf pour les tests propres à la calculatrice `JCalculator`.

Opérateurs binaires

Nom	Classe	Entrée	Sortie
Addition	OperatorAdd	3 1.2 +	4.2
Addition avec 1 nombre négatif	OperatorAdd	-3 1.2 +	-1.8
Addition avec 2 nombres négatifs	OperatorAdd	-3 -1.2 +	-4.2
Soustraction	OperatorSub	5 1 -	4.0
Soustraction avec 1 nombre négatif	OperatorSub	-5 1 -	-6.0
Soustraction avec 2 nombres négatifs	OperatorSub	-5 -1 -	-4.0
Multiplication	OperatorMult	2 3 *	6.0
Multiplication avec 1 nombre négatif	OperatorMult	-2 3 *	-6.0
Multiplication avec 2 nombres négatifs	OperatorMult	-2 -3 *	6.0
Division	OperatorDiv	9 2 /	4.5
Division avec 1 nombre négatif	OperatorDiv	-9 2 /	-4.5
Division avec 2 nombres négatifs	OperatorDiv	-9 -2 /	4.5
Division par 0	OperatorDiv	9 0 /	Infinity
Diviser un 0	OperatorDiv	0 2 /	0.0
0 divisé par 0	OperatorDiv	0 0 /	NaN
Opération binaire sans chiffre	OperatorAdd	+	<empty>
Opération binaire avec un seul chiffre (JCalculator)	OperatorAdd	2 +	4.0
Opération binaire avec un seul chiffre (extension)	OperatorAdd	2 +	2.0

Opérateurs unaires

Nom	Classe	Entrée	Sortie
Chiffre au carré	OperatorPow	3 x^2	9.0
Chiffre à virgule au carré	OperatorPow	3.5 x^2	12.25

Nom	Classe	Entrée	Sortie
Chiffre négatif au carré	OperatorPow	-3 x^2	9.0
0 au carré	OperatorPow	0 x^2	0.0
Racine carrée	OperatorSqrt	9 sqrt	3.0
Racine carré d'un chiffre à virgule	OperatorSqrt	12.25 sqrt	3.5
Racine carré d'un chiffre négatif	OperatorSqrt	-9 sqrt	NaN
Inversion	OperatorInvert	5 1/x	0.2
Inversion d'un chiffre à virgule	OperatorInvert	0.2 1/x	5.0
Inversion d'un chiffre négatif	OperatorInvert	-5 1/x	-0.2
Inversion de 0	OperatorInvert	0 1/x	Infinity
Inversion de signe	OperatorSignInv	3 +/-	-3.0
Inversion de signe d'un chiffre à virgule	OperatorSignInv	2.3 +/-	-2.3
Inversion de signe d'un chiffre négatif	OperatorSignInv	-3 +/-	3.0
Inversion de signe de 0	OperatorSignInv	0 +/-	-0.0

Autre

Nom	Classe	Entrée	Sortie
Entrée d'un chiffre	OperatorNumber	3	3.0
Entée d'un chiffre à virgule (extension)	OperatorNumber	3.5	3.5
Entrée d'un chiffre à virgule (JCalculator)	OperatorNumber + OperatorDecimal	3 . 5 <enter>	3.5
Entrée d'un mauvais caractère (extension)	OperatorNumber	a	Invalid
Entrée vide (extension)	OperatorEnter	<enter>	Invalid
Entrée vide (JCalculator)	OperatorEnter	<enter>	0.0
Backspace (JCalculator)	OperatorBackspace	5 . 2 <=	5
MemoryStore + MemoryRecall (JCalculator)	OperatorMS + OperatorMR	5 <MS> 9 <MR>	5
Clear	OperatorC	5 <enter> 4 <C>	curr: 0 stack: 5.0
ClearError (JCalculator)	OperatorCE	5 <enter> 4 <CE>	curr: 0 stack:

Nom	Classe	Entrée	Sortie
			<empty>
Exit (extension)	-	exit	-