# Calculator

POO1 – Laboratoire 08

Bouyiatiotis - Jaquet

# Table des matières

# Objectif

Réaliser, en utilisant JCalculator.java (fichier fournis), une calculatrice qui permet de faire les opérations en utilisant la notation polonaise inverse.

Un diagramme de classe partiel est fourni. Il faudra le compléter en spécifiant ce que nous allons utiliser pour State et Operator.

Il nous faudra compléter JCalculator.java ainsi que créer les classes State, stockant l'état de la machine non graphique de manière à pouvoir le réutilisé pour le mode console, et Operator permettant de réaliser les opérations de la calculatrice.

Il faudra aussi rendre possible l'utilisation de State et Operator pour le mode console.

## Mise en œuvre :

- Définir une hiérarchie de classes avec pour racine une classe Operator en factorisant au maximum le code. Cette classe devra posséder une méthode public void execute(), qui est automatiquement invoquée par l'interface à l'appui d'un bouton (initialisation effectuée dans la méthode JCalculator.addOperatorButton).
- Représenter l'état interne (non graphique) de la calculatrice (valeur courante, pile, erreur, etc).
- Remplacer les valeurs null des appels à addOperateurButton par des instances ad hoc de la classe Operator.
- Définir la méthode JCalculator.update(), invoquée après chaque opération, depuis addOperateurButton (qui ne devra pas être modifiée) pour réactualiser l'interface (valeur courante, état de la pile). Le rapport comprendra un diagramme des classes complet ainsi qu'une description des choix de

## Exemple d'utilisation :

Soit à évaluer l'expression (3.5 + 4) / (2.52 + 1). Avec cette calculatrice, les opérations seront:

- Appui des touches 3, . et 5, et Ent. La valeur 3.5 est placée sur la pile.
- Appui de la touche 4.
- Appui de la touche +: évaluation du résultat, 7.5.
- Appui des touches 2 (la valeur précédemment calculée, 7.5, est placée sur la pile), . et 5.
- Appui de la touche x^2: évaluation du résultat intermédiaire 6.25.
- Appui de la touche 1 (la valeur 6.25 est placée sur la pile).
- Appui de la touche +: évaluation du résultat intermédiaire 7.25.
- Appui de la touche /: évaluation du résultat final 1.0344827586206897.

# Diagramme des classes

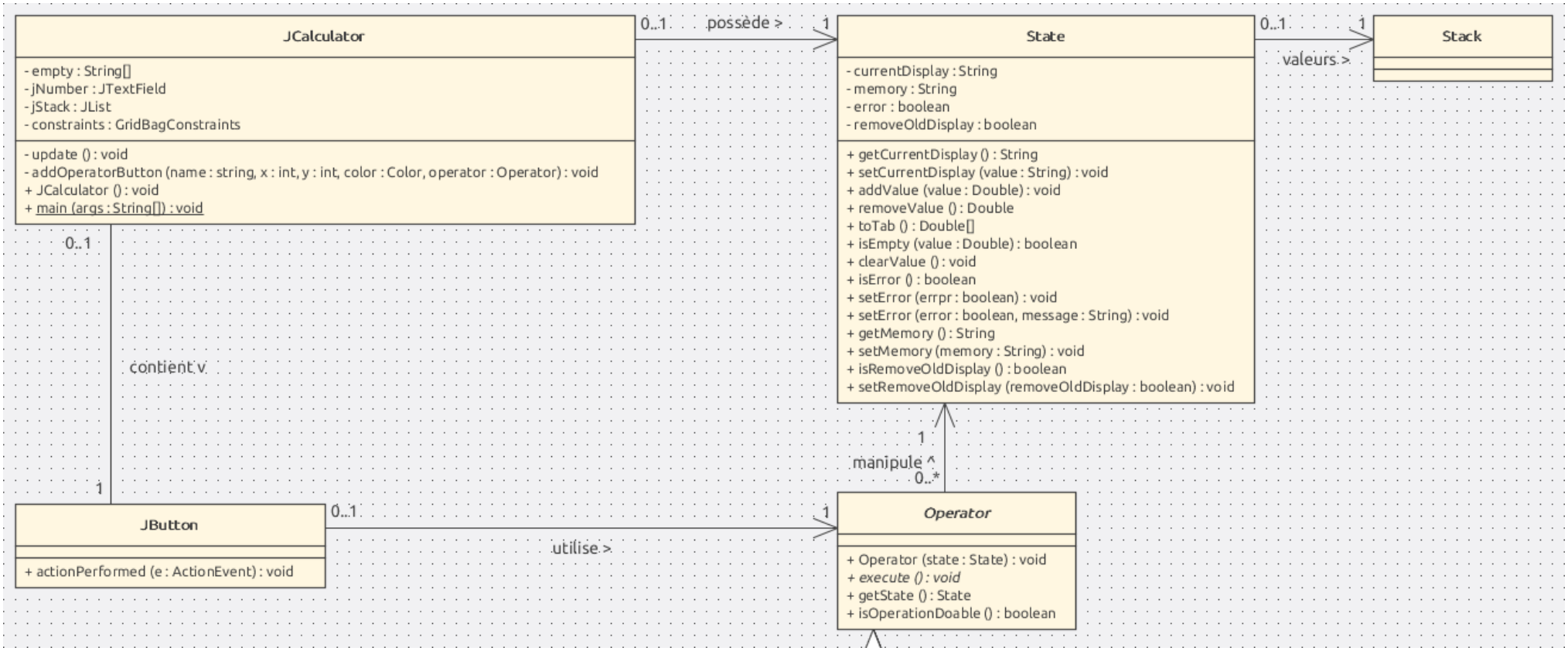



*Figure 1 UML des 5 classes principale gérant la calculatrice*

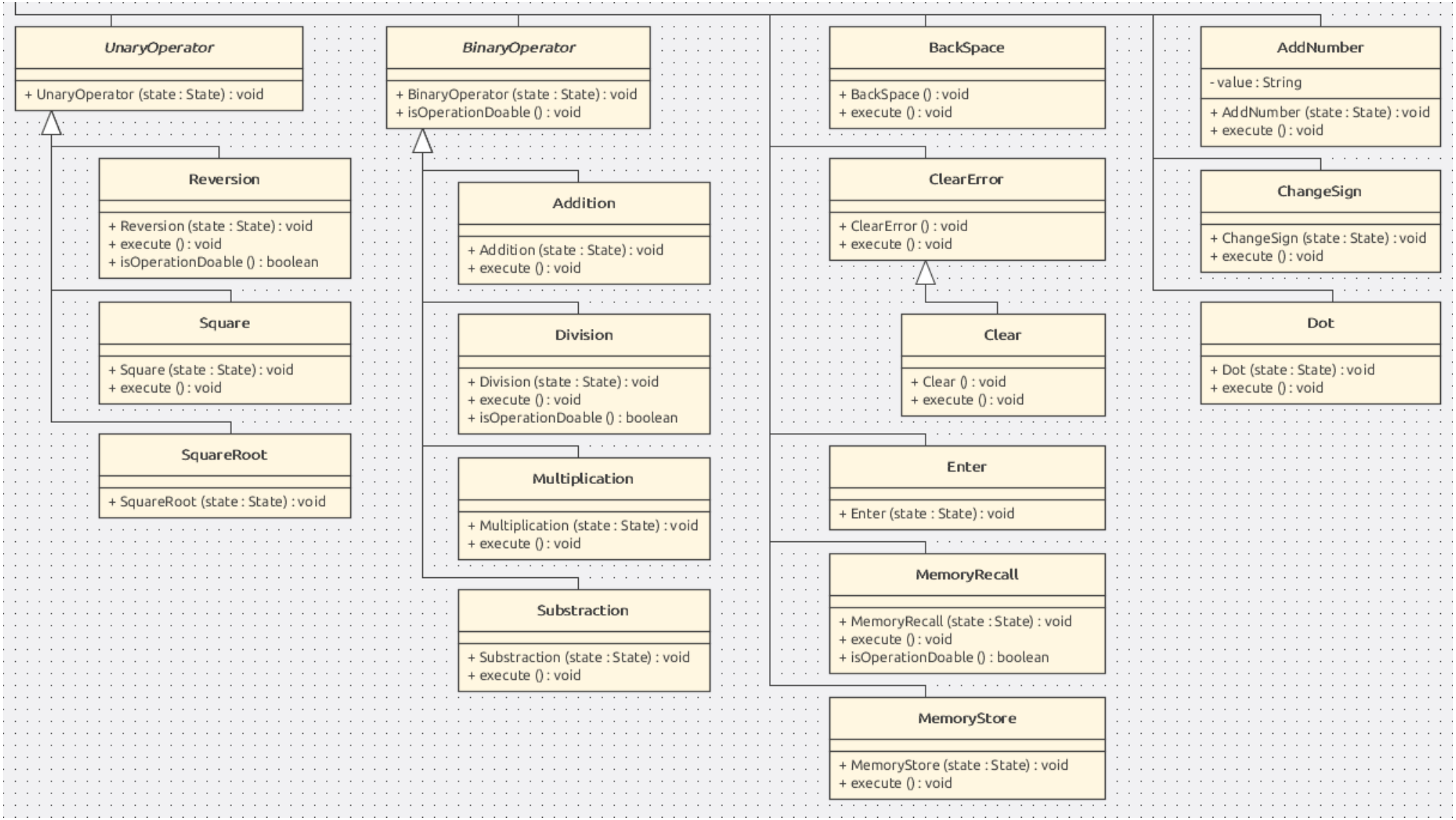



*Figure 2 UML de toutes les sous-classes de la classe Operator*

Les classes UnaryOperator, BinaryOperator, BackSpace et AddNumber héritent toutes de la classes Operator présente dans la figure 1

# Description des classes

Comme indiqué par le professeur, les classes JCalculator, JButton et Stack ne vont pas être détaillé autant que les classes State et Operator.

Sur la première image (Figure 1) se trouve l'UML complété selon celui qui nous a été fourni. Sur la seconde image (Figure 2) se trouve toutes les sous-classes qui serviront aux différentes opérations pour la calculatrice.

## JCalculator :

La classe permet de gérer l'affichage sur l'interface graphique via les éléments ci-dessous. Les éléments sont écrit de la manière suivante : `<Type> <nom>`

- JTextField jNumber : Qui est l'affichage sur la ligne d'entrée.
- JList jStack : Qui affiche les éléments dans la stack.
- State state : Permet de garder en mémoire ce qu'il y a à afficher. Les attribues de JCalculator utiliseront les valeurs contenue dans state pour l'affichage.

Elle construit aussi l'affichage des boutons de commande sur la calculatrice et d'assigner les opérations grâce à la méthode `addOperatorButton()`.

Chaque bouton aura son opération assignée qui sera une sous-classe qui redéfinira la méthode execute() de la super classe Operator. Ici, toutes les touches sont une opération, même les nombres.

La classe JCalculator permet également de mettre à jour l'affichage après une opération avec la méthode update().

## State :

State est la classe qui permet de garder en mémoire les informations entrées par l'utilisateur. Pour cela, la classe dispose des attributs suivants :

- Stack<Double> values : Stocke les valeurs entrées et permet de réaliser les opérations arithmétiques. On ajoute une valeur à l'intérieur après un « enter » ou après avoir entré un nombre après une opérations arithmétique.
- String currentDisplay : Garde en mémoire les valeurs entrées sur la calculette mais qui ne sont pas encore stocké dans la pile.
- String memory : Permet de garder une valeur en mémoire pour la réutilisé plus tard sans la placer dans la pile.
- boolean error : Permet de signaler s'il y a erreur. En cas d'erreur, celle-ci sera indiquée dans l'attribut currentDisplay.
- boolean removeOldDisplay : Permet de détecter si la valeur de currentDisplay est obtenue suite à une opération ou bien si elle est récupérée de memory. Cela permet, entre autre, de placer la valeur contenue dans currentDisplay dans la stack directement après avoir appuyé sur d'autre valeurs plutôt que de la changé ou sa suppression via le backSpace.

En plus des getter, des setter et des méthodes, nous avons créé la méthode « toTab » qui transforme la stack courante en tableau de Double pour l'affichage de la stack sur l'interface graphique via jStack ainsi que des méthodes publiques permettant d'ajouter et supprimer des éléments de la pile (respectivement addValue et removeValue).

La classe Operator manipulera les informations à l'intérieurs de State alors que JCalculator ne fera que les afficher. State représente l'état mémoire de la machine.

# Operator :

La classe Operator gère toutes les opérations sur les boutons via une méthode execute() qui est abstract car chaque opération sera découpé en sous-classe de Operator et donc demandera une redéfinition de la méthode execute.

Operator est coupée en 3 catégories principales :

- Number : Ce sont les opérations qui manipulent uniquement l'entrée du nombre. Comme les touches 1, 2, 3 etc… ou encore le « . » (représentant la virgule) ou bien le changement de signe.
- Arithmetic : Représente toutes les opérations arithmétiques de la calculette. Addition, soustraction, division etc... Toutes ses opérations manipulent directement la partie dit mémoire (la pile).
- Memory : Ce sera toutes les opérations qui enregistrent et/ou suppriment les éléments de la pile. Comme le « Enter », « BackSpace » etc…

## Fonctionnement :

Ici nous décrirons le fonctionnement des différents fichiers `.java` se trouvant dans le package Operator.

Pour la catégorie number il y en a 3 sous-classes :

- AddNumber : Sous-classe ayant un attribut « String value » qui enregistre le nombre associé ainsi cela nous permet de savoir quelle valeur est passée suivant le bouton appuyé. La classe s'adapte selon l'état de currentDisplay ou removeOldDisplay. Si 0 se trouve de base dans le currentDisplay alors il est remplacé par la nouvelle valeur.
- Dot : Rajoute un point currentDisplay si un point s'y trouve déjà alors aucune action n'est faite.
- ChangeSign : Change le signe devant currentDisplay, l'alterne entre -/+. Il n'y a pas de signe possible pour 0.

Pour les sous-classes Arithmetic, il y a les opérateurs binaire (addition, soustraction etc…) et les opérateurs unaire (carré, racine etc…). Il est important de noter que les opérateurs unaires travaillent avec une seule valeur et les opérateurs binaires travaillent avec deux valeurs. Ces deux types d'opérateurs passent à « true » le removeOldDisplay :

- Opération unaire : Se réalise directement dans le currentDisplay sans interagir avec la pile.
- Opération binaire : Calcule la valeur, selon l'opérateur arithmétique sélectionné, en sélectionnant ce qu'il y a dans le currentDisplay et la valeur en tête de pile.

Les sous-classes Memory effectuent des opérations sur la mémoire ou sur l'affichage :

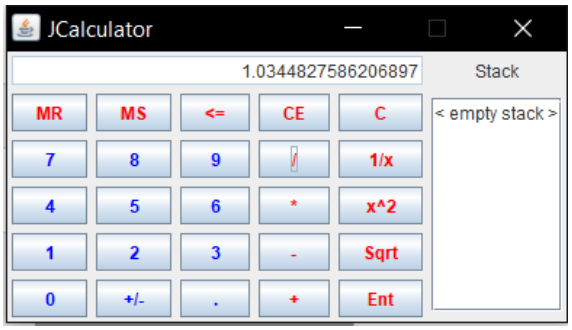
- Enter : Enregistre la valeur dans le currentDisplay dans la pile.
- BackSpace : Enlève le dernier élément entré par l'utilisateur.
- Clear : Remet à zéro l'interface, supprime la pile et supprime les erreurs.
- ClearError : Remet à zéro le currentDisplay et supprime les erreurs.
- MemoryStore : Stocke la valeur de currentDisplay en mémoire pour la réutiliser plus tard.
- MemoryRecall : Récupère la valeur stockée en mémoire.

# Test

Dans cette partie nous testerons le bon fonctionnement du programme.

## Test graphique :

Résultat des suites d'opérations fournis dans la donnée :




Tests supplémentaires :

| Situation | Attendue | OK/Erreur |
|---|---|---|
| Après avoir lancé le programme, entrée d'un zéro | Rien ne se passe | OK |
| Après avoir lancé le programme, entrée d'un nombre | Remplace le zéro par la valeur | OK |
| Appuie sur le bouton « . » | Place un point après le chiffre (que ce soit aux départs du programme ou pendant) | OK |
| Appuie sur le bouton « . » alors qu'une virgule est déjà présente | Ne fait rien | OK |
| Appuis sur le bouton +/- | Alterne entre + si – et – si +, le 0 n'a pas de signe, ne fait rien en cas d'erreur | OK |
| Appuyé sur les nombres | S'ajoutent 1 à 1 derrière le précédent | OK |
| Appuie sur le Ent | Ajoute la valeur de l'affichage dans la stack | OK |
| Appuie sur un opérateur Unaire | Résous l'opération avec la valeur d'affichage | OK |
| Appuie sur un opérateur Binaire (avec valeur dans la pile) | Résout l'opération avec la valeur d'affichage et la première valeur dans la stack et place le résultat dans l'affichage tout en supprimant celle contenue dans la stack | OK |
| Appuie sur un opérateur Binaire sans valeur dans la pile | Dans l'affichage un message d'erreur s'affiche | OK |
| Inverse de 0 | Retourne un message d'erreur disant qu'il n'est pas possible de résoudre l'opération | OK |
| Racine d'un nombre négatif | Retourne un message d'erreur disant impossible de résoudre l'opération. | OK |
| Appuie sur le bouton MS puis remise à 0 pour appuyer sur MR | Sauvegarde la valeur en mémoire pour la charger après, affichage de la valeur enregistré | OK |
| Appuie sur BackSpace | Supprime la dernière entrée de l'utilisateur, mise à 0 si suppression de la dernière entrée | OK |
| Appuie sur le bouton CE | Supprime le message d'erreur et permet d'entrée de nouvelle valeur | OK |
| Appuie sur le bouton C | Remise à 0 de la calculatrice (Suppression de la stack et des messages d'erreur) | OK |

# Extension

Après avoir réalisé la calculatrice avec l'interface graphique, il nous a été demandé de rendre possible de faire la même chose, mais avec un affichage console.

## Calculator

Comme pour la classe JCalculator, celle-ci possède un attribut state permettant de garder en mémoire l'état dit mémoire de la machine.

Cependant, comme il n'y a pas t'interface graphique il faut détecter l'opération, que ce soit ajouter un nombre ou bien faire une opération. Pour cela nous utilisons un « Map<String, Operator> » :

- String : compare l'entrée de l'utilisateur et permet de rediriger sur la bonne opération.
- Operator : Réalise l'opération associée à la String.

Le fonctionnement de base ne change pas, juste quelque modification, comme le bouton enter qui n'existe pas (l'utilisateur appuie lui-même sur entrer après chaque entré) ou les opérations C ou CE qui doivent être entrée manuellement.

# Test de fonctionnement :

Dans un premier temps nous avons testé avec les mêmes entrées fournies dans la documentation pour voir si nous avions les mêmes résultats :

```
1
1.0

2
2.0 1.0

3
3.0 2.0 1.0

+
5.0 1.0

sqrt
2.23606797749979 1.0

+
3.23606797749979

exit
BUILD SUCCESSFUL (total time: 1 minute 15 seconds)
```

# Diagramme des classes

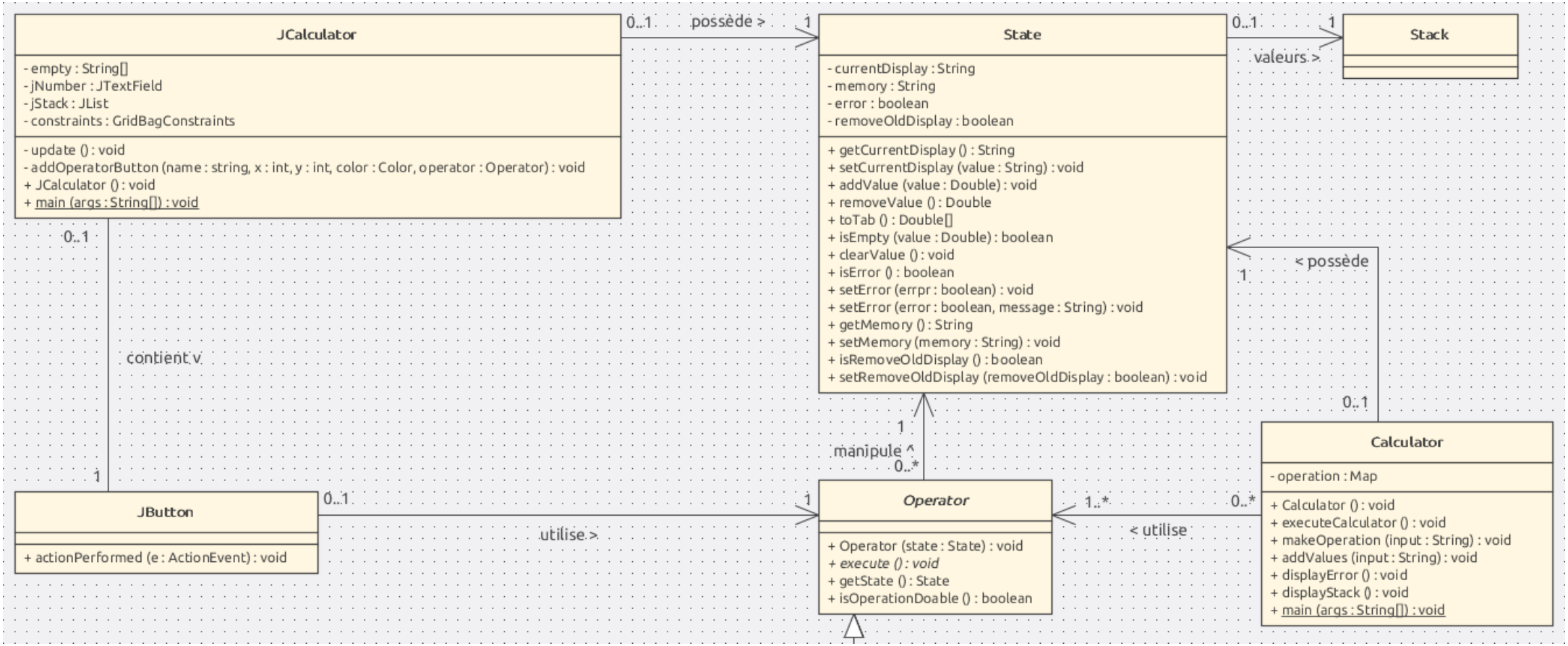



*Figure 3 Diagramme des classes avec la classe Calculator permettant l'affichage console*

## Tests complémentaires

Après chaque situation l'appuie sur enter est réalisé :

| Situation | Attendue | OK/Erreur |
|---|---|---|
| Entrée d'une valeur | La valeur s'ajoute à la pile et s'affiche | On ne peut entrer 0 au départ |
| Entrée d'un opérateur binaire avec 2 ou plus valeurs dans la pile | L'opérateur réalise l'opération avec les 2 nombres les plus récents inséré dans la pile | OK |
| Entrée d'un opérateur unaire avec 1 ou plus valeur dans la pile | Réalise l'opération unaire avec la dernière valeur entrée | OK |
| Entrée d'un opérateur unaire alors qu'il n'y a aucune valeur | Réalise l'opération avec 0 | OK |
| Entrée d'un opérateur binaire alors qu'il n'y a aucune ou 1 valeur contenue dans la pile | Message d'erreur indiquant l'erreur et demandant d'entrée CE ou C | OK |
| Division par 0 (avec l'opération ou le reverse) | Message d'erreur indiquant l'impossibilité de l'opération et d'entrée CE ou C | OK |
| Racine d'un nombre négatif | Message d'erreur indiquant l'impossibilité de l'opération et d'entrée CE ou C | OK |
| Entrée CE | Retourne à l'état ou on entre les valeurs | OK |
| Entrée C | Retourne à l'état ou on entre les valeurs et supprime ce qu'il y a dans la pile | OK |

## Remarque

Si le nombre entré ou l'opération réalisée dépasse la valeur max d'un double, la valeur affichée devient alors « Infinity ». Il peut encore être manipulé par l'utilisateur et ne provoque pas d'erreur.