

# Programmation et conception orientées objet

*Projet de réalisation*

-

## Calculatrice à notation polonaise inverse



---

**LO21**

Sébastien FERRY - Yoann VANDECNOCKE

## Introduction

Dans le cadre de l'enseignement de programmation et conception orientées objet de l'UTC, nous avons été amenés à réaliser une calculatrice à notation polonaise inverse, codée en C++ sous QtCreator, avec une interface graphique construite à l'aide de QtDesigner. En plus de présenter le contexte et les objectifs de ce projet, le présent rapport a pour but d'exposer nos réflexions ainsi que nos choix de conceptions et de développement, accompagné d'un aperçu de l'interface utilisateur.

## Nature et contexte du projet

**Contexte :** Projet à but pédagogique, effectué dans le cadre de l'UV LO21 de l'UTC.

**Maître d'ouvrage :** Antoine Jouglet, en sa qualité de responsable de l'UV.

**Intitulé du projet :** Calculatrice à notation polonaise inverse.

**Résultats attendus :** Logiciel fonctionnel et intuitif, code source et documentation Doxygen, sujet complet disponible [ici](#).

**Public cible :** La notation polonaise inverse étant peu courante, notre projet s'adresse à un public d'utilisateur très spécifique et spécialisé. Cela facilite le projet ; en effet, il est toujours plus difficile de concevoir un logiciel qui soit intuitif et abordable pour un large public.

**Délais :** projet à réaliser en moins de deux mois. Rendu définitif le 15 Juin 2012.

**Budget :** budget horaire uniquement, sur une base de six heures par semaine minimum par membre de l'équipe porteuse.

**Moyens matériel :** parc informatique de l'UTC, équipé du framework orienté objet Qt, possédant un puissant IDE, QtCreator et un assistant de création d'interface graphique, QtDesigner ; ainsi que du logiciel de génération de documentation, Doxygen.

**Supports :** polycopié de cours LO21, documentation Qt, Internet.

**Équipe porteuse :**

- Sébastien Ferry, deuxième semestre de génie informatique à l'UTC
- Yoann Vandecnocke, premier semestre de génie informatique à l'UTC

## Choix de conception

Avant toute réalisation, nous avons établi un modèle conceptuel UML, disponible en annexe, permettant de définir l'architecture du programme par des relations entre différentes classes.

Le premier choix aura été celui de la gestion des types de données traitées par la calculatrice. Pour rappel, celle-ci doit pouvoir traiter tous les types de nombre (entier, rationnel, réel et complexe) ainsi que des expressions encadrées par des guillemets. Pour cela, nous avons défini une classe abstraite *Constante* dont héritent les classes *Complexe* et *Expression*. La classe *Constante* est donc virtuelle pure : on ne peut pas l'instancier.

Nous avons également créé une classe *Rationnel*, ayant pour attribut deux *double* : un numérateur et un dénominateur. Par la suite, notre classe *Complexe* aura pour attribut deux objets de type *Rationnel* : une partie réelle et une partie imaginaire.

Nous pouvons alors définir les types de nombres ainsi :

- un complexe est un objet de type *Complexe* possédant une partie réelle et une partie imaginaire
- un rationnel est un objet de type *Complexe* avec sa partie imaginaire nulle
- un réel est un objet de type *Complexe* avec sa partie imaginaire nulle, et sa partie réelle a pour numérateur le réel, et pour dénominateur 1
- un entier est défini de la même manière qu'un réel, à la différence qu'il est arrondi à la valeur entière la plus proche.

Le principal atout de ce choix de conception est qu'une fois le mode complexe activé, ses parties réelles et imaginaire peuvent être affichées dans n'importe quel mode : entier, rationnel ou réel.

Par la suite, la classe *MainWindow* gère la fenêtre de notre programme et les interactions avec l'utilisateur. Ayant choisi de donner la possibilité à l'utilisateur d'ouvrir plusieurs onglet, cette classe aura comme attribut un *QVector* contenant des objets de la classe *Calculatrice*.

La classe *Calculatrice* gère quant à elle toutes les opérations déclenchées par l'utilisateur, elle possède une liste de *Pile* : une nouvelle *Pile* est ajoutée à la liste après chaque action, ce qui nous permet d'annuler ou de rétablir les actions effectuées. La *Calculatrice* possède aussi tous les paramètres : le type de nombre, le mode d'angle, etc.

La classe *Pile* est un *QVector* dans lequel sont stockées des pointeurs sur objets de type *Constante*, et dont les fonctions ont été adaptées. On peut ainsi stocker dans nos *Pile* des complexes, des réels, ou encore des expressions.

La gestion des fenêtres secondaires d'aide, de paramétrage et d'information se fait à l'aide de classe spécifique à chaque fenêtre : dans le même ordre, les classes *Aide*, *Parametres* et *A propos*.

Afin de gérer au mieux les erreurs de notre programme, nous avons implémenté une classe *Erreurs*, dérivée de la classe standard *exception*. Ceci nous permet alors de traiter de manière spécifique les erreurs propres à notre calculatrice, en affichant des messages d'erreurs à l'écran par l'intermédiaire de *QMessageBox*.

L'état courant de notre programme est sauvegardé dans des fichiers au format ini. Ce type de fichier est très pratique d'utilisation car il permet de d'organiser de manière efficace les données grâce à une séparation en sections. Cette sauvegarde permet à l'utilisateur de retrouver le programme dans l'état où il l'a laissé en quittant, car les données et les paramètres, sont chargés dans les fichiers à chaque ouverture du programme. Les paramètres de chaque instance de calculatrice sont sauvegardés dans un unique fichier parametre.ini, tandis qu'un fichier pileN.ini contient les données de la calculatrice numéro N.

## Développement, interface graphique et utilisation

Malgré tout le soin porté et le temps dévoué à notre conception UML, nous avons parfois dû, au cours du développement, revenir sur certain choix de conception. Les causes pouvant être : une mauvaise compréhension ou interprétation d'un aspect du sujet, des difficultés techniques ou la découverte d'une possible optimisation. Nous avons donc plusieurs fois rétro-conçu notre modèle de base, même si le rendu final reste très proche de notre première conception UML.

Voici un aperçu de l'interface graphique principale du programme :

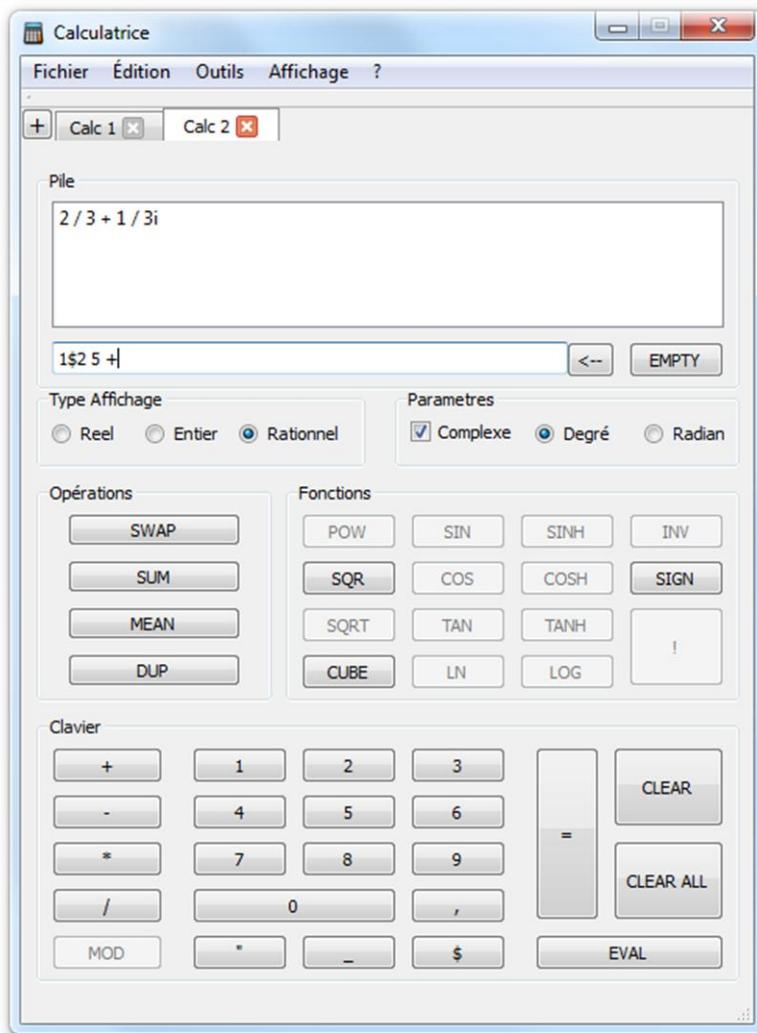


Figure 1 - Interface utilisateur

**Pile** : Cette section comprend les zones d'affichage et de saisie.

**Type Affichage** : (Les nombres entrés par l'utilisateur seront automatiquement convertis)

- Réel : les nombres sont affichés sous forme classique, avec ou sans virgule
- Entier : les nombres sont affichés sous forme entière, positive ou négative, sans virgule. Les résultats d'opération et les chiffres entrés par l'utilisateur sont arrondis
- Rationnel : les nombres sont affichés sous forme de fractions réduites, à savoir un numérateur et un dénominateur séparés par un '/'.

**Paramètre** : Permet d'activer/désactiver le mode complexe, et de choisir le type d'angle.

**Opérations** : Cette section permet d'effectuer des opérations sur la pile, réglables dans les paramètres (cf. Navigation du menu).

**Fonctions** : Cette section comprend toutes les fonctions mathématiques ne pouvant être appelées depuis le clavier physique de l'ordinateur.

**Clavier** : Cette section comprend les touches et les opérateurs manipulables depuis le clavier physique de l'utilisateur, via des touches et des raccourcis claviers. Elle peut être masquée (cf. Navigation du menu).

**Navigation du menu :**

- Fichier : comprend la commande *Quitter* (Ctrl+Q ou Alt+F4)
- Edition : comprend les commandes *Annuler* (Ctrl+Z) et *Rétablir* (Ctrl+Y)
- Outils : comprend la commande *Paramètres* (Ctrl+P) qui ouvre une fenêtre permettant de paramétriser les fonctions SUM, MEAN et SWAP et de modifier le nombre d'éléments à afficher dans la pile
- Affichage : permet de masquer ou de montrer la section **Clavier**
- ? : comprend les commandes *Aide* (F1) affichant une fenêtre d'aide et *A propos* (F5) affichant les informations sur le logiciel.

**Messages d'erreurs** : l'apparition d'une fenêtre d'avertissement indique à l'utilisateur que le programme rencontre une erreur, un message lui en indique la nature. Voici un exemple de message d'erreur.

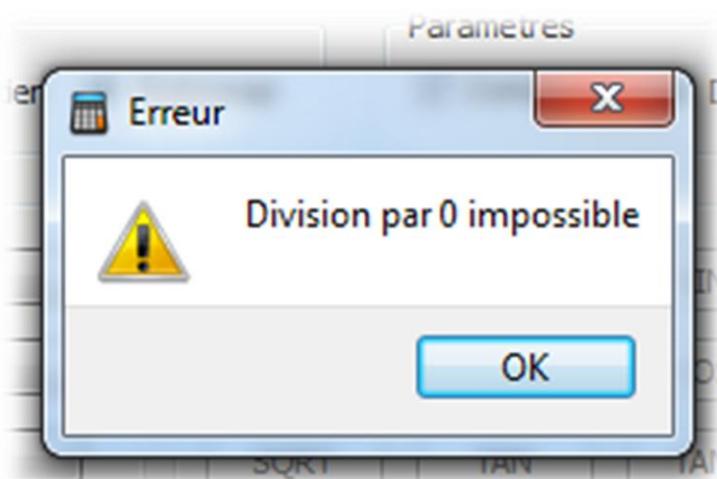
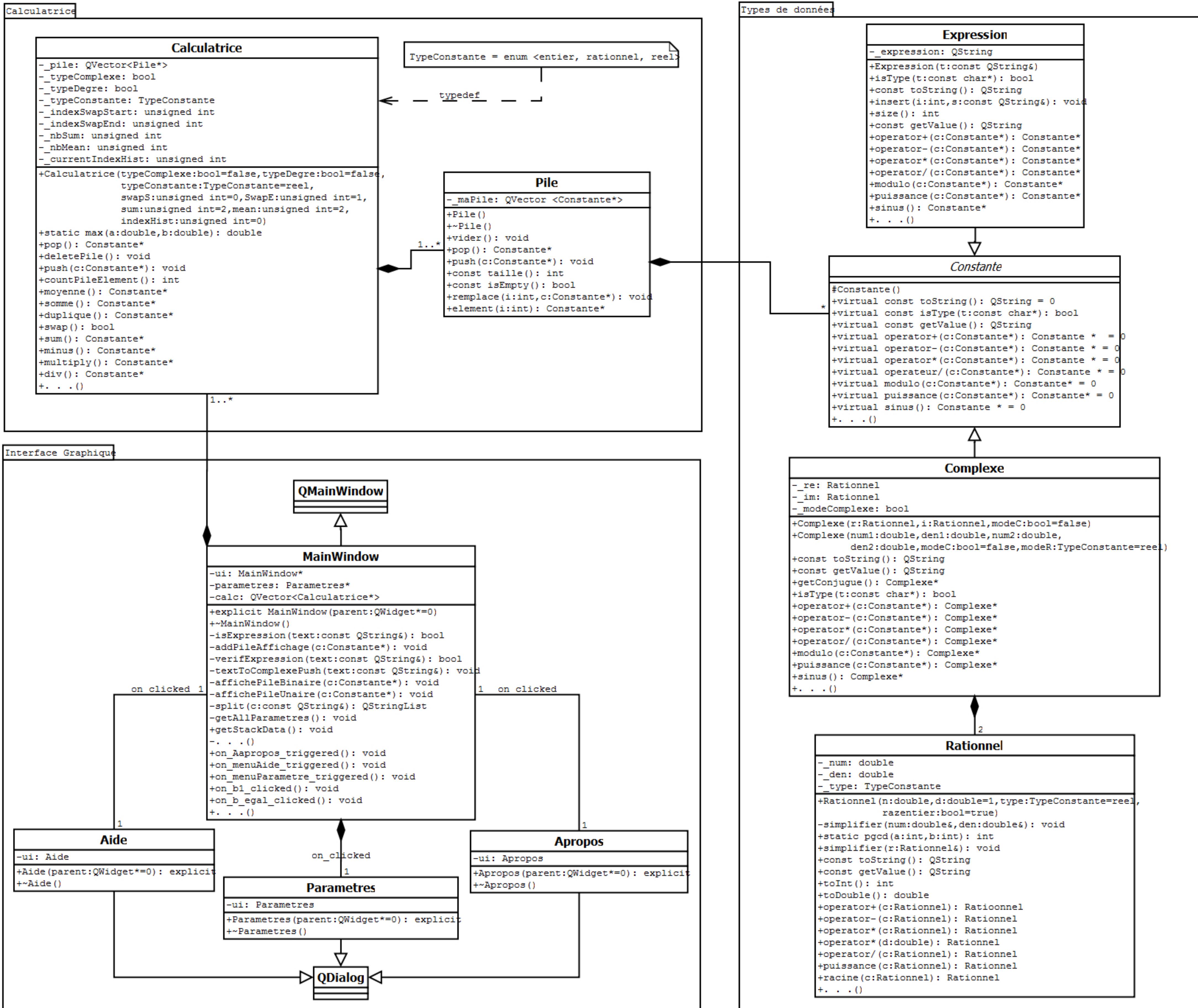


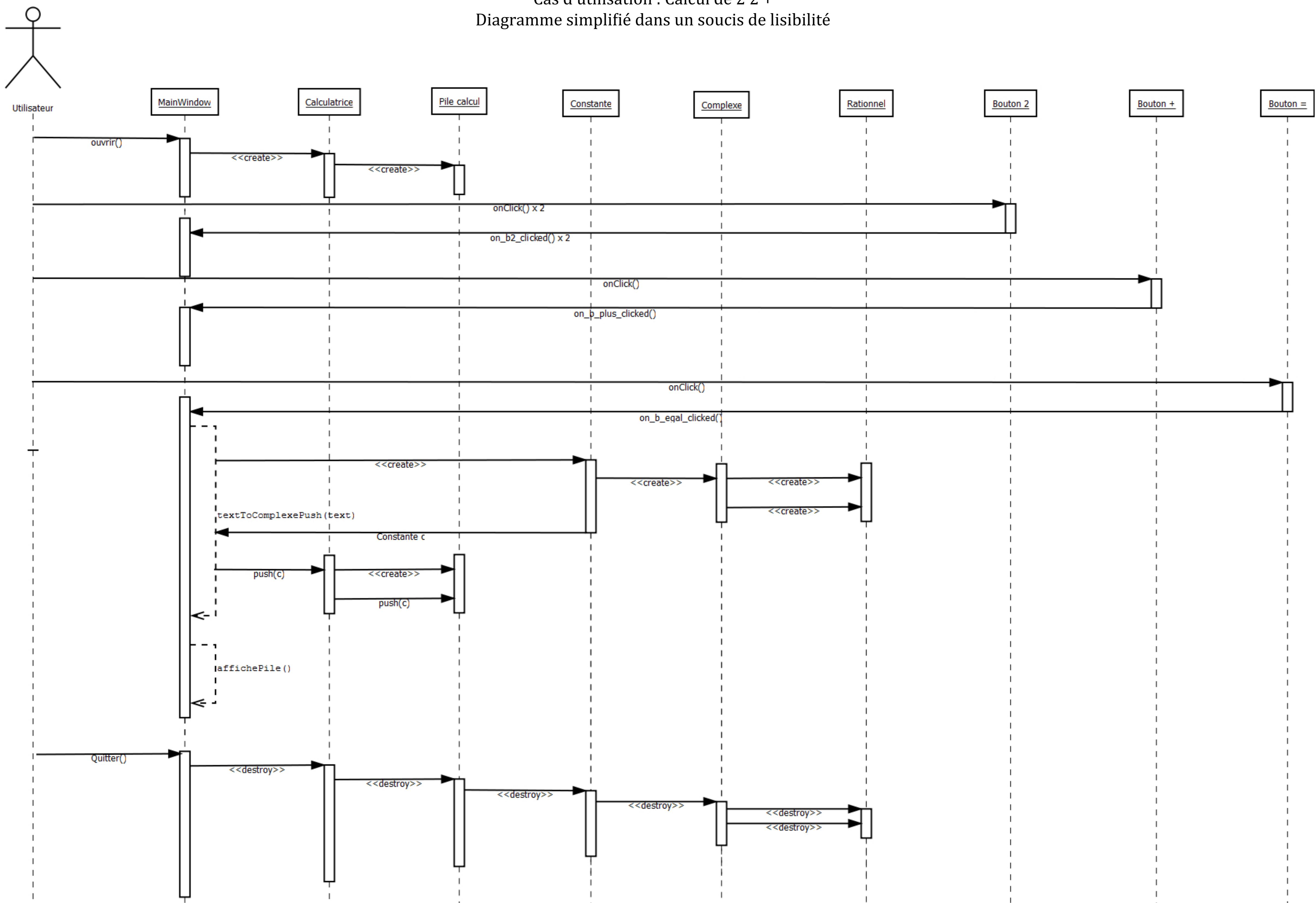
Figure 2 - Erreur de division par 0

# Annexe 1 : Modèle conceptuel UML

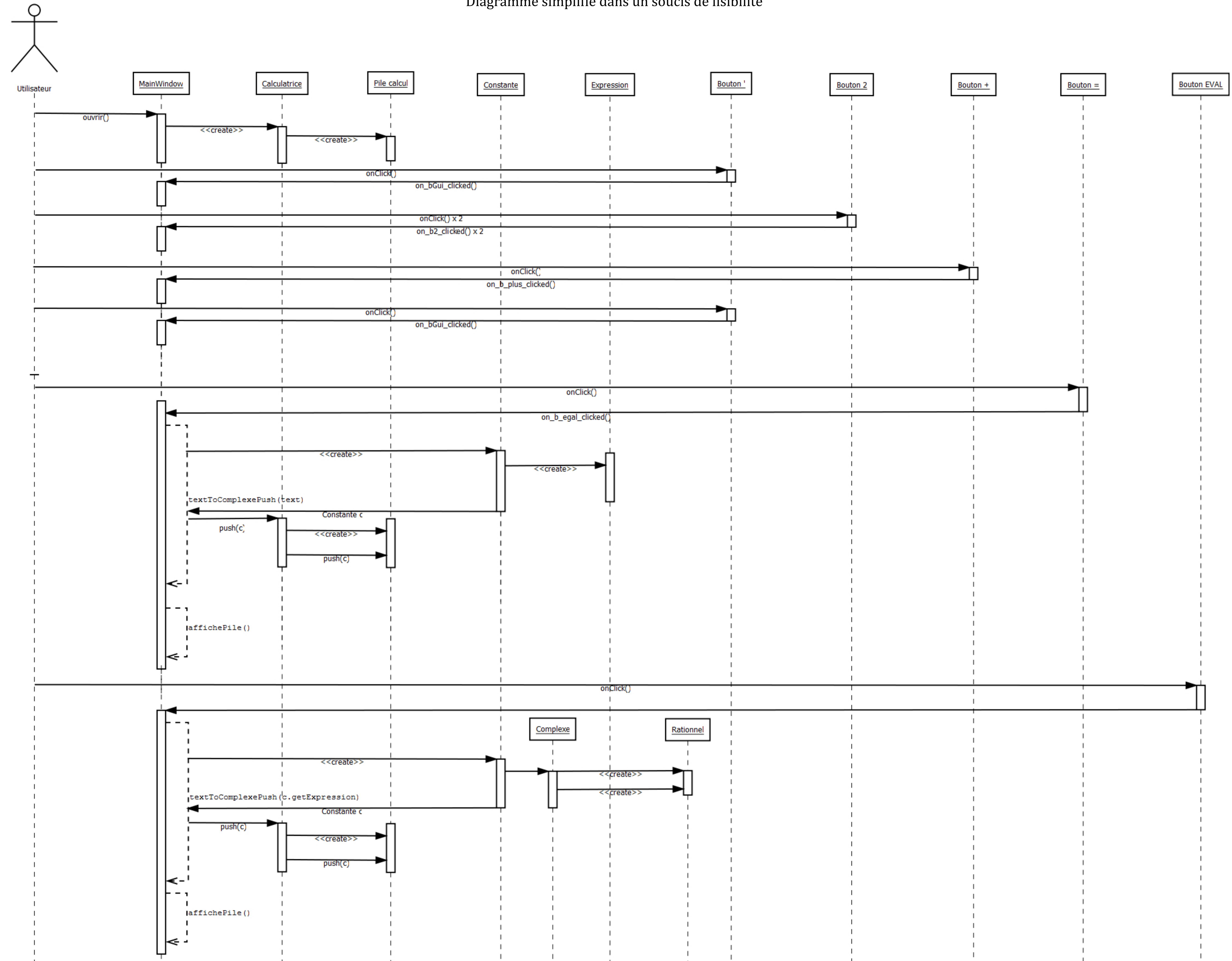


## Annexe 2 : Diagrammes de séquence

Cas d'utilisation : Calcul de  $2 \cdot 2 + 1$   
Diagramme simplifié dans un soucis de lisibilité



Cas d'utilisation : Ajout de l'expression '2 2 +' puis Evaluation  
Diagramme simplifié dans un soucis de lisibilité



## Annexe 3 : Contenu de la fenêtre d'aide

### Calculatrice à notation polonaise inverse

La notation polonaise inverse est une méthode de notation mathématique permettant de s'abstenir de l'utilisation de parenthèse.

Lors de la résolution d'une expression exprimée en notation polonaise inversée, un opérateur est remplacé par la valeur de l'opération qu'il représente en utilisant les n termes le précédents (opérateur unaire, un terme, opérateur binaire, deux termes, etc.).

Ainsi :

1+1 s'écrit 1 1 +

2 x 2 + 1 s'écrit 2 2 x 1 +

(2 +3) x 4 s'écrit 2 3 + 4 x

Pour plus d'information, vous pouvez consulter la page [wikipedia](#) dédiée.

### Utilisation de la calculatrice

#### Pile de calcul

La calculatrice dispose d'une pile pouvant être remplie d'éléments sur lesquels pourront être appliquées des fonctions.  
Un ajout dans la pile s'effectue par l'intermédiaire de la zone de saisie.

#### Peuvent-être ajoutés à la pile

**Des chiffres seuls** : entrer un chiffre dans la zone de saisie et appuyez sur le bouton = (Entrée)

Pour entrer un complexe (le mode complexe doit être actif), séparer sa partie réelle de sa partie imaginaire par le signe \$.  
Par exemple, pour 2+3i, saisir 2\$3, sans espace.

Pour entrer une fraction (le mode rationnel doit être actif), séparer son numérateur de son dénominateur par le signe /.  
Par exemple, pour 2/3, saisir 2/3, sans espace.

**Des calculs** : entrer une série de nombre et d'opérateurs, séparés entre eux par des espaces. Le résultat du calcul sera ajouté à la pile. Par exemple, pour 2+2, entrer 2 2 +

**Des expression** : une expression est une série de nombres et/ou d'opérateurs encadrés de guillemets : """. L'expression est ajoutée dans l'état à la pile. Pour calculer son contenu, appuyer sur la touche EVAL (voir plus bas, Bouton EVAL).  
Exemples : "2 2 +" ou "1 2 3 4 + - \*" ou "1 2 3" etc.

**Des opérateurs seuls** : l'ajout d'un opérateur seul entraînera le dépilement des deux derniers éléments de la pile (ou du dernier dans le cas d'un opérateur unaire comme le sinus, ou le carré), et l'ajout du résultat de l'opération à la pile.

#### Description des fonctionnalités

**Bouton +** : Somme des deux nombres/expressions précédent(e)s

**Bouton -** : Soustraction des deux nombres/expressions précédent(e)s

**Bouton \*** : Multiplication des deux nombres/expressions précédent(e)s

**Bouton /** : Division des deux nombres/expressions précédent(e)s

**Bouton MOD** : Reste de la division entière des deux entiers précédents (mode entier seulement)

**Bouton POW** : Fonction puissance : 2 3 POW <=> 2 puissance 3

**Bouton SQR** : Fonction carré : 2 SQR <=> 2 au carré

**Bouton SQRT** : Fonction racine carrée

**Bouton CUBE** : Fonction cube

**Bouton SIN** : Fonction sinus

**Bouton COS** : Fonction cosinus

**Bouton TAN** : Fonction tangente

**Bouton SINH** : Fonction sinus hyperbolique

**Bouton COSH** : Fonction cosinus hyperbolique

**Bouton TANH** : Fonction tangente hyperbolique

**Bouton LN** : Fonction logarithme népérien

**Bouton LOG** : Fonction logarithme décimal

**Bouton INV** : Fonction inverse

**Bouton SIGN** : Inversion du signe (modes réel et rationnel uniquement)

**Bouton !** : Fonction factorielle (mode entier uniquement)

**Bouton SWAP** : Inverse les éléments x et y de la pile (x et y paramétrables dans Outils -> Paramètres)

**Bouton SUM** : Somme des x derniers éléments de la pile (x paramétrable dans Outils -> Paramètres)

**Bouton MEAN** : Moyenne des x derniers éléments de la pile (x paramétrable dans Outils -> Paramètres)

**Bouton DUP** : Duplique le dernier élément de la pile

**Bouton =** : ajoute le contenu de la zone de saisie à la pile de calcul. Effectue le calcul si un opérateur est présent

**Bouton EVAL** : calcule le contenu d'une expression entrée dans la zone de saisie ou dans la pile

**Bouton CLEAR** : Retire le dernier élément de la pile

**Bouton CLEAR ALL** : Vide la pile

**Bouton <--** : Efface un caractère de la zone de saisie

**Bouton EMPTY** : vide la zone de saisie

### Liste des raccourcis clavier

**Entrée** : Ajouter un élément à la pile (=)

**Suppr** : Supprimer le dernier élément de la pile (CLEAR)

**Ctrl+C** : Vider la pile (CLEAR ALL)

**Ctrl+E** : Evaluer une expression (EVAL)

**Effacement** : Effacer un caractère de la zone de saisie (<--)

**Ctrl+Effacement** : Effacer la zone de saisie (EMPTY)

**Ctrl+M** : Fonction Modulo (MOD)

**Ctrl+Z** : Annuler

**Ctrl+Y** : Rétablir