Architecture logicielle de CalcMVC
v. 1.0

X. Skapin

21 janvier 2025

Table des matières

Ce document décrit l'architecture logicielle du projet `CalcMVC`. L'exécutable est une calculatrice la plus générique en termes de type d'opérandes et d'opérations. L'objectif est d'exploiter le "*Modèle / Vue / Contrôleur*" pour concevoir cette application : dans ce contexte, la Vue et le Modèle doivent être découplés au maximum, la communication entre les deux étant régulée par le Contrôleur.

Cette application manipule également un certain nombre de "*Patrons de conception*" ("*Design Patterns*" [1]) qui font partie intégrante de la méthodologie de la Programmation Orientée Objet.

La suite de ce document décrit les structures de données de l'application en `JavaFX`, réparties par `paquetages`. Des extraits du diagramme de classe `UML` [2] de l'application sont fournis en complément. La documentation `Javadoc` fournie à part détaille les fonctionnalités de chaque structure.

1 common

Ce `paquetage` réunit les structures de données utilisées par les autres paquetages.

1.1 misc

Ce `paquetage` décrit des structures génériques. Pour des raisons pédagogiques, ce paquetage est inclus dans cette application précise mais pourrait être compilé en une librairie externe, utilisable dans n'importe quelle application.

1. Lien wikipedia
2. Ce diagramme n'affiche pas les méthodes privées, sauf dans le cas des constructeurs.

1.1.1 MyCommand

Cette `interface` représente le patron de conception "`Commande`", qui transforme une requête en un objet manipulable. Dans le cadre de notre application, nous considérons que chaque modèle, vue et contrôleur est dédié à une action à exécuter et va donc implémenter cette interface.

Cette interface est dite *fonctionnelle*, dans la mesure où elle ne contient qu'une fonction (forcément abstraite) : "`execute()`".

FIGURE 1 – `MyCommand`

1.1.2 MyObservable, MyConcreteObservable

`MyObservable` est une `interface` représentant le patron de conception "`Observateur/Observé`", qui définit un mécanisme de souscription permettant de notifier plusieurs objets (les "observateurs") à propos d'événements se produisant sur l'objet observé.

`MyConcreteObservable` est une `classe abstraite` implémentant cette `interface`. Elle agit en tant qu'objet "observé" et va stocker des références vers tous ses observateurs. C'est notamment ce mécanisme qui permettra aux objets observés `Modèles` et aux `Vues` d'envoyer des notifications à l'observateur `Contrôleur` qui les pilote.

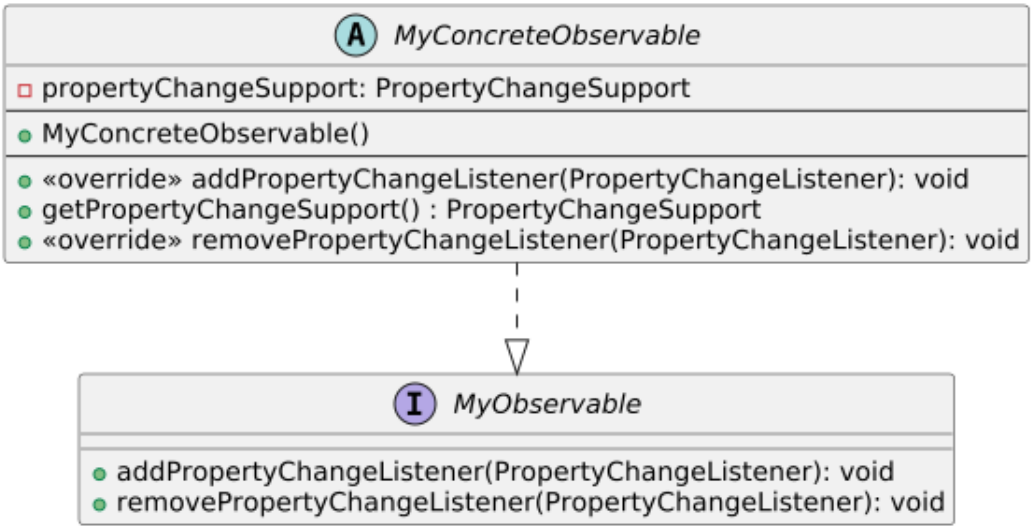

FIGURE 2 – `MyObservable, MyConcreteObservable`

1.1.3 PackageResearch

Cette `classe` regroupe les méthodes permettant d'explorer un `paquetage` et de rechercher des fichiers répondant à un certain critère.

Cette classe utilise le patron de conception "`Singleton`" qui définit une instance *unique* de classe. Cette technique est très classique pour éviter de créer plusieurs instances alors qu'une seule suffit pour répondre aux besoins. La contrepartie est que l'accès à cette instance doit être global, ce qui se traduit par "`public static`" en `Java`.

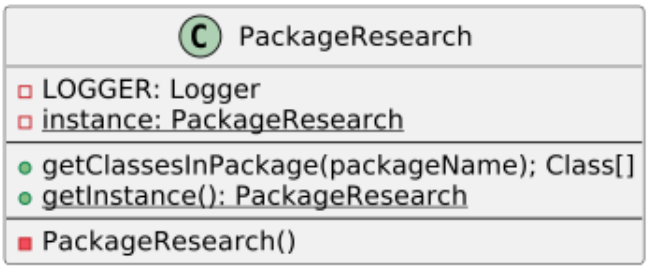

FIGURE 3 – `PackageResearch`

1.2 operation

Ce `paquetage` regroupe les structures orientées opérations, utilisées par les `Modèles/Vues/Contrôleurs`.

1.2.1 OperandType

Cette énumération décrit les types d'opérandes sur lesquels s'appliquent les opérations. Ce terme "opérations" est assez général : il prend en compte tout élément 'pas seulement numérique) qui peut être traité.

Actuellement, les types en question sont :
— `LONG` pour appliquer des opérations sur des valeurs de type `Long` ;
— `EXTRA` pour représenter des valeurs non numériques ;
— `UNKNOWN` pour représenter des valeurs par défaut.

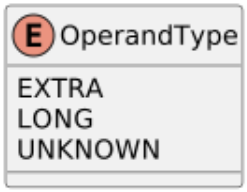

FIGURE 4 – `OperandType`

1.2.2 OperationType

Cette énumération liste les opérations (au sens général du terme) et les raccourcis textuels associés. Actuellement, cette liste comporte :
— des opérations numériques binaires : `ADD`, `DIV`, `MUL`, `SUB` ;
— une opération numérique unaire : `SQU` (carré d'une valeur) ;
— l'opération `QUIT` pour quitter l'application ;
— `UNKNOWN` pour représenter des valeurs par défaut.
Il y a une correspondance entre `OperandType` et `OperationType` : il faut les utiliser de manière pertinente pour les classes associées au `MVC`.

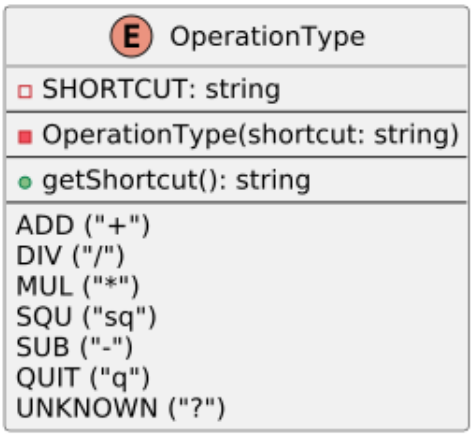

FIGURE 5 – `OperationType`

1.2.3 OperationTypeException

Une exception traitant les erreurs liées aux types d'opérations.

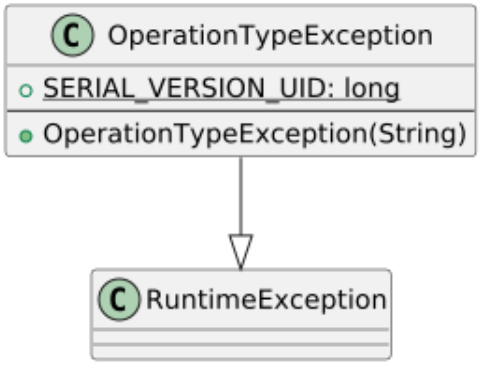

FIGURE 6 – `OperationTypeException`

1.2.4 OperandAnnotation

Une `annotation` est une forme de `méta-donnée` qui fournit des informations sur une structure, mais pas sur le programme en lui-même. Une `annotation` classique est `@override`, qui indique qu'une méthode surcharge ou redéfinit une autre méthode ; cette information est utile pour le développeur, le compilateur n'en a pas besoin.

Ici, `OperandAnnotation` est une annotation spécialement créée pour indiquer qu'une structure est associée à un type d'opérande donné. Grâce à cette information, une recherche dynamique de toutes les classes associées à cet opérande précis sera réalisée.

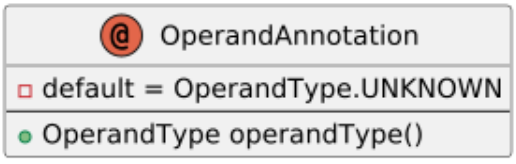

FIGURE 7 – `OperandAnnotation`

1.2.5 OperationCommon

Cette classe est un singleton. Elle permet en particulier de créer dynamiquement (par le mécanisme de *réflexivité* de `Java`) une série de classes traitant le même type d'opérandes et situées dans le même paquetage.

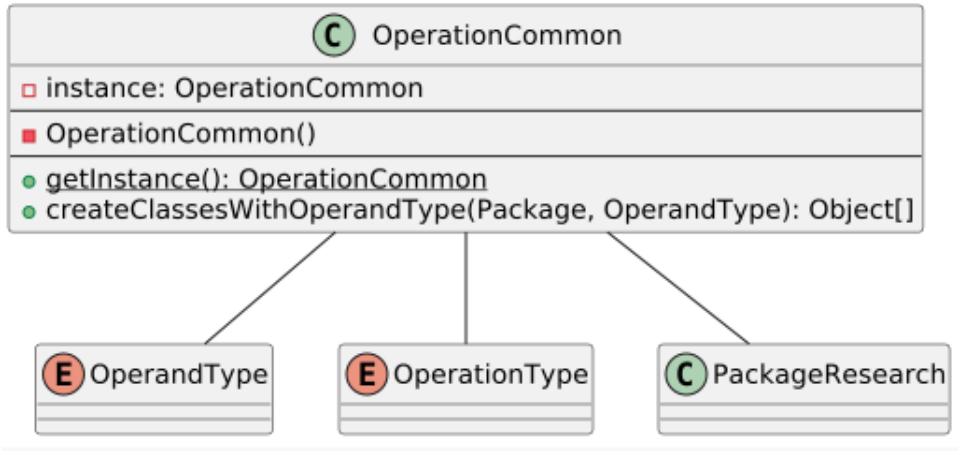

FIGURE 8 – `OperationCommon`

1.2.6 CommonBase

Cette classe abstraite sert de base à toutes classes définies selon le modèle `MVC`. Dans ce cadre, elle se comporte comme un `Observateur` et une `Commande`. Elle implémente également l'interface `Comparable` pour différencier les classes filles.

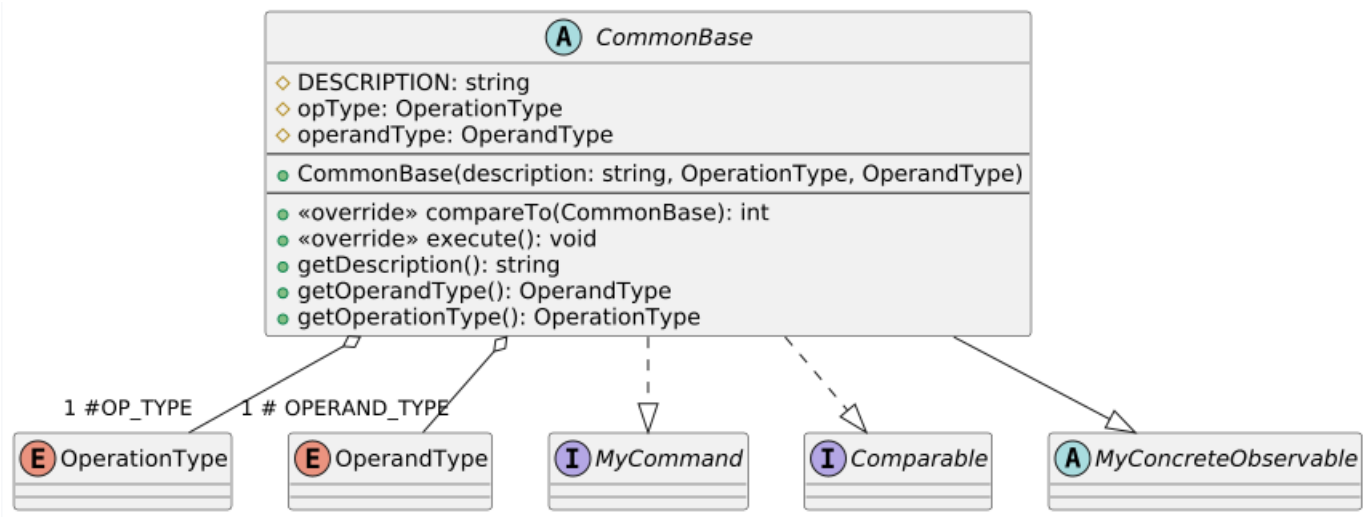

FIGURE 9 – `CommonBase`

1.2.7 CalcCommonBase

Cette `classe abstraite` regroupe toutes caractéristiques communes aux calculatrices, définies selon le modèle `MVC`. Elle contient donc une collection de classes `CommonBase`.

La méthode `createOpElements()` est définie sous deux formes :
— déclarée en tant que `méthode abstraite`, pour être spécialisée (surchargée) par les classes filles ;
— et définie en tant que `méthode concrète`, pour factoriser le code commun utilisé par les classes filles.

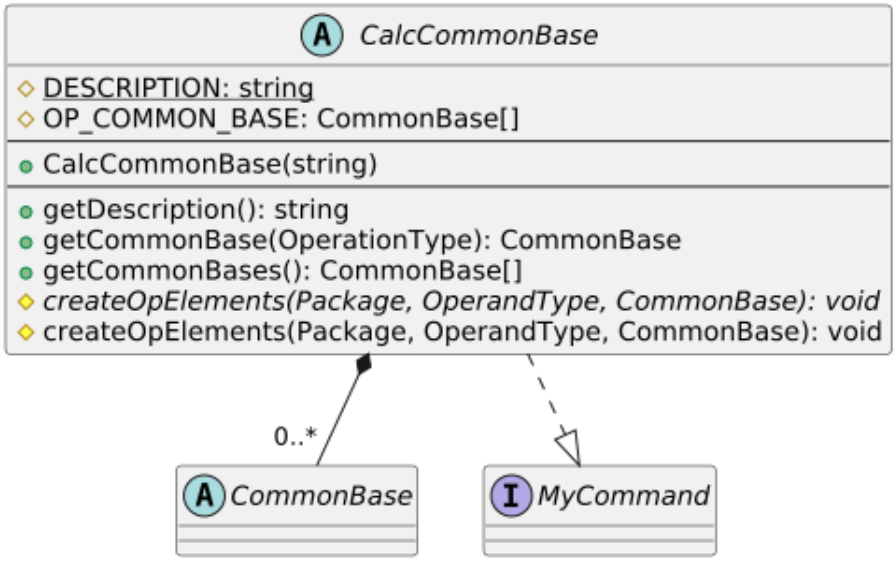

FIGURE 10 – `CalcCommonBase`

2 model

Ce paquetage regroupe toutes les structures relatives au *Modèle* du `MVC`.

2.1 Model

Cette classe `abstraite` est la classe mère de toutes les structures exécutant un traitement lié au *noyau fonctionnel* de l'application sur des opérandes (par exemple, le calcul d'une opération arithmétique). Elle est `paramétrée` (notation : "`<T>`") par le type d'opérande sur lequel l'opération sera appliquée.

Les opérandes à traiter parviennent à cette classe sous forme de `String` : la méthode `abstraite convertOperandsToTyp` convertit ces opérandes en un type conforme au traitement attendu. la méthode `convertResultToString()` convertit le résultat de l'opération en `String`.

Cette classe hérite de `CommonBase` et implémente `MyCommand` par transitivité : le traitement est déclenché en appelant la méthode `execute()`. La méthode abstraite `apply()` est également utilisée, pour spécialiser le traitement par les classes filles.

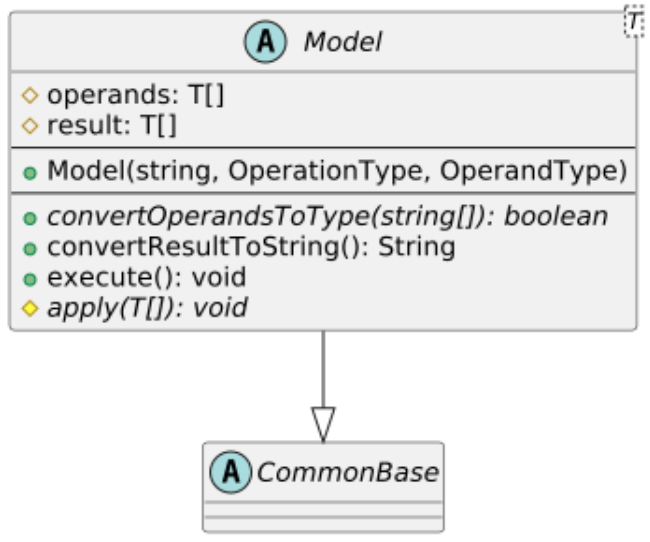

FIGURE 11 – `Model`

2.2 CalcModel

Cette classe `abstraite` représente une calculatrice paramétrée héritant de `CalcCommonBase`. Ses éléments sont des `Model` de même paramètre.

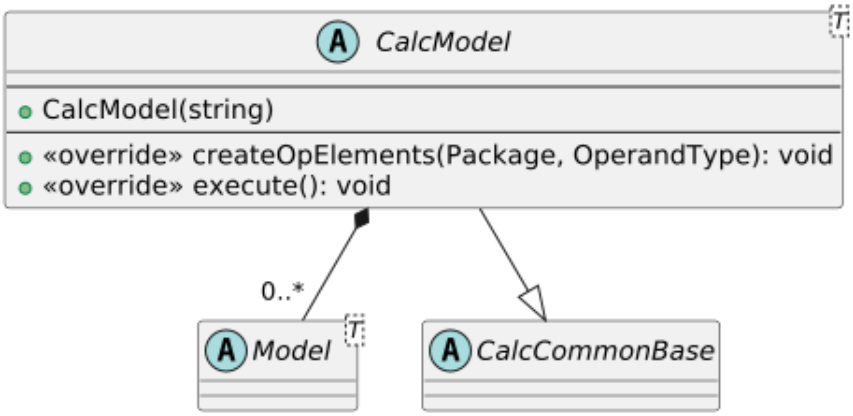

FIGURE 12 – `CalcModel`

2.3 UnaryModel

Cette classe `abstraite` est la classe mère de tous les `Model` représentant une opération numérique unaire, c'est-à-dire agissant sur un seul opérande : par exemple, $x \to x^2$.

Un `Model` contient une collection d'opérandes ; toute opération unaire est appliquée sur *chaque élément* de cette collection, et les résultats sont eux-mêmes stockés dans une collection.

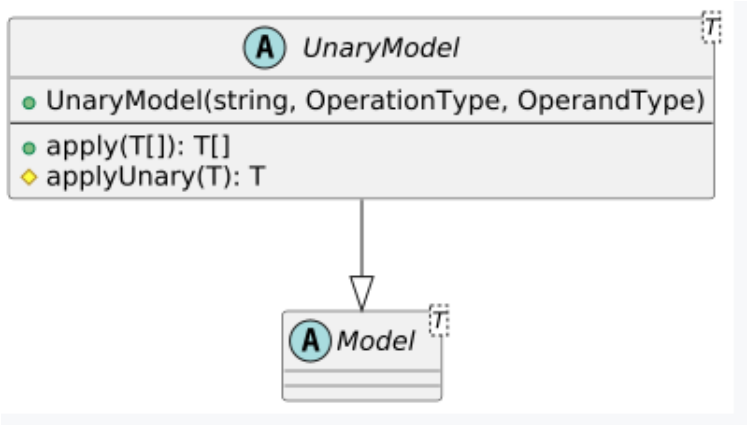

FIGURE 13 – `UnaryModel`

2.4 BinaryModel

Cette classe `abstraite` est fondée sur le même principe que `UnaryModel`. Mais l'opération associée est binaire et s'applique donc sur deux opérandes, par exemple pour l'addition : $x, y \to x + y$.

Lorsque cette opération est appliquée sur l'ensemble des opérandes du `Model` sous-jacent, un seul résultat global est calculé.

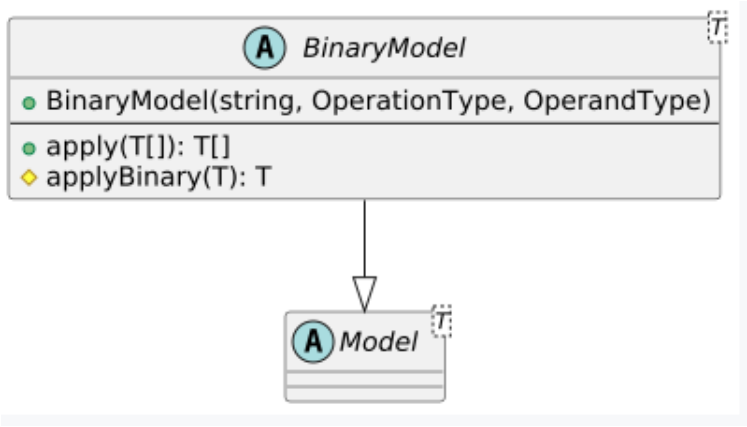

FIGURE 14 – `BinaryModel`

2.5 longmodel

2.5.1 CommonLongModel

Cette classe `abstraite` est un `singleton` regroupant les traitements communs à toutes les opérations du `Model` portant sur des `Long`.

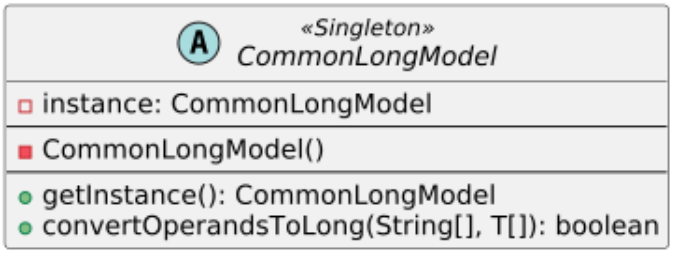

FIGURE 15 – `CommonLongModel`

2.5.2 UnaryLongModel

Cette classe `abstraite` hérite de `UnaryModel` pour spécialiser le traitement d'opérations numériques *unaires* sur des `Long`. Pour ce faire, elle utilise les méthodes définies dans `CommonLongModel`.

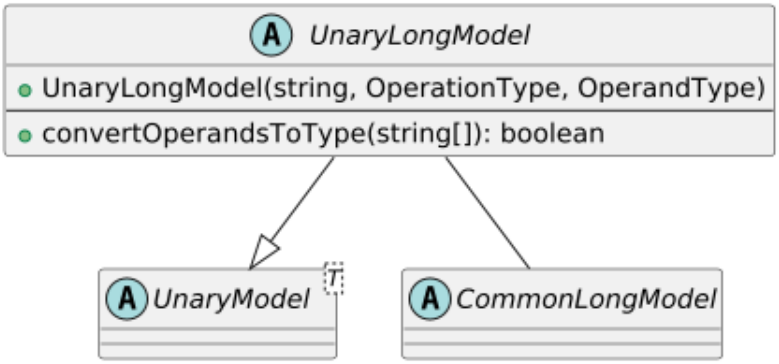

FIGURE 16 – `UnaryLongModel`

2.5.3 BinaryLongModel

Cette classe `abstraite` reprend le même principe que `UnaryLongModel`, mais pour des opérations numériques *binaires* sur des `Long`.

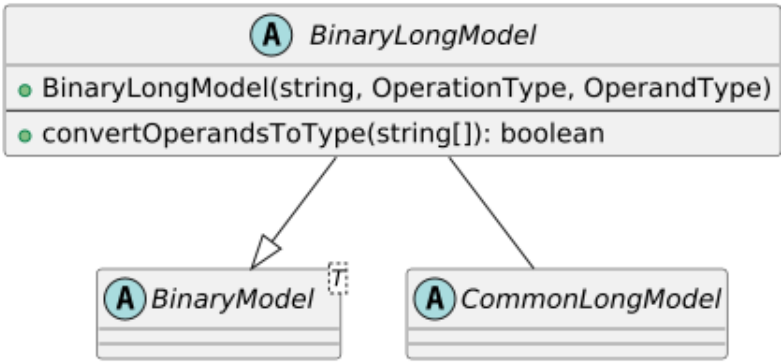

FIGURE 17 – `BinaryLongModel`

2.5.4 AddLongModel et cie.

Les classes `AddLongModel`, `DivLongModel`, `MulLongModel` et `SubLongModel` ont toutes le même profil : elles représentent respectivement les opérations *binaires* d'addition, division, multiplication et soustraction, sur des `Long`.

Elles utilisent l'`annotation` `OperandAnnotation` pour être instanciées automatiquement par `OperationCommon`.

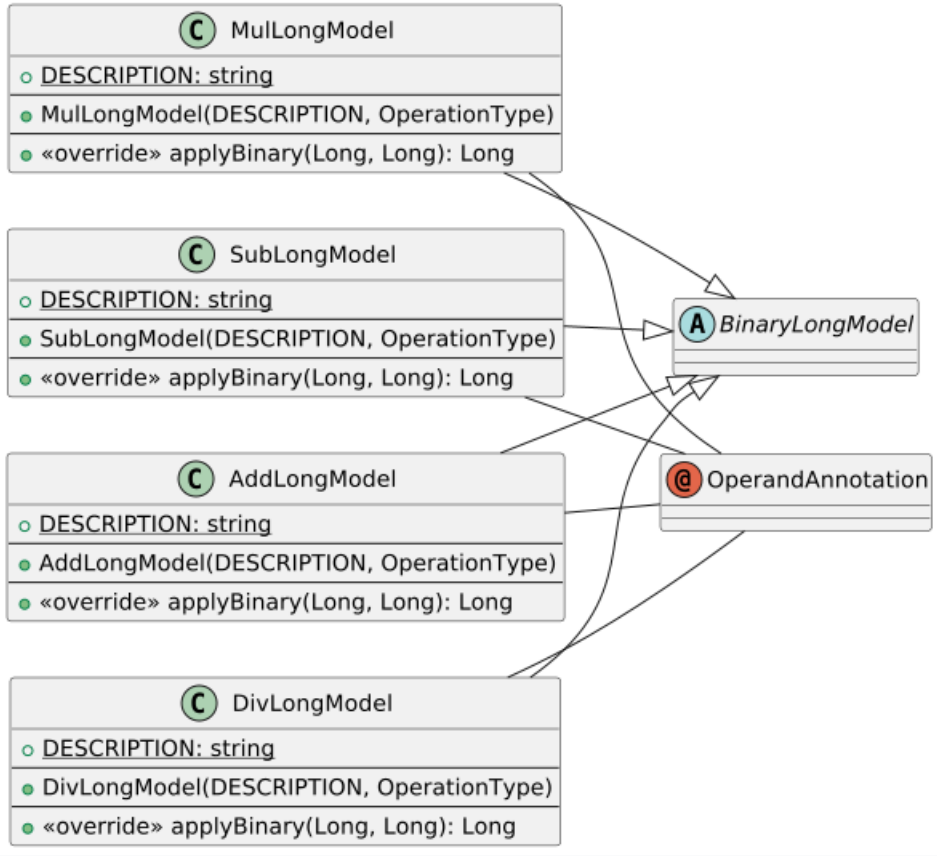

FIGURE 18 – Opérations dans `longmodel`

2.5.5 SquareLongModel

Cette classe représente l'opération *unaire* de carré sur des `Long`.

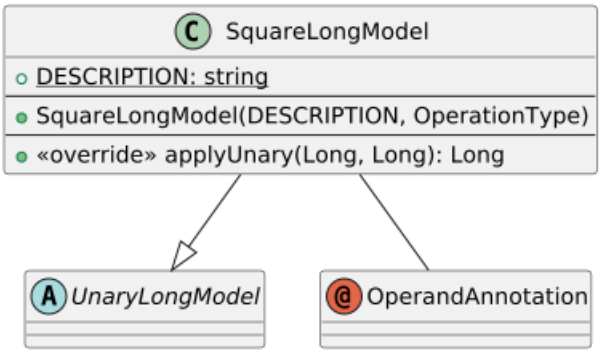

FIGURE 19 – `SquareLongModel`

2.5.6 CalcLongModel

Cette classe spécialise `CalcModel` pour des opérations sur des `Long`.

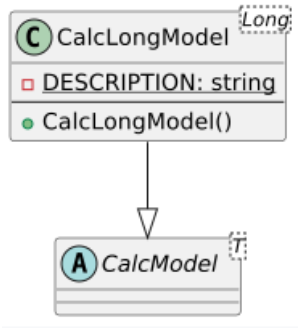

FIGURE 20 – `CalcLongModel`

2.6 extramodel

Ce paquetage réunit les classes qui ne représentent pas des opérations numériques, mais suivent le modèle `MVC`.

2.6.1 QuitModel

Cette classe représente les instructions exécutées au niveau du noyau fonctionnel lorsque l'utilisateur a décidé de quitter l'application.

On peut noter qu'en héritant de la classe paramétrée `Model`, cette classe doit elle-même être paramétrée, même si ce paramètre (qui représente un type d'opérande numérique) n'est pas utilisé.

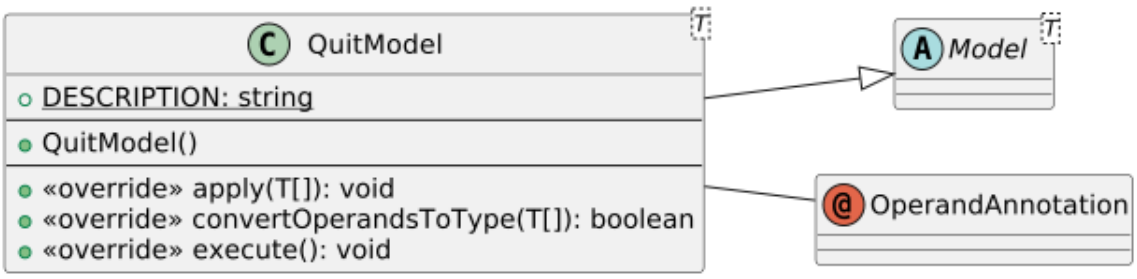

FIGURE 21 – `QuitModel`

3 view

Ce `paquetage` regroupe l'ensemble des classes liées à la partie *Vue* du `MVC`. Cette vue gère uniquement la ligne de commande.

3.1 View

Cette classe `abstraite` hérite de `CommonBase` et représente la partie "*Vue*" d'une opération générique. En particulier, elle interagit avec l'utilisateur pour récupérer les entrées et afficher les résultats.

Elle possède un membre de type `OperationType` pour déterminer quelle est l'opération précise à exécuter, en fonction du choix de l'utilisateur.

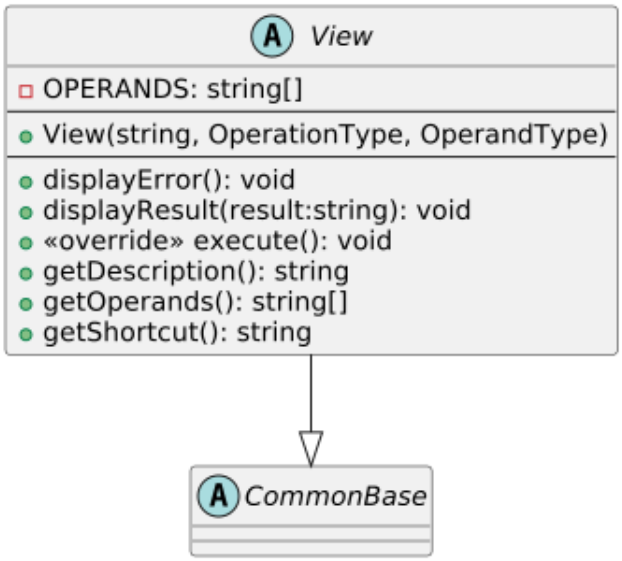

FIGURE 22 – `View`

3.2 CalcView

Cette classe abstraite hérite de `CalcCommonBase` et représente la partie "*Vue*" d'une calculatrice générique. Tous ses éléments, qui héritent de `CommonBase`, seront générés dynamiquement par réflexivité.

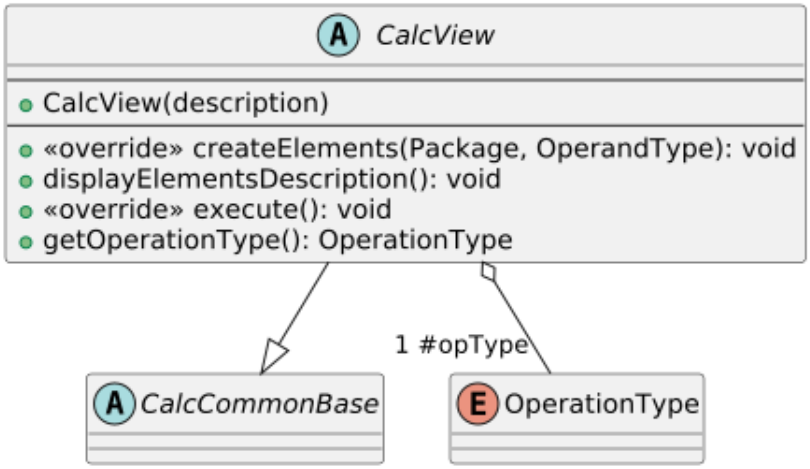

FIGURE 23 – `CalcView`

3.3 longview

Ce *paquetage* contient les *Vues* associées à toutes les opérations numériques. La récupération des opérandes entrés par l'utilisateur et l'affichage des résultats sont les mêmes, que les opérations soient unaires ou binaires.

3.3.1 AddLongView et cie.

Les classes `AddLongView`, `DivLongView`, `MulLongView`, `SquareLongView` et `SubLongView` ont toutes le même profil : elles représentent la récupération d'opérandes et l'affichage de résultats des opérations équivalentes présentées dans le paquetage `longmodel`.

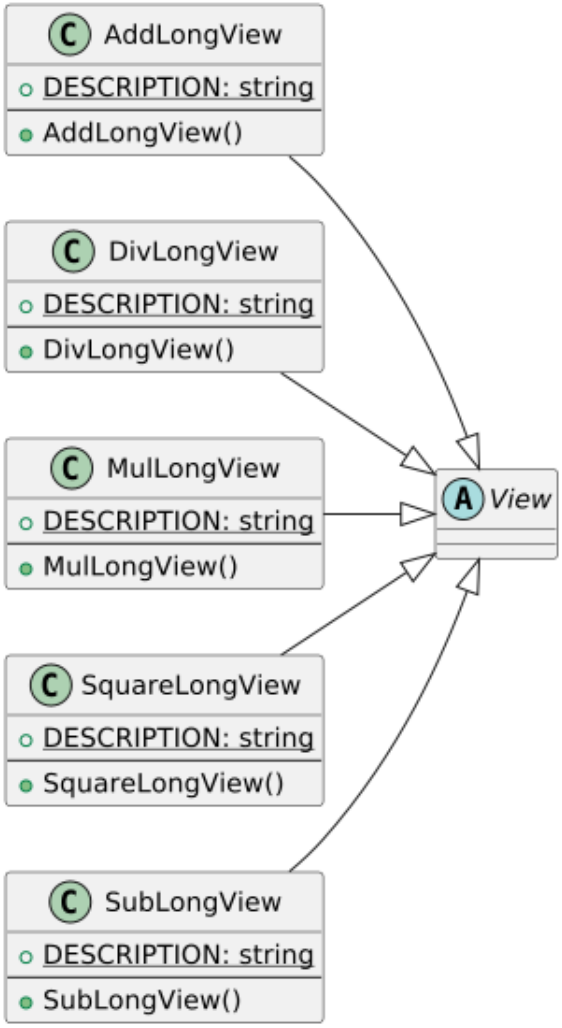

FIGURE 24 – Opérations dans `longview`

3.3.2 CalcLongView

Cette classe est simplement une spécialisation de `CalcView`, pour traiter les opérandes de type `Long`.

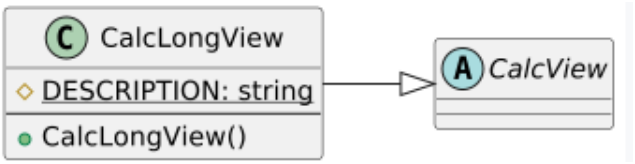

FIGURE 25 – `CalcLongView`

3.4 extraview

Ce `paquetage` contient les *Vues* qui ne sont pas liées à des opérations arithmétiques.

3.4.1 QuitView

Cette classe est l'équivalente de `QuitModel` pour la partie *Vue* du `MVC`.

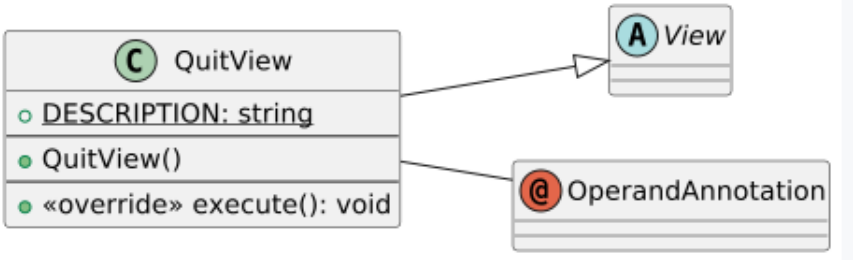

FIGURE 26 – `QuitView`

4 controller

Ce `paquetage` regroupe toutes les classes relatives à la partie *Contrôleur* du `MVC`.

4.1 Controller

Cette classe `abstraite` pilote les classes `Model` et `View` et joue le rôle d'observateur sur ces classes. Lorsque ces dernières ont terminé un traitement, elles envoient des notifications au `Controller`.

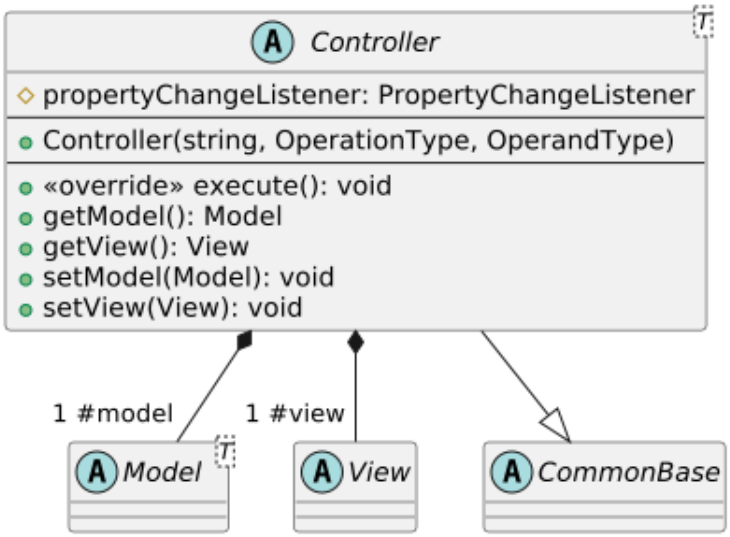

FIGURE 27 – `Controller`

4.2 CalcController

Cette classe `abstraite` pilote les classes `CalcModel` et `CalcView` et joue le rôle d'observateur sur ces classes. Lorsque ces dernières ont terminé un traitement, elles envoient des notifications au `Controller`.

4.3 longcontroller

Ce paquetage regroupe les *Contrôleurs* liés aux opérations sur des `Long`.

4.3.1 AddLongController et cie.

Les classes `AddLongController`, `DivLongController`, `MulLongController`, `SquareLongController` et `SubLongController` ont toutes le même profil : elles pilotent les classes de homonymes dans les paquetages `View` et `Model`.

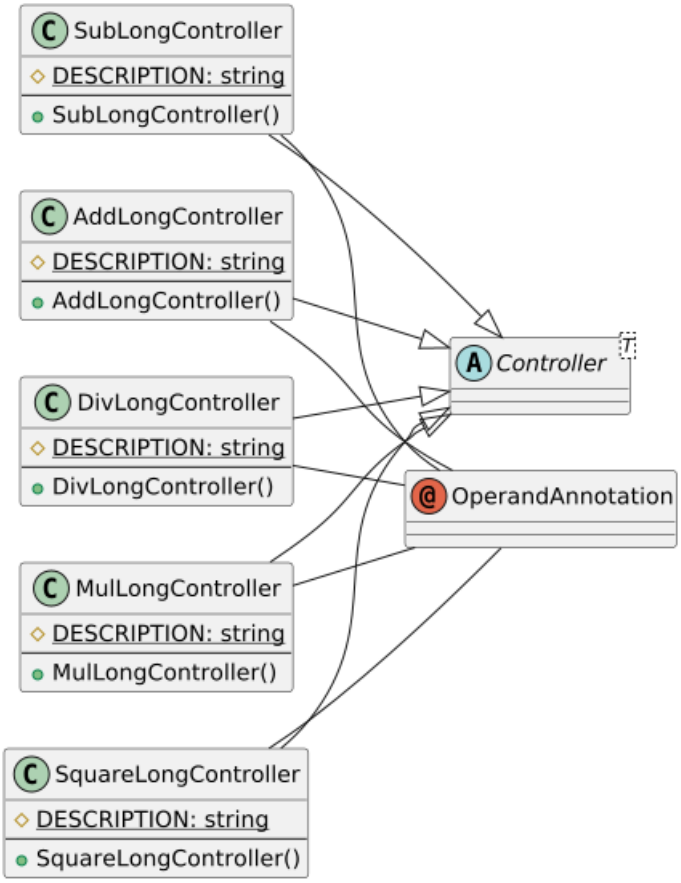

FIGURE 28 – *Contrôleurs* d'opérations dans longcontroller

4.3.2 CalcLongController

Cette classe est simplement une spécialisation de `CalcController`, pour traiter les opérandes de type `Long`.

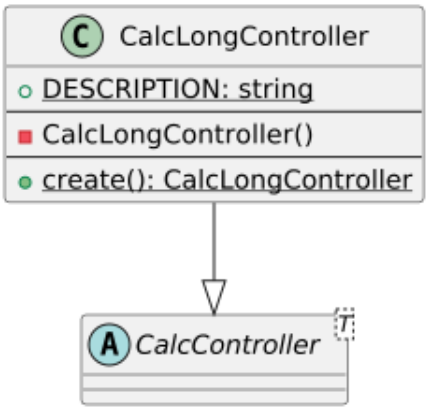

FIGURE 29 – `CalcLongController`

4.4 extracontroller

Ce `paquetage` contient les *Contrôleurs* qui ne sont pas liés à des opérations arithmétiques.

4.4.1 QuitController

Cette classe pilote `QuitModel` et `QuitView`.

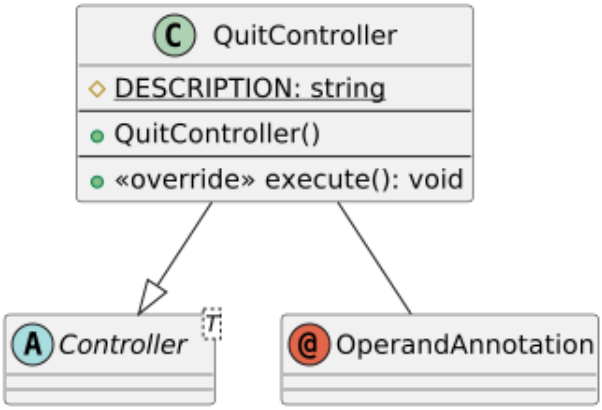

FIGURE 30 – `QuitController`

5 calcop

Ce paquetage gère la calculatrice générique.

5.1 Calc

Cette classe `abstraite` est paramétrée par le type d'opérande voulu et contient un `CalcController` qui pilote l'ensemble des autres classes.

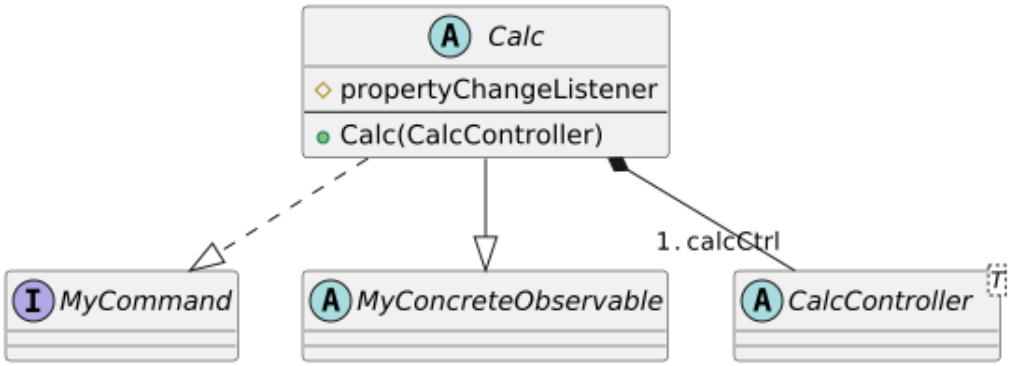

FIGURE 31 – `Calc`

5.2 calclong

Ce paquetage contient uniquement `CalcLong`.

5.2.1 CalcLong

Cette classe `concrète` représente la calculatrice globale dédiée au traitement des opérandes de type `Long`.

6 main

`Paquetage` principal.

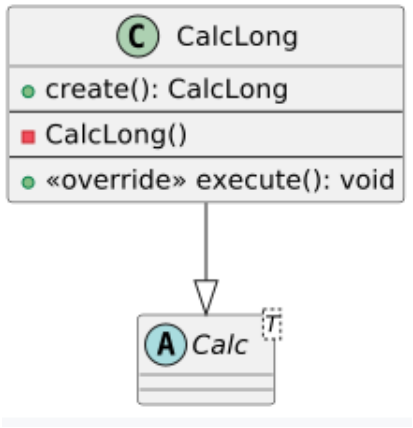

FIGURE 32 – `CalcLong`

6.1 MainCalcMVC

Classe héritant de `Application`. Son rôle consiste principalement à créer une instance de `CalcLong` et de lancer son exécution.

7 Exécution

Lorsque le programme est démarré, l'affichage, qui repose sur la *Vue* du `MVC`, est le suivant (rappelons qu'actuellement, l'unique Vue définie pour cette application est textuelle) :

```
Calculator for LONG operands. Select one operation's shortcut below.
ADD LONG operation. Shortcut: +
DIV LONG operation. Shortcut: /
MUL LONG operation. Shortcut: *
SQUARE LONG operation. Shortcut: sq
SUB LONG operation. Shortcut: -
QUIT LONG calc. Shortcut: q
Your choice :
```

FIGURE 33 – `Démarrage de l'application`

Le code actuel est défini seulement pour le type `Long` (il pourrait facilement être étendu à d'autres types). Les opérations disponibles sont affichées, ainsi que leur raccourci. De nouveau, il est facile d'ajouter d'autres opérations, en prenant les classes concrètes représentant les triplets `MVC` existants comme modèles.

Dans l'exemple suivant, l'utilisateur a décidé de réaliser une addition : il sélectionne le raccourci associé (" `+`") puis entre une séquence de valeurs. Si ces valeurs ne représentent pas des nombres entiers (par exemple des nombres flottants, des chaînes de caractères, etc.), un message d'erreur sera affiché.
Une fois le calcul réalisé, le résultat est affiché. Puis la boucle principale reprend.

```
Your choice : +
Enter the sequence of space-separated operands.
45 67 89 00 21
The result of ADD LONG is [222].
Calculator for LONG operands. Select one operation's shortcut below.
ADD LONG operation. Shortcut: +
DIV LONG operation. Shortcut: /
MUL LONG operation. Shortcut: *
SQUARE LONG operation. Shortcut: sq
SUB LONG operation. Shortcut: -
QUIT LONG calc. Shortcut: q
Your choice :
```

FIGURE 34 – `Une addition`

Ensuite, l'utilisateur décide d'appliquer une fonction unaire, en l'occurrence le calcul du carré. Cette fois-ci, la séquence de valeurs entrées est traités différemment : le calcul est effectué sur chaque opérande. Puis de nouveau, le résultat est affiché et la boucle reprend.

```
Your choice : sq
Enter the sequence of space-separated operands.
34 56 78 90
The result of SQUARE LONG is [1156, 3136, 6084, 8100].
Calculator for LONG operands. Select one operation's shortcut below.
ADD LONG operation. Shortcut: +
DIV LONG operation. Shortcut: /
MUL LONG operation. Shortcut: *
SQUARE LONG operation. Shortcut: sq
SUB LONG operation. Shortcut: -
QUIT LONG calc. Shortcut: q
Your choice :
```

FIGURE 35 – `Passage au carré`

Enfin, l'utilisateur décide de quitter l'application : il choisit le raccourci associé et l'application se termine.

```
Your choice : q
Bye bye!

Process finished with exit code 0
```

FIGURE 36 – `Arrêt de l'application`