LO21 UTC

Rapport Projet LO21
Calculatrice en notation polonaise inversée

Auteurs : Han Jia et Antoine Hars

Introduction :

Dans le cadre de l'UV LO21, un cours sur la Programmation Orientée Objet, nous devons réaliser un projet de calculatrice en notation polonaise inversée.

1. Choix de conception :

- Dans le cas de la classe MainWindow contenant l'instanciation de l'interface graphique, nous avons choisi d'implémenter le design pattern Singleton pour limiter la création d'interfaces graphiques en parallèle. Notre souhait était de n'avoir qu'une seule interface de la calculatrice à la fois.

- Dans le cas de la classe Pile, nous avons choisi d'implémenter un Design Pattern Singleton pour éviter la création d'autres pile en parallèle. Notre souhait était d'avoir une pile pour une interface graphique de l'application.

- Nous avons choisi d'implémenter un design pattern Observer entre les classes Pile et MainWindow. Le but de cette implémentation est de mettre en œuvre une mise à jour automatique de l'affichage de la Pile dans l'interface graphique de MainWindow à chaque changement de la Pile (causé par l'appel de la méthode EvalPressed(), EntrerPressed() et autres méthodes qui modifient le contenu de la Pile).
  Après chaque traitement sur la Pile, une notification est envoyée de la part de la classe Pile et notifie à tous les Observateurs de la Pile qu'il y a une mise à jour a effectuer.

- Au niveau des différents types de Constante, nous avons choisi d'implémenter le design pattern Composite entre une Expression et le reste des types de Constantes regroupés dans une classe Type.
  Le fait d'utiliser ce design pattern nous assure une utilisation optimale des Constantes qu'elles soient de simples entiers, réels ou qu'elles soient un regroupement de Constantes.
  Cela nous permet de manipuler les Expressions et les autres Constantes de la même façon et d'avoir un code plus générique sur la manipulation des données.

- Le fait que nous pouvons utiliser des complexes ou non, nous a incité à implémenter un deuxième design pattern Composite entre des constantes de type Complexe et des constantes de type NoNComplexe.
  Les classes Complexe et NonComplexe sont toutes les deux des classes qui héritent de la classe Type mais la classe NonComplexe est composés de deux NonComplexe.

- Nous avons choisi d'implémenter un design pattern Factory dans la

gestion de la création des Constantes.
Le but de l'implémentation de ce design pattern, est d'assurer l'instanciation de la Constantes qui doit être instanciée et non d'une non adéquate par rapport aux attentes.
La création de l'objet est décidé par les classes filles de la classe Constantes. Nous avons écrit la classe FactoryConstante (abstraite) qui contient une méthode creeConstante(...) qui vérifie le type de données entré et qui instancie l'objet adéquate en retour. Pour éviter une instanciation d'objet de type NonComplexe ou encore Type ou Constante, nous avons mis ces trois classes en abstrait. Au final l'instanciation se fait selon les classes Expression, Reel, Entier, Rationnel et Complexe.
Le fait d'avoir couplé deux design pattern Composite et un design pattern Factory nous assure une bonne gestion au niveau de la création mais aussi de la manipulation des Constantes de la pile quelque soit leur type.

- Dans le but de faciliter l'utilisation des différents opérateurs, nous avons choisi de faire hériter la classe Operateur par la classe MainWindow.
  La classe Operateur est une classe regroupant toutes les différentes manipulations possibles sur la Pile et sur ses Constantes.
  L'héritage d'Operateur par MainWindow nous permet une gestion simple des Constantes. Le cœur du traitement du programme se trouve dans le MainWindow au final.

- Pour instancier une gestion des logs, nous avons choisi d'implémenter les classes LogSystem et LogMessage qui nous permettent d'afficher sur la ligne de commande tous les messages les plus importants comme la sauvegarde dans un fichier, le chargement d'un fichier dans la Pile ou la création de l'interface graphique.

- Pour la gestion des Constantes avec les modes, nous avons choisi d'utiliser un mode d'évaluation des Constantes entrées dans la ligne de commande pour ne créer que les Constantes du mode choisi. Cela induit sur une rigueur de l'utilisateur à veiller au mode et aux type choisis en paramètre.

2. Implémentations non terminées :

Lors de la réalisation du projet, plusieurs développement n'ont pas pu être finis dû fait à une mauvaise gestion du timing.

- Nous n'avons pu finaliser la sauvegarde du contexte de la pile dans un fichier ainsi que le chargement de ce contexte.
- La création des messages de log est bien opérationnelle mais elle n'est pas implémentée à tous les endroits requis, notamment au niveau des opérateurs qui manipule la Pile comme l'opérateur SWAP.

3. Cas d'utilisation :

Entier :
1 2 =
+ =
CLEAR
1 2 * =
50 - =
'2 3 + 4' =
* =
EVAL
2/3 = (erreur)

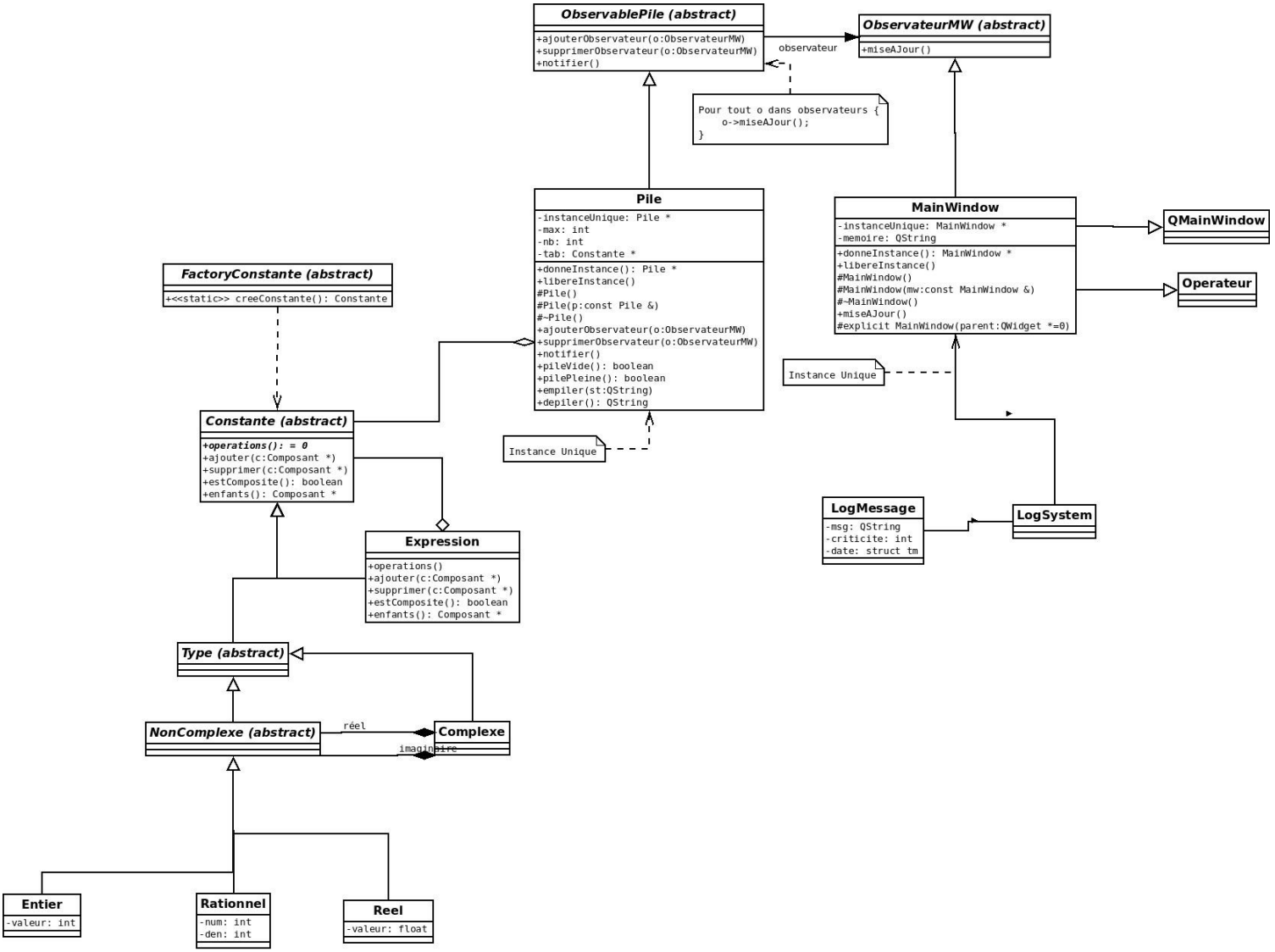
Rationnel :
2/8 4/8
annuler
rétablir
=
* =
CLEAR

Réel :
30.0 sin =
deg
30.0 sin =
CLEAR

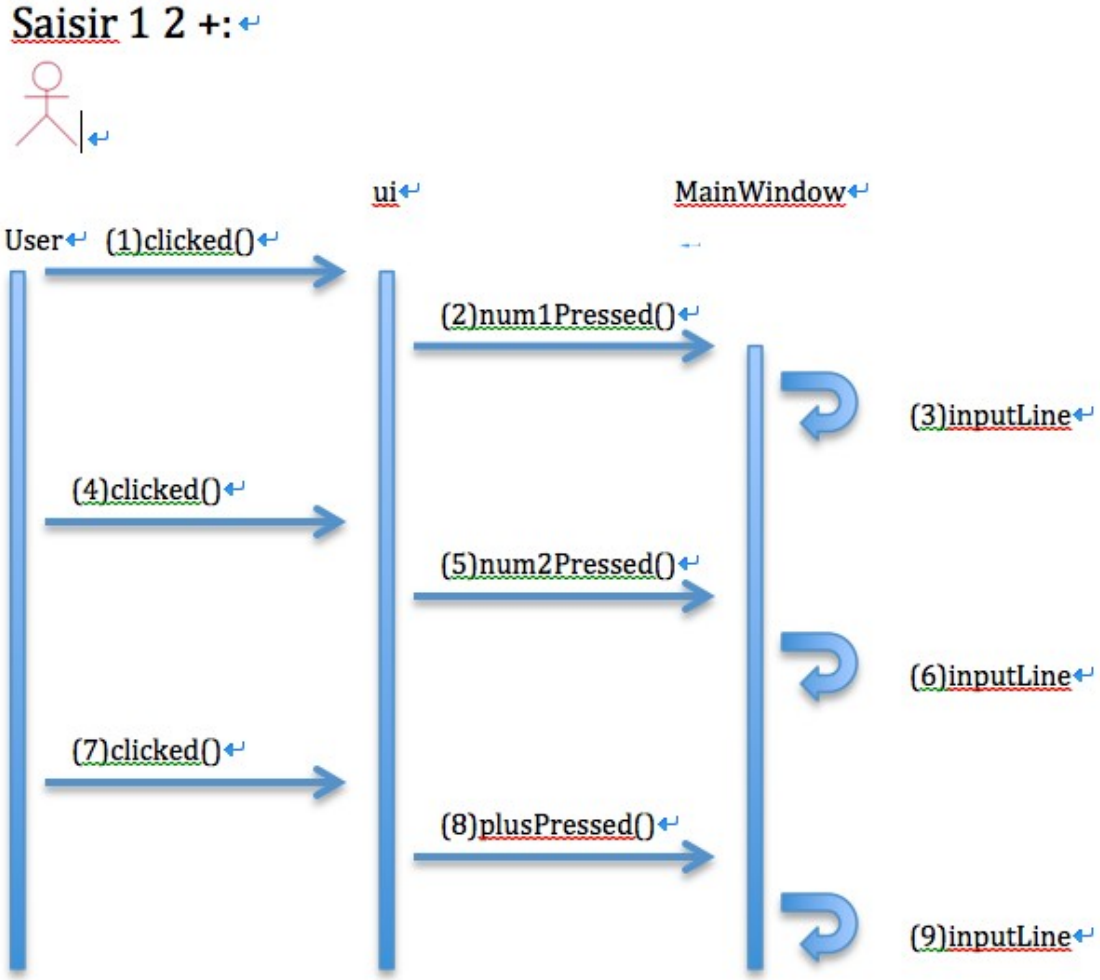
Complexe :
2.1$6/4 4$3 =
* =

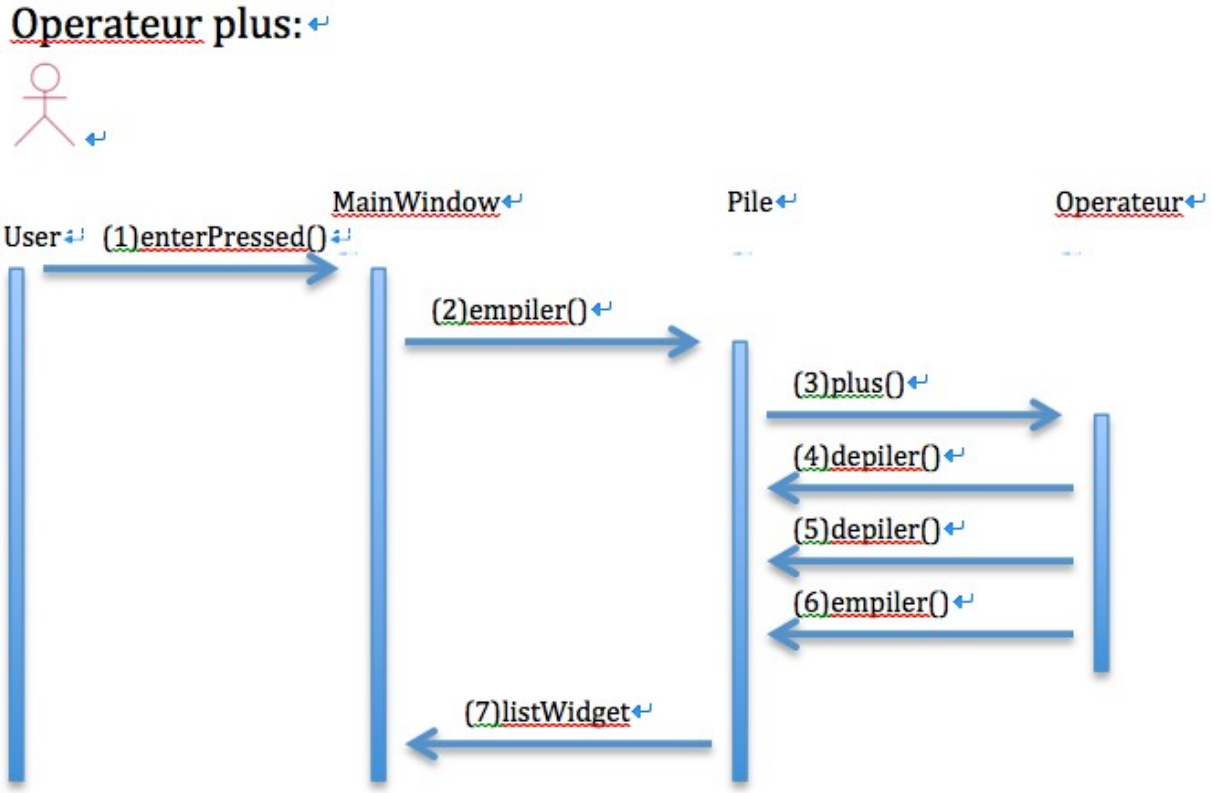
Non Complexe :
2 3 4 =
1 2 SWAP =
1 2 SWAP =
DROP =
3 SUM =

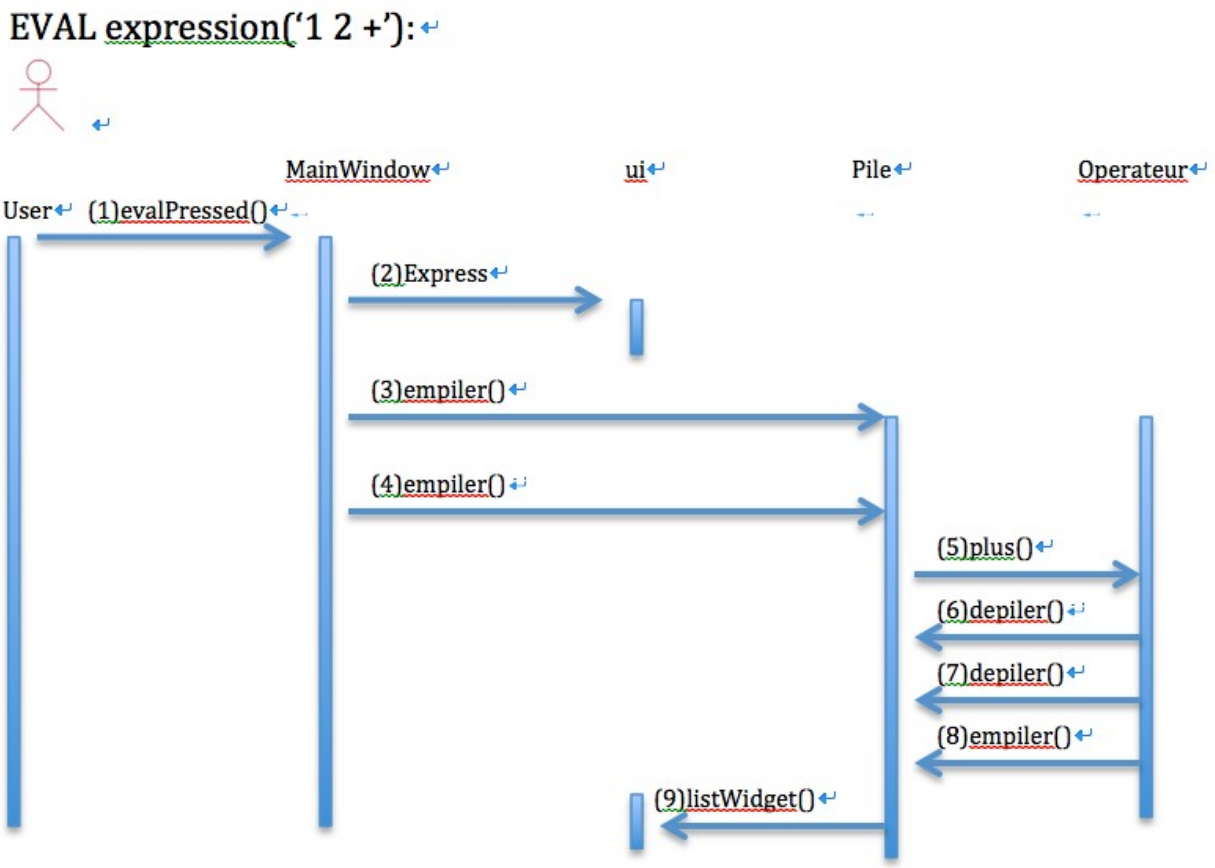
4. UML :


ObservablePile (abstract)
+ajouterObservateur(o:ObservateurMW)
+supprimerObservateur(o:ObservateurMW)
+notifier()
observateur
ObservateurMW (abstract)
+miseAJour()
Pour tout o dans observateurs {
o->miseAJour();
}
Pile
-instanceUnique: Pile *
-max: int
-nb: int
-tab: Constante *
+donneInstance(): Pile *
+libereInstance()
#Pile()
#Pile(p:const Pile &)
#~Pile()
+ajouterObservateur(o:ObservateurMW)
+supprimerObservateur(o:ObservateurMW)
+notifier()
+pileVide(): boolean
+pilePleine(): boolean
+empiler(st:QString)
+depiler(): QString
MainWindow
-instanceUnique: MainWindow *
-memoire: QString
+donneInstance(): MainWindow *
+libereInstance()
#MainWindow()
#MainWindow(mw:const MainWindow &)
#~MainWindow()
+miseAJour()
#explicit MainWindow(parent:QWidget *=0)
QMainWindow
Operateur
Instance Unique
Instance Unique
FactoryConstante (abstract)
+<<static>> creeConstante(): Constante
Constante (abstract)
+operations(): = 0
+ajouter(c:Composant *)
+supprimer(c:Composant *)
+estComposite(): boolean
+enfants(): Composant *
Expression
+operations()
+ajouter(c:Composant *)
+supprimer(c:Composant *)
+estComposite(): boolean
+enfants(): Composant *
LogMessage
-msg: QString
-criticite: int
-date: struct tm
LogSystem
Type (abstract)
NonComplexe (abstract)
réel
imaginaire
Complexe
Entier
-valeur: int
Rationnel
-num: int
-den: int
Reel
-valeur: float

6. Diagrammes de séquence :







Conclusion :

Ce projet nous a permis de mettre en œuvre les enseignements que nous avons pu suivre tout au long du semestre dans l'UV LO21 et fut donc très bénéfique. Le bémol dans la réalisation de ce projet est la mauvaise gestion du timing, les délais estimés dans la réalisation des différentes parties du projet ont été sous-estimées.