Abdoul Khadre Diallo – Lu Yan

Rapport Projet LO21 : Calculatrice à notation polonaise inversée

Introduction :

Dans le cadre de l'UV LO21, nous étions chargés de réaliser une calculatrice à notation polonaise inversée avec toutes les notions de conception orientée objet vues au cours de ce semestre. Notre calculatrice était donc chargée d'effectuer tous les calculs de base (addition, soustraction…) avec des variables allant des complexes aux entiers. La principale différence avec les calculatrices normales était que celle –ci, mis à part la technique de la notation polonaise inversée, serait gérée par une pile.

Diagramme de classes :

Notre diagramme de classes peut être scindé en deux parties :

- Données
- Interface

Données :

Dans les données, nous avons la classe générale Expression, à laquelle on a appliqué le design pattern composite. En effet, une expression est composée d'opérateurs et de nombres.
Dès lors, nous avons les classes Nombre et Operation.
La classe Nombre, comme nous le montre le diagramme (annexe), est la classe de base des nombres.
Les attributs de la classe Nombre sont au nombre de 5. Nous avons choisi ce nombre précis d'attributs car nous nous représentions un nombre comme deux « variables ». Chaque variable a une partie réelle, une partie imaginaire, un dénominateur réel, un dénominateur imaginaire. Il y a un attribut qui n'a pas de valeur numérique mais qui n'en est pas moins importante. C'est l'attribut mode. Car en effet, chaque construction d'un nombre se fait dans un mode. Par exemple, un complexe se construit dans le mode ''complexe''. De ce fait, on pouvait directement connaitre le mode de construction d'un nombre donc le type réel de ce nombre (en utilisant l'accesseur emmode()) de la classe Nombre .
Pour implémenter les méthodes de notre ensemble de données, nous nous sommes principalement servis de l'abstraction de la classe Nombre (donc nécessairement du polymorphisme) ce qui nous obligeait à ré implémenter ces méthodes pour chaque classe fille. Nous avons, en résumé utilisé les notions de spécialisation et de généralisation vues en cours. En

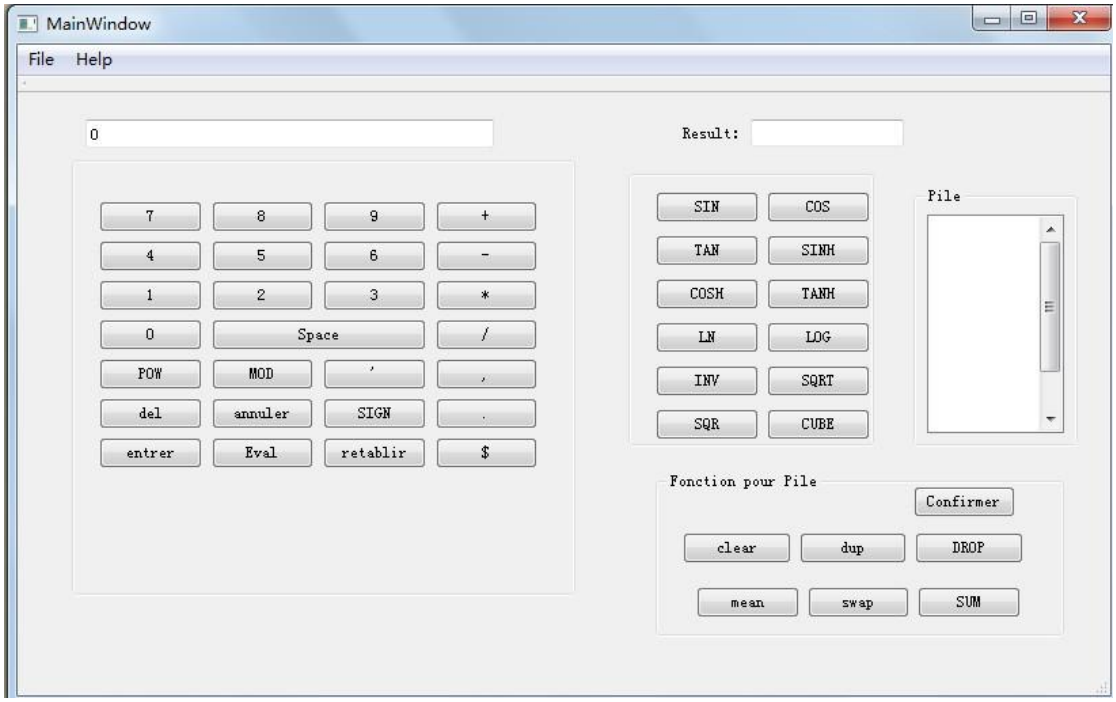
effet, pour les méthodes générales et communes à toutes les classes, on les implémentait directement dans la classe mère.
Par contre, celles qui étaient plus spécifiques étaient directement implémentées dans les classes concernées.
Nous avons donc utilisé cette manière de concevoir pour implémenter/surcharger les operateurs +, * , - et / qui seraient utilisés dans l'implémentation des opérations (addition, soustraction, multiplication…) de nombres.
La méthode setMode(), qui est une méthode protégée, a été utilisée dans la surcharge de l'operateur d'affectation. En effet, lorsqu'on affectait à un nombre, un nombre, on voulait s'assurait que, lorsque l'affectation, qu'ils soient tous deux du même type. D'autre part, les utilisateurs, d'après notre manière de concevoir n'auront pas le droit de modifier la ''nature '' d'un nombre.
Dans les opérations, de manière générale, il fallait dans chacune d'elle chercher la nature du nombre pour pouvoir d'abord dire si l'opération est possible et ensuite faire l'opération de manière spécifique. Car en fonction de la nature du nombre, l'opération changeait de comportement vis-à-vis de celui-ci. L'addition de deux complexes ne sera par exemple pas la même chose que l'addition d'un complexe et d'un réel ou d'un entier et d'un rationnel etc.

Concernant les opérations, nous avons deux types d'opérations : binaires et unaires. Dès lors, ce sont les classes filles concrètes de la classe abstraite opération. Ces deux classes ont chacune une méthode evaluer() qui a été définie virtuelle pure à la classe expression.

La classe complexe ne peut être construite que par deux variables de type chacune de type :
-double
-Reel
-Rationnel
-Entier
Et le constructeur sans argument appelle le constructeur de la classe de base (Nombre)
Chacune de ces classes est donc obligée de définir la méthode evaluer() afin d'être instanciable.
Concernant les opérations en tant que tel, on assigne à chaque opération une valeur entière choix. En fonction de la valeur de choix, l'on décidera quelle opération effectuer.

Remarques : concernant les opérations addition, soustraction, multiplication et division, nous avons choisi de toujours renvoyer un résultat sous forme de complexe. Car il y a eu un problème de conversion que nous n'avons pas pu anticiper. Il nous aurait fallu changer tout notre UML . Donc nous avons choisi de retourner un résultat sous forme de complexe.

Interface :

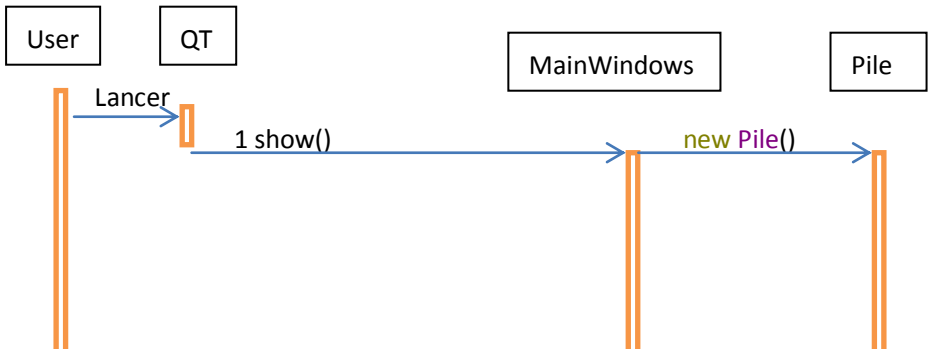


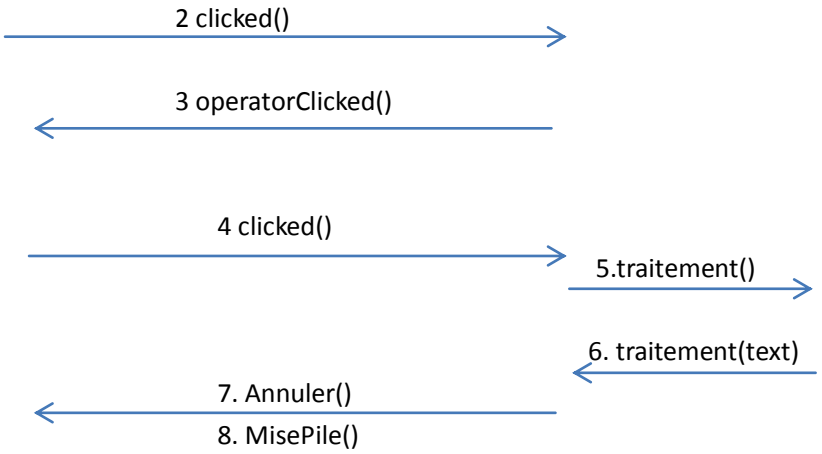
On a utilisé l'outil de design de QT pour créer notre interface de calculatrice. (comme ci-dessus)
Après, on a lié les signaux et les slots avec la fonction « connect ». Plus précisément, on a lié les fonctions de bouton avec l'action « cliquer ce bouton » avec le code ci-dessous.
connect(ui->INV,SIGNAL(clicked()),this,SLOT(operatorClicked()));
Le slot "*operatorClicked() "* sert à afficher tous que l'utilisateur clique dans la ligne de commande.

Si l'utilisateur clique le bouton « entrer », le slot «traitment » va appeler les fonctions de pile pour traiter les entrées dans la ligne de commande. Et il va mettre 0 sur la ligne de commande et va mettre à jour le contenu dans la pile et le label de résultat.

Diagrammes de Séquence :

Séquence : Entrer un nombre





Remarque:

« 2.click() » : Le utilisateur clique un bouton du nombre.

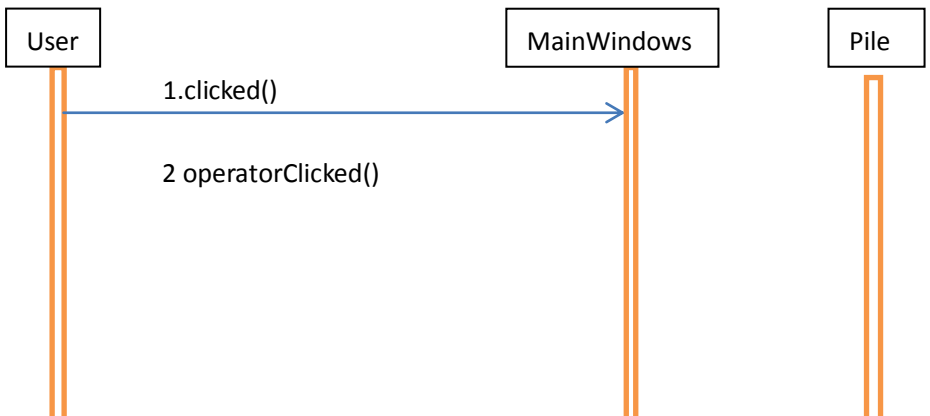
«3 operatorClicked() » : Le « MainWindows » affiche le nombre dans la ligne de commande.

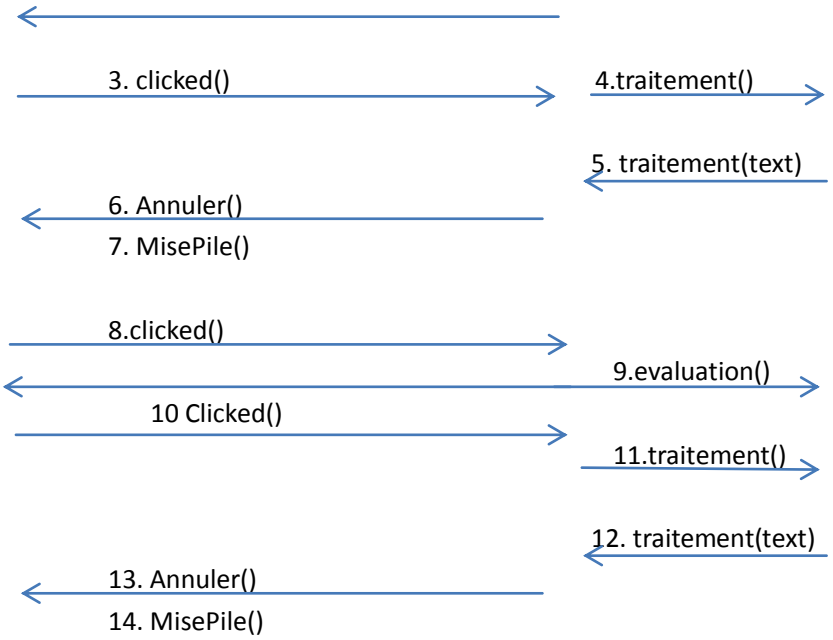
« 5.click() » : Le utilisateur clique un bouton du <entrer>.

« 7.Annuler » : La ligne de commande mise en zéro.

« 8. MisePile() » : Le Pile mise en jour. Le résultat va être affiche.

Séquence : Evaluer une expression constante





Remarque :

Les étapes de 1 à 7 sont comme le diagramme précédent sauf que les entrées par étape 3 deviennent deux guillemets, les chiffres et les opérations. C'est-à-dire on a besoins plusieurs clicks à l'étape 3.

« 8 clicked() » : l'utilisateur clique sur le bouton « eval ».

« 9.evaluation()» : la ligne de commande s'affiche le dernier élément (l'expression constante)de pile sans guillemets.

« 10.clicked()» : l'utilisateur clique sur le bouton « entrer ».

Conclusion :

Ce Projet a pris beaucoup de notre temps mais il nous a permis de mettre en pratique nos connaissances en conception et programmation orientées objet acquises en cours et d'en découvrir de nouvelles.

Le travail en binôme n'est pas facile surtout dans le timing et la communication mais cela allège tout de même la charge de travail. Nous sommes globalement satisfaits de l'utilisation plus poussée du C++ et du Framework Qt.