Université de Nantes
Faculté des sciences et Techniques
M1 ALMA - Architecture Logicielle

Mini editeur de texte en Scala

Alexis Ruchaud
Muriel Cadiot

Dossier de conception

Decembre 2014

Table des matières

CHAPITRE 1

Introduction

1.1 Presentation du projet

Ce projet consiste en la création d'un editeur de texte proposant à l'utilisateur différentes fonctionnalités. L'application a été réalisée en Scala à l'aide de l'IDE Eclipse tandis que les diagrammes de conception ont été créés à l'aide du plugin Eclipse Papyrus.
Le mini éditeur de texte permet à l'utilisateur d'entrer du texte dans une zone de travail (Buffer) et de réaliser différentes actions sur ce dernier. Ces actions sont celles que l'on peut retrouver dans la plupart des editeurs de texte : la possibilité de selectionner du texte, de le copier/couper, de le coller et des fonctions pour annuler un changement ou d'en rétablir un précedemment annulé. Ces commandes pourront être groupées sous forme de macro afin de permettre à l'utilisateur de créer lui même des commandes personnalisées afin de faciliter son travail.

1.2 Organisation du document

Pour présenter ce projet nous étudierons tout d'abord la partie analyse et conception de l'application en commençant par la problématique du projet en présentant un cas d'utilisation expliquant les possibilités de l'utilisateur. Nous verrons ensuite différents diagrammes de classe représentant le projet. Enfin nous observerons avec un diagramme de séquence l'execution du mini editeur.
Dans une deuxième partie nous présenterons la partie développement de l'application en présentant les classes du projet ainsi que leur rôle.
Enfin nous conclurons sur les perspectives du projet en présentant les différentes améliorations possibles du mini editeur.

CHAPITRE 2

Analyse et conception

2.1 Cas d'utilisation

Le cas d'utilisation suivant illustre l'utilisation d'une commande. Lorsque l'utilisateur execute une commande il appel la classe mère "Command" qui selon l'objet utilisé appel la commande correspondante. De plus le lien "include" reliant la commande à "Save" indique qu'une sauvegarde du Buffer sera réalisée à chaque execution d'une commande. Cette sauvegarde ce fait à l'aide du patron de conception Memento

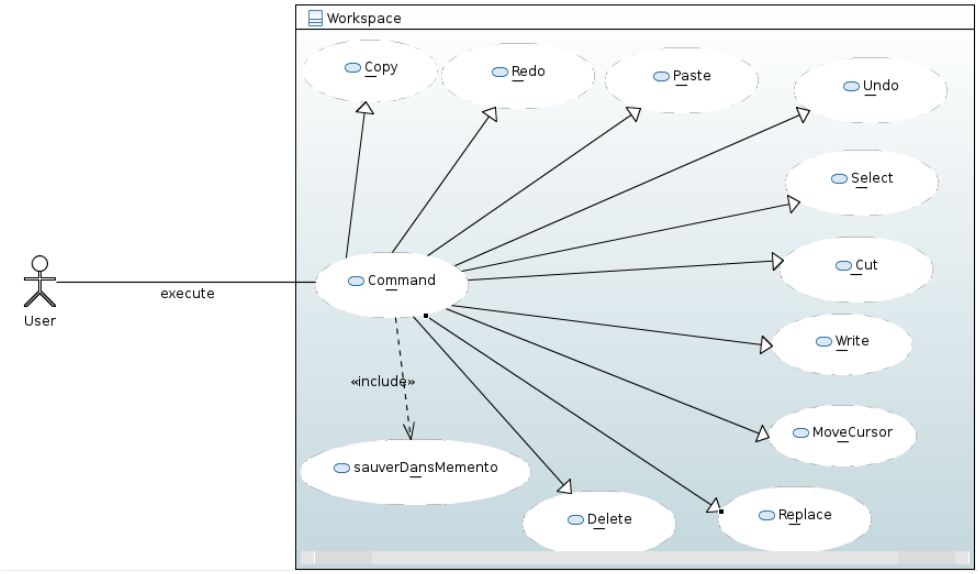

FIGURE 2.1 – UseCase : Command

2.2 Diagrammes de classe

Dans cette section, nous mettrons en avant les diagrammes de classes des différents patrons de conception qui sont contenus dans le mini éditeur de texte. Nous avons choisi de montrer en priorité ces diagrammes car ils sont l'angle le plus intéressant par lequel observer le programme. Vous trouverez en annexe le diagramme de classe complet de l'application.

2.2.1 Command pattern

Le pattern command est un patron de conception permettant de séparer le code appellant une action de l'action elle même. Il permet entre autres la création d'interface réalisant des commandes de tel façon que l'interface ne connaisse pas l'action réalisée par la commande.
Un éditeur de texte possèdant de nombreuses commandes (Copy , Cut , Undo etc..) il nous a semblé naturel de choisir ce pattern dans la création de l'application. En plus de l'encapsulation fournie par ce pattern, son utilisation nous permettra au besoin de créer de nouvelles commandes très facilement sans avoir a modifier/ajouter beaucoup de code.

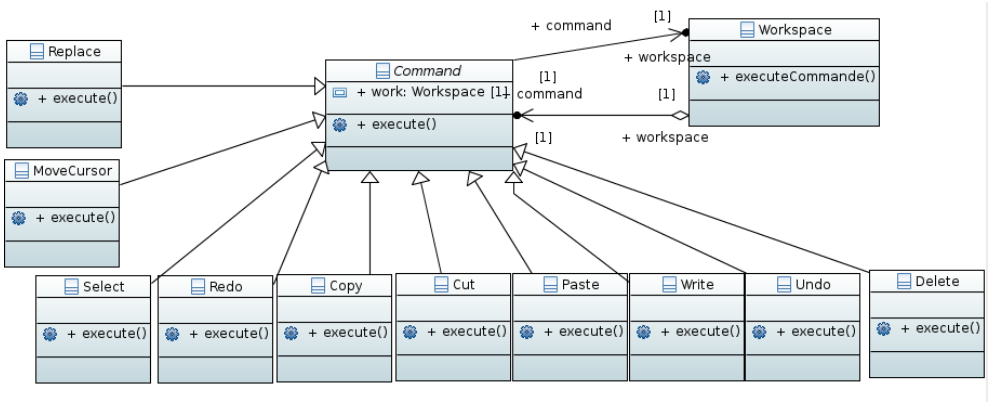

FIGURE 2.2 – Command pattern

2.2.2 Memento Pattern

Le patron de conception Memento permet de créer des objets permettant de sauvegarder et de restaurer l'état interne d'un programme à un moment donné. Notre éditeur de texte comportant des commandes *Undo* et *Redo* l'utilisation de ce pattern nous a grandement facilité la tâche pour la réalisation de ces fonctions. Voici le diagramme de ce patron de conception. [1]

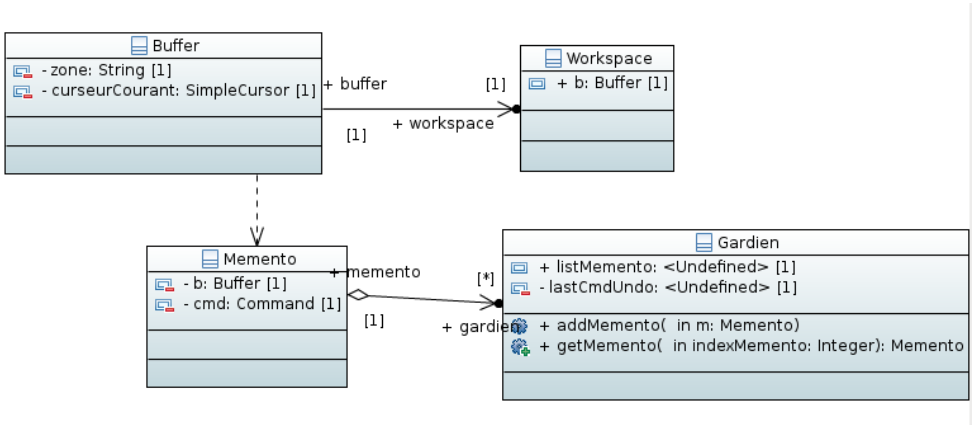

FIGURE 2.3 – Memento Pattern

1. Le type de Gardien.listMemento est une liste de memento, nous avons laissé le champ indéfini car nous n'avons pas trouvé comment faire un type ListBuffer dans Papyrus

2.2.3 Composite Pattern

Le pattern composite permet la création de structure contenant des objets avec des comportements communs. Dans notre cas, nous avons utilisé le pattern composite pour permettre a l'utilisateur de créer des macros. Grâce au composite, l'utilisateur pourra manipuler une liste de commande de la même façon qu'une commande. La seule différence sera que toute les commandes de la liste seront éxecutées lors de l'appel de la macro. [2]

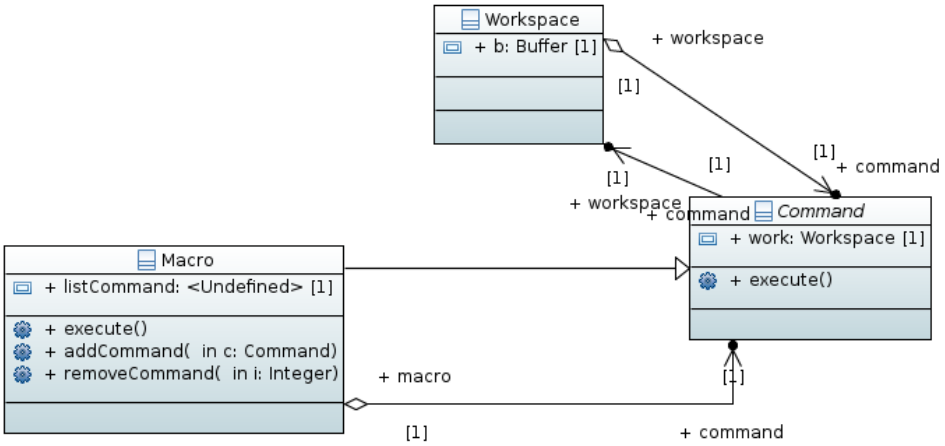

FIGURE 2.4 – Composite Pattern

2.2.4 Observer Pattern

Le pattern observer permet de créer un objet qui observera un autre objet et effectuera une opération lors du changement de celui-çi. Lorsque l'objet observé est modifié, les observateurs recoivent une notification et effectuent une action. Dans notre cas le pattern Observer s'applique sur le Workspace, quand ce dernier est modifié par une commande, l'observateur est notifié et appel un affichage du buffer [3].

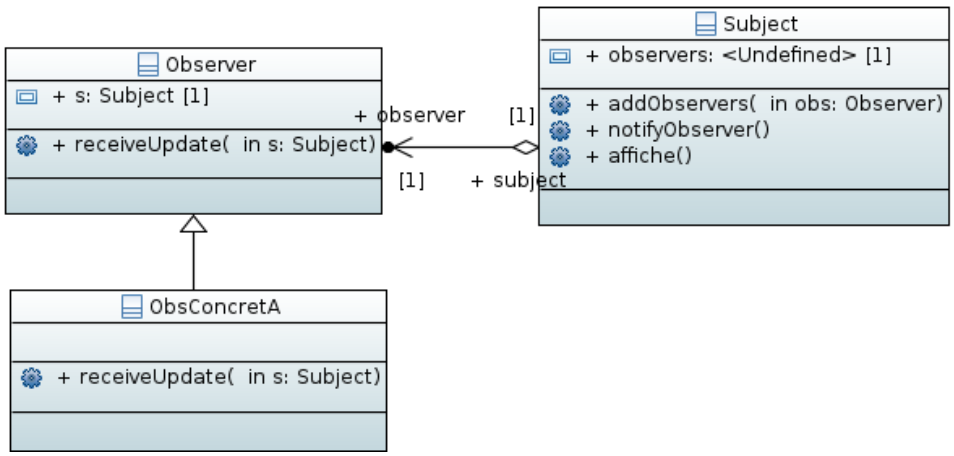

FIGURE 2.5 – Observer Pattern

2. Ici listCommand est du type ListBuffer<Command>
3. Ici observers est du type ListBuffer<Observer>

2.3 Diagramme de séquence

Nous allons présenter ici un diagramme de séquence représentant l'action utilisateur *Write*.
Pour appeler la commande Write, il faut tout d'abord créer un objet Command de type Write que l'on transmet au Workspace. Celui-çi ce charge de lancer l'éxecution de la commande en l'envoyant à la classe Write. Une fois le message de retour arrivé, le Workspace notifie les utilisateurs du changement qui a eu lieu. Le buffer récupere la notification avec ca méthode *receiveUpdate()* ce qui déclanche l'affichage.
L'exécution des autres commandes fonctionnant de la même manière que la commande *Write*, nous ne présenterons sous forme de diagramme de séquence que ce cas la.

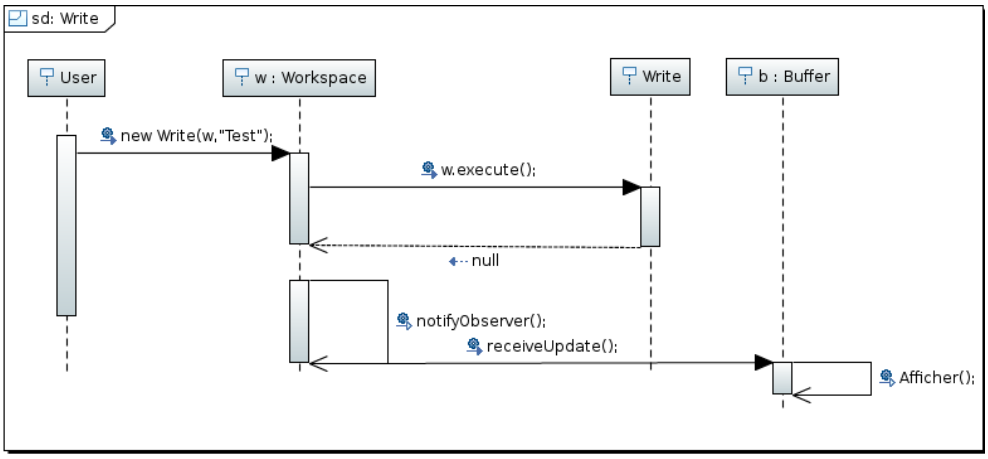

FIGURE 2.6 – Diagramme de sequence : Write

CHAPITRE 3

Developpement

Dans cette section nous présenterons les classes contenant l'application en expliquant rapidement leur fonction.

Buffer La classe Buffer représente la zone de texte sur laquelle travaille l'utilisateur.

ClipBoard Le ClipBoard est la classe ou l'on stocke la sélection après avoir copié ou coupé.

Command La classe abstraite Command est la classe mère des commandes concrètes.

Copy Copy hérite de Command et permet de copier une selection dans le ClipBoard.

Cursor Cursor est la classe mère de SimpleCursor et DoubleCursor.

Cut Cut hérite de Command et permet de couper une string selectionnée et de la mettre dans le ClipBoard.

Delete Delete hérite de Command et permet de supprimer une string selectionnée du Buffer.

DoubleCursor DoubleCursor hérite de Cursor et permet de délimiter à l'aide de deux entiers une châine de caractère à l'interieur du buffer.

Gardien Le Gardien est la classe chargée de garder en mémoire l'historique du Buffer ainsi que de renvoyer une version plus ancienne de ce dernier.

Macro La classe Macro hérite de Command. Elle fonctionne de la même manière que les autres commandes à la différence près qu'elle execute une liste de commande.

Memento La classe Memento permet de créer des historiques du Buffer qui seront stockés dans le Gardien.

MoveCursor MoveCursor hérite de Command et permet de déplacer le curseur courant afin d'insérer du texte ailleurs qu'à la fin du buffer.

ObsConcretA ObsConcretA hérite de Observer et permet d'afficher le buffer lors de la notification de modification de ce dernier.

Observer Classe mère de ObsAffiche.

Paste Paste hérite de Command et permet de coller le contenu du ClipBoard à la position du curseur courant dans le buffer.

Redo Redo hérite de Command et permet de remettre en place la dernière action annulée avec Undo.

Replace Replace hérite de Command et permet de remplacer la selection par le contenu du ClipBoard.

Selection La classe Selection contient la chaîne de caractère située entre les curseurs de DoubleCursor.

Selectionner Hérite de Command et permet de selectionner une partie du buffer.

SimpleCursor Hérite de Cursor et représente le curseur courant du buffer.

Subject Contient la liste des observeurs et permet de les notifier lors d'un changement du buffer.

Undo Undo hérite de Command et permet d'annuler la dernière action réalisée.

Workspace Workspace contient le buffer, le ClipBoard et est la classe ou sont appelées les commandes.

Write Write hérite de Command et permet de demander à l'utilisateur d'entrer une chaîne de caractère pour l'ajouter dans le buffer.

CHAPITRE 4

Bilan

4.1 Conclusion

Le langage de programmation Scala étant totalement nouveau pour nous, l'implémentation de l'application a pris plus de temps qu'il nous en aurait fallu avec un langage que nous connaissions. Il reste néanmoins avantageux d'avoir pu travailler sur un langage inconnu ne serait ce que pour la connaissance personnelle. En effet il est toujours interessant d'avoir des bases dans de nombreux domaine afin de ne pas se retrouver désamparer lors d'un travail sur un de ces domaines.
De même l'utilisation de Papyrus était totalement nouvelle, nous avions jusque la toujours utiliser des logiciels exterieurs à Eclipse qui n'était pas toujours conforme à la norme UML. Le fait que Papyrus soit directement lié à l'IDE permet de mieux lier la conception du projet avec son implementation.
Enfin l'utilisation de patron de conception pour la réalisation de certaines fonctionnalités nous ont grandement facilité la tâche. En plus de cela, la présence de patron de conception dans le code donne une capacité d'extensibilité à l'application.

4.2 Perspectives

Bien que respectant les consignes données en début de projet, de nombreuses améliorations de l'application reste possible.
Tout d'abord, avec une connaissance plus approfondie du langage Scala nous pourrions réecrire une grande partie du code dans un but d'optimisation et d'un meilleur respect de l'aspect Génie Logiciel de l'implémentation.
Un ajout d'une interface graphique au projet serait aussi une amélioration importante possible afin de rendre l'utilisation du mini éditeur de texte plus simple.
Enfin il reste de nombreuses améliorations que l'on peut retrouver dans les éditeurs de texte dit standarts qui pourraient être implémentés dans ce projet.
En conclusion, il reste de nombreuses pistes d'améliorations qui pourraient être suivies : au niveau des fonctionnalités, de l'implémentation ou de la conception

ANNEXE A

Annexe

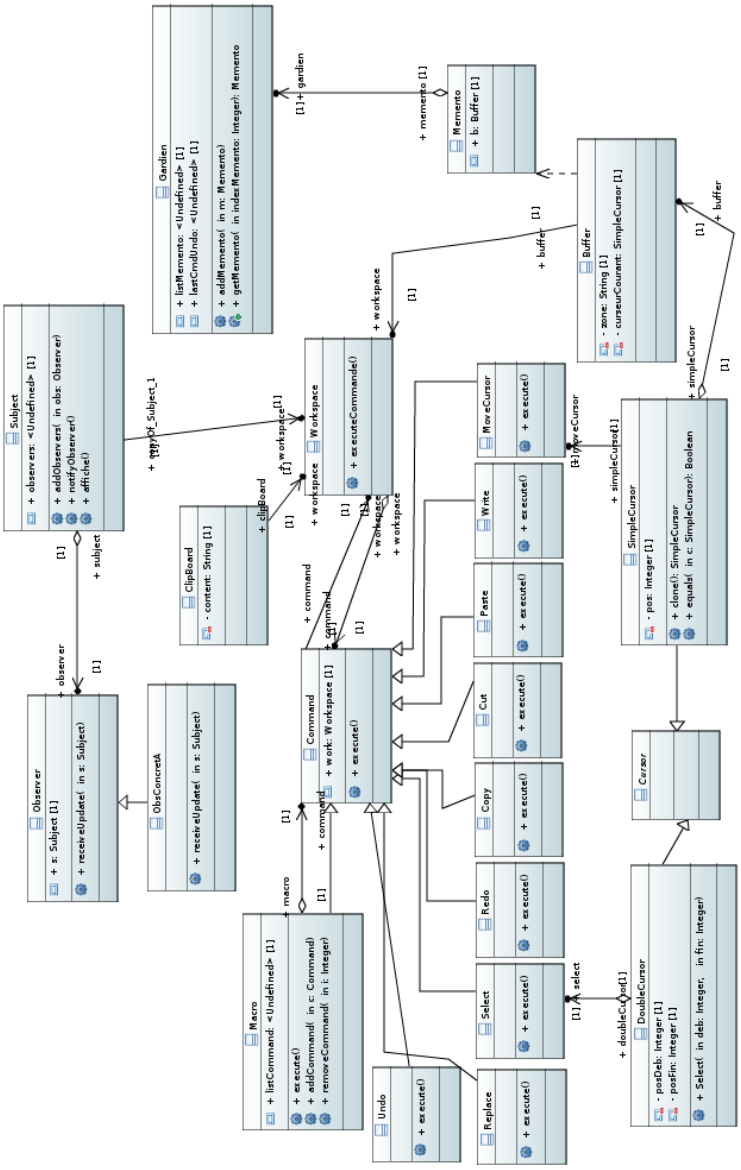

FIGURE A.1 – Diagramme UML de l'application