ACO : Compte-rendu quant au développement du Mini-Éditeur

Dorian Dumanget Corentin Chédotal

7 décembre 2017

1 Présentation du projet

1.1 Présentation générale

Le présent projet est un Mini-Éditeur de texte réalisé en Java. Il doit être possible dans ce Mini-Éditeur d'écrire, de copier, couper, coller, sélectionner du texte, enregistrer un ensemble de commande qui devra pouvoir être rejoué (les macros-commandes), ainsi que de pouvoir défaire les dernières actions effectuées et les refaire si besoin.

Le projet a été séparé en plusieurs versions afin de pouvoir réaliser et intégrer les différentes fonctions au fur et à mesure. Les différentes versions du projets sont présentés ci-dessous.

1.2 Version 1

La première version du projet devait permettre à l'utilisateur de pouvoir écrire du texte, copier, couper et coller du texte et sélectionner tout ou une partie du texte (pour pouvoir écrire par dessus ou copier par exemple). Il fallait pour cela mettre une place un moteur permettant d'effectuer les différentes actions demandées. De plus nous avons dû mettre en place une interface afin de pouvoir demander à l'utilisateur quelles commandes il souhaite effectuer.

1.3 Version 2

Dans la deuxième version du projet, il devait être possible de pouvoir enregistrer une suite de commande en plus des fonctionnalités de la version 1. Cette suite de commande devait pouvoir être enregistrée et l'enregistrement doit pouvoir être arrêté sur demande de l'utilisateur, afin de pouvoir relancer exactement les commandes souhaitées. Nous avons alors dû mettre en place un enregistreur capable de stocker une suite de commande pouvant ensuite être relancées sans passer par l'intermédiaire de l'IHM pour obtenir les paramètres des différentes fonctions.

1.4 Version 3

Dans cette troisième et dernière version du projet, il devait être possible de défaire les dernières commandes effectuées et de refaire les commandes si elles avaient été défaite. Pour cela, nous avons créé un gestionnaire capable de restauré le texte ainsi que la sélection mais sans changer le contenu du presse-papier.

1.5 État actuel

Le rendu du projet a atteint comme attendu le spécification de la version 3 présentée ci-dessus et de ses versions précédentes. Elle vient complète avec documentation *Doxygen* et, comme demandé, tests unitaires *JUnit 5*. Sont rendues avec ce rapport les trois versions ensembles mais chacune d'entre elle est aussi disponible séparément dans des *releases* sur le repertoire *GitHub* public du projet. Le projet dans son intégralité est mis à disposition avec une licence libre, la LPRAB 1.0.

2 Architecture du projet

2.1 Les différents Patron de Conception

2.1.1 Command

Afin de séparer totalement la partie affichage de la partie calcul, nous avons utilisés le PC `Command`.

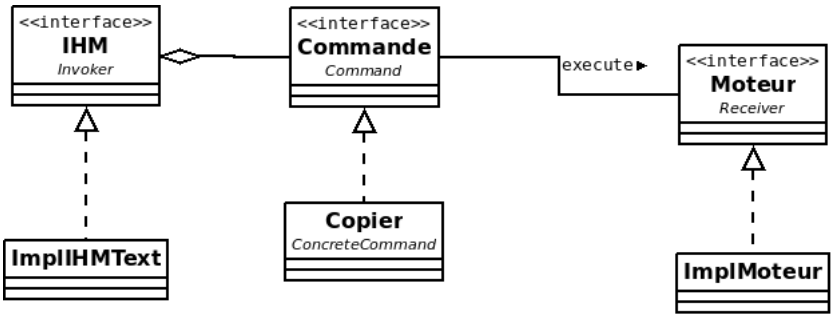

FIGURE 1 – Un exemple du *pattern* `Command`

Ce PC est utilisé a plusieurs moments dans le projet. Il est tout d'abord utilisé dans la version 1 du Mini-Éditeur afin de séparer l'IHM (partie affichage), du moteur (partie calcul). Nous avons donc l'IHM qui joue le rôle de l'`Invoker` et le moteur qui joue le rôle du `Receiver`.
Il est ensuite utilisé dans la version 2 du Mini-Éditeur afin de séparer l'IHM de l'enregistreur. Nous avons donc l'IHM qui joue à nouveau le rôle de l'`Invoker` et l'enregistreur qui joue le rôle du `Receiver`.
Et enfin il est également utilisé dans la version 3 du Mini-Éditeur afin de séparer l'IHM du gestionnaire D/R. Nous avons donc l'IHM qui joue encore le rôle de l'`Invoker` et le gestionnaire qui joue le rôle du `Receiver`.

Les classes qui implémentent le rôle des `ConcreteCommands` sont les fonctions de notre Mini-Éditeur (Couper, Copier, Coller ...).

L'avantage de l'utilisation de ce PC, est la modularité des classes concrètes. En effet, à partir du moment où les classes concrètes implémentent les interfaces données, il sera toujours possible de changer la classe concrète.

2.1.2 Memento

Afin de pouvoir enregistrer des informations, pour les réutiliser après, nous avons utilisé le PC `Memento`.

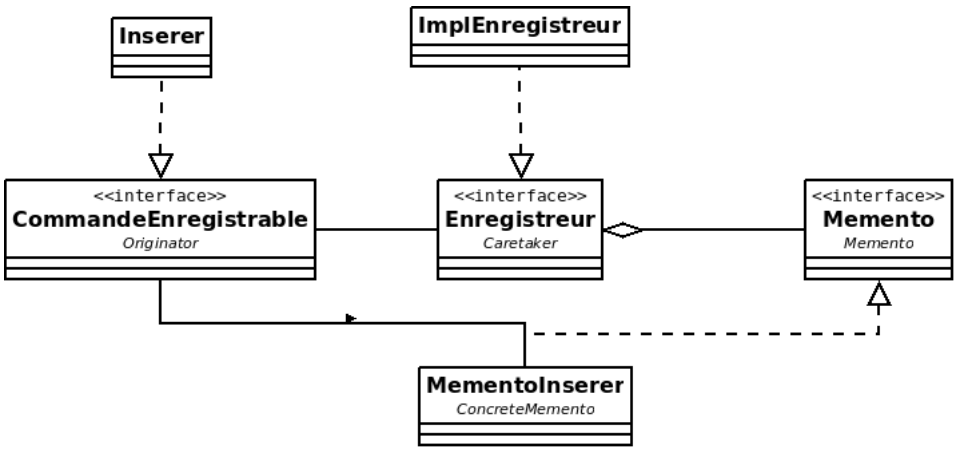

FIGURE 2 – Un exemple du *pattern* `Memento`

Ce PC est utilisé lors de la version 2 du Mini-Éditeur afin de pouvoir sauvegarder l'état des fonctions, notamment `Insérer` et `Sélectionner` qui demandent normalement des paramètres à l'IHM. Nous avons alors l'enregistreur qui joue le rôle du `Caretaker`, les commandes qui jouent le rôle de `Originator` et de nouvelles classes (MementoInserer et MementoSelectionner) sont alors créées pour jouer le rôle de `Memento`.

2.2 Nos choix d'implémentation

2.2.1 L'interface Homme-Machine (IHM)

Concernant l'interface, nous avons décidés de réaliser une IHM minimaliste permettant d'effectuer toutes les actions demandées. Cette interface est uniquement textuelle, et fonctionne en écrivant les commandes à réaliser. Notre interface est sensible à la casse, ce qui pourrait être amélioré dans de nouvelles versions.

Au niveau de l'implémentation, nous avons stockés les différentes fonctions exécutables dans une Map dont la clé est une String permettant de nommer les fonctions en utilisant le nom que l'on souhaite. Cette implémentation a tout d'abord un aspect pratique, puisque le nom des fonctions peut être afficher facilement et l'utilisateur sait exactement quoi entrer dans le terminal afin d'effectuer une commande.

2.2.2 Le moteur

Pour la partie moteur, nous avons réalisé un moteur capable d'effectuer les actions de base attendu pour un éditeur de texte standard (Copier, coller, couper, sélectionner et inserer du texte).
Concernant l'implémentation nous avons choisi d'utiliser des classes extérieurs au moteur afin de stocker le contenu du texte, la sélection sur ce texte ainsi qu'un presse-papier. Nous avons alors 3 classes supplémentaires permettant de garder toutes ces informations. Le rôle du moteur est alors de modifier ces classes afin qu'elles contiennent effectivement ce qui est souhaité par l'utilisateur.

2.2.3 La commande Affiche

Afin de pouvoir afficher directement le texte contenu dans le Buffer présent dans le moteur mais tout en gardant une séparation entre le moteur et l'IHM, nous avons ajouté une classe supplémentaire suivant le principe du patron de conception `Command`, la classe `Affiche` qui interroge le moteur afin de connaître le texte contenu dans le Buffer ainsi que la position de la sélection.

2.2.4 L'enregistreur

L'enregistreur est la partie qui s'occupe de sauvegarder un ensemble de commandes afin de pouvoir les ré-exécuter ultérieurement. Pour cela nous avons utiliser une classe interne spécifique, la `Paire`. Cette `Paire` contient 2 éléments indissociables : la `Commande` à ré-exécuter ainsi que le `Memento` correspondant à cette commande. Ainsi, il nous suffit de sauvegarder un ensemble de `Paire` afin de pouvoir retrouver facilement la correspondance entre chaque `Commande` et le `Memento` qui lui est associé au moment de l'exécution.

2.2.5 Le gestionnaire D/R

Le dernier point important de notre implémentaion est le `gestionnaire Defaire/Refaire` qui permet de revenir à un état du `Buffer` et de la `Selection` précédemment enregistrer. Pour cela nous avons utiliser une classe interne les `Etat` qui contiennent un `Buffer` et une `Selection`. On utilise alors une fonction du moteur qui permet de charger rapidement un `Buffer` et une `Selection` pour modifier l'état courant.
Pour garder en mémoire ces différents états nous utilisons 2 piles différentes (une pour défaire l'autre pour refaire) ainsi qu'un état "courant".

3 Évolutions futures

3.1 Idées

3.1.1 Permettre plusieurs macros

Une bonne évolution du projet serait d'éventuellement permettre d'avoir plus d'une macro pour l'utilisateur. Il faudrait pour cela refaire une implémentation de l'interface `Enregistreur`. Celle-ci pourrait par exemple avoir un système de Hash avec chaque macro ayant un nom. Nom qui serait demandé à l'utilisateur par l'IHM après qu'il est tapé les commandes `Demarrer`, `Stopper` ou `Rejouer`.

Cependant cette idée sortant du cadre de la spécification de la version 3 elle ne fut pas creusée durant le temps de développement.

3.1.2 Extension d'`Enregistrable`

Il n'apparaît pas idiot de faire en sorte que les classes `Buffer`, `PressePapier` et `Selection` implémente l'interface `Enregistrable`.

En effet, en plus d'avoir un sens sémantiquement ces classes sont copiées par notre implémentation du Gestionnaire D/R. Utiliser un système basé une fois de plus sur le Memento pourrait être plus pratique et plus malléable que notre implémentation actuelle à base d'états.

Cependant nous nous sommes rendu-compte de cette possibilité un peu tard dans le développement et un *refactoring* du code ne semblait plus approprié à ce moment-là.

3.1.3 Nouvelle IHM

Bien que la version actuelle de l'IHM mise à disposition de l'utilisateur avec la version actuelle du projet est fonctionnelle elle n'en reste pas moins textuelle.

De plus, comme présenté plus haut elle est soumise à la casse. Plutôt que de faire de petites améliorations du format textuel nous aurions aimés pouvoir intégrer une deuxième IHM. Celle-ci n'aurait pas été textuelle mais belle et bien graphique. Plusieurs pistes on été rapidement examinés à mi-développement comme la librairie *Swing* ou encore les générateurs de fenêtres et GUI d'*Eclipse* ou *IntelliJ*. Cependant nous avons rapidement déterminé, et à raison, que nous n'aurions pas le temps de la développer en plus des autres projets du semestre.

Elle n'en reste pas moins une idée de développement futur du projet, idée qui mettrait bien en oeuvre les avantages des *design patterns* employés puisque les IHM sont interchangeables.

3.2 Conclusion

La réalisation de ce projet a permis de mettre en oeuvre des patrons de conceptions nouveaux et à travailler plus en détail dans une architecture purement orientée objet. Son développement a nécessité une plus grande attention au fonctionnement global de l'application et à l'interaction entre les classes plutôt que de s'intéresser au code pur de celles-ci. De plus l'envergure du projet (en termes de nombre de fichiers et de versions différentes) a permis la mise en place et l'emploi courant des fonctionnalités de contrôle de version et de branchement de *git*. A cela s'ajoutait la demande de réalisation de documentation qui a rappelé que cela reste une charge de travail conséquente mais nécessaire. Particulièrement avec les différentes versions de l'éditeur.

A Annexes

Note : Les diagrammes UML sont aussi disponibles dans l'archive contenant le code.

Table des matières

Table des figures

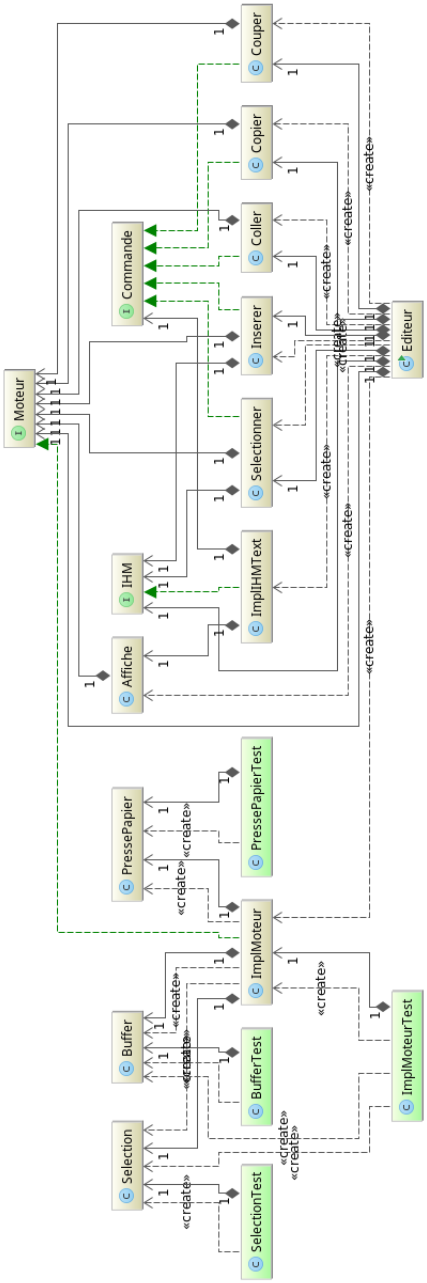

FIGURE 3 – Diagramme UML de la version 1 du Mini-Éditeur

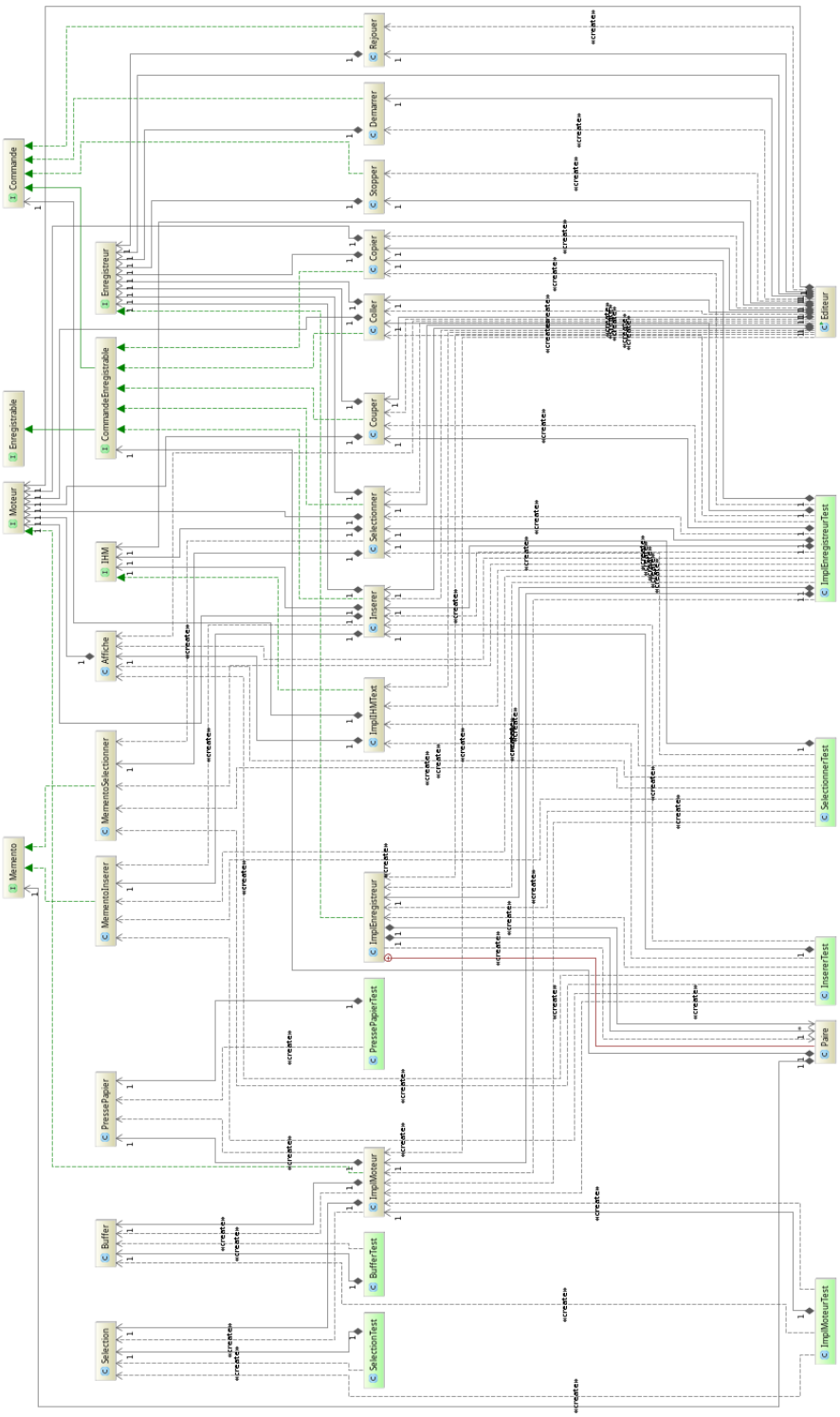

FIGURE 4 – Diagramme UML de la version 2 du Mini-Éditeur

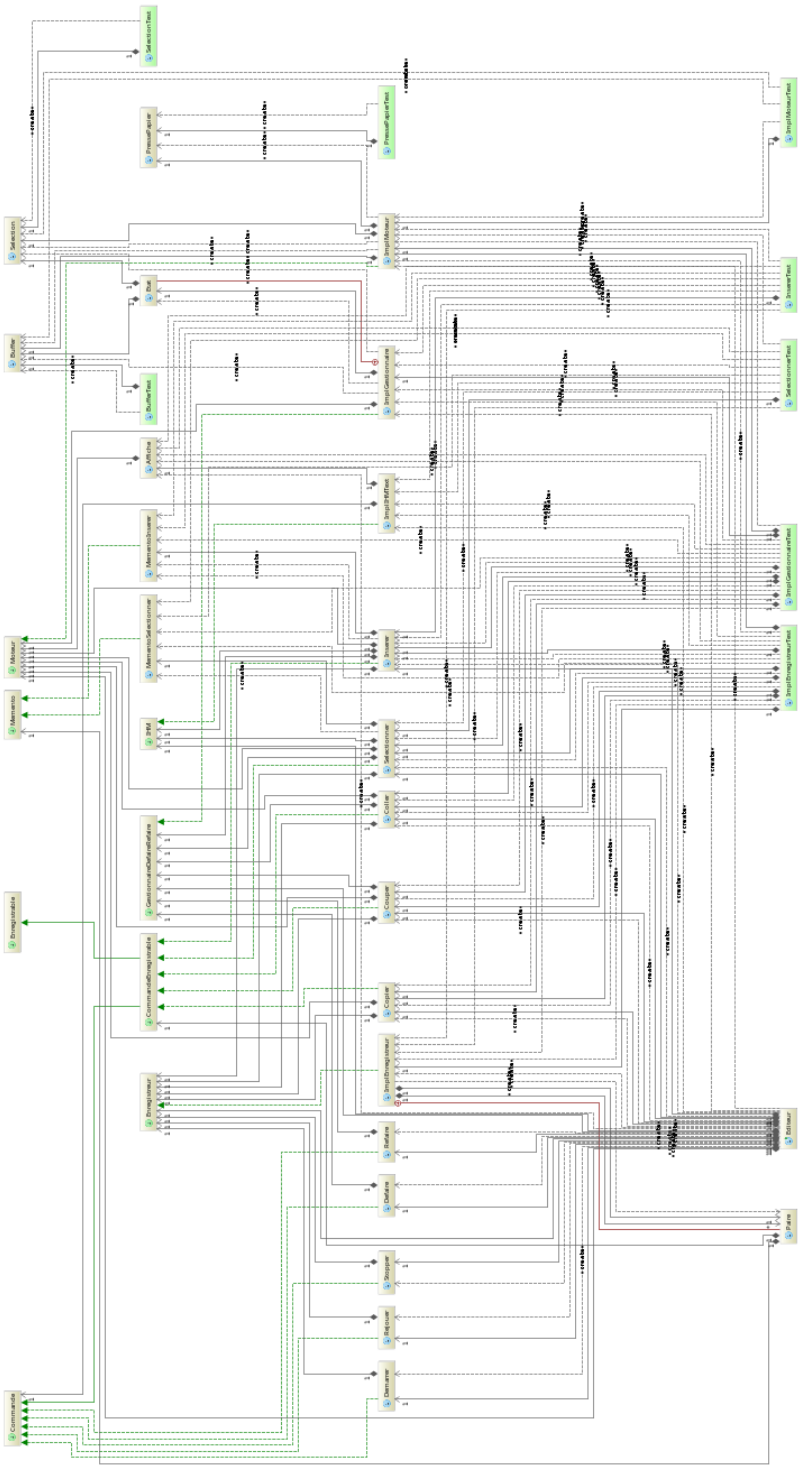

FIGURE 5 – Diagramme UML de la version 3 du Mini-Éditeur