# Rapport du Projet d'Analyse et Conception Objet: le MiniEditor

Vincente NATTA & Salomé COAVOUX

9 décembre 2013

# Table des matières

# 1 Stratégie de tests

Pour implémenter correctement notre mini éditeur, nous avons utilisé les tests unitaires avec *JUnit*. Comme demandé, nous avons tout d'abord écrit le squelette de notre code, puis les tests avant de se lancer dans l'implémentation proprement dite. Cela nous a beaucoup aidé, notamment pour la fonction *select*. En effet, cette fonction est la base de toutes les autres, puisqu'elle est appelée avant chaque action pour déterminer la sélection courante. Une batterie assez fournie de tests a donc été écrite afin de ne rien laisser au hasard.

Le simple test d'une sélection plus grande que le contenu du buffer en est un très bon exemple. Cela n'est jamais censé se produire, l'IHM ne devant permettre de sélectionner au maximum que la totalité du buffer. Cependant, le *KeyListener* existant n'ayant pas été enlevé, il est tout à fait possible d'ajouter du texte dans l'IHM grâce aux raccourcis clavier (*ctrl + v* pour coller, par exemple) sans que cela ne se répercute sur notre buffer. La taille de la sélection de l'intégralité du texte affiché devient alors plus grande que la taille effective du buffer.

# 2 Version 1

Lors de la démonstration en TP, la première version était terminée, et la deuxième en cours de développement, mais l'IHM n'étant pas encore au point, nous n'avions pas pu montrer notre travail. Le problème a été réglé depuis.

En plus des tests unitaires, nous avons à la main testé toute une série de suites de commandes pour nous assurer du bon fonctionnement de notre éditeur et de l'interface. Nous avons de plus ajouté des traces d'exécutions dans le code d'*EditorEngine* pour vérifier que le résultat affiché dans l'interface et celui de notre *Buffer* correspondaient bien à la même chose. Ces traces d'exécution ont été laissées dans les trois versions afin d'en montrer le bon fonctionnement.

En raison de nos problèmes de *KeyListener*, nous n'avons pas voulu ajouter à notre interface les raccourcis clavier, qui étaient pourtant simples à développer. Ainsi, utiliser *ctrl + v* sans avoir auparavant utilisé *ctrl + c* à l'intérieur de la zone de texte pouvait copier du texte venant d'une autre application, et qui ne se répercutait pas sur notre *Buffer*. Il aurait été complexe d'implémenter correctement ces raccourcis, pour une fonctionnalité qui nous paraissait dispensable, aussi avons nous préféré nous pencher sur d'autres problèmes plus importants. C'est pourquoi aucun raccourci clavier n'est utilisable dans notre éditeur.

## 3 Version 2

Pour implémenter la deuxième version, nous avons ajouté à la première la possibilité d'enregistrer et de rejouer des commandes. Cela se caractérise par l'ajout d'une nouvelle classe *Record* liée à l'*EditorEngine*. Cette classe contient notamment un booléen déterminant s'il y a ou non un enregistrement en cours, une valeur de sélection au début de l'enregistrement, et trois tableaux dynamiques de même taille contenant une sélection, un identifiant de commande et une éventuelle chaîne de caractères. A chaque commande enregistrée, on ajoute un élément dans chacun de ces tableaux.

La sélection correspond à la sélection courante lors de l'exécution de la commande. Celle-ci sera utilisée lorsque l'on rejouera les commandes, en y ajoutant la translation de la sélection au début de l'enregistrement. Les identifiants sont symbolisés par des entiers. chaque commande a son identifiant propre, et lorsque l'on rejoue, on utilise un *switch case* sur ces identifiants afin de déterminer l'action à rejouer. Enfin, la chaîne de caractères n'est utile que pour la commande *insert*, qui doit prendre une *String* en paramètre.

Cette version nous a posé problème lorsqu'il a fallut lier nos nouvelles commandes à l'interface graphique. En effet, bien que les commandes *enregistrer* et *arrêter* n'influencent pas le contenu du *Buffer*, la commande *rejouer*, en revanche, change sa valeur, et doit donc actualiser l'affichage de l'IHM. Ce problème a été réglé par l'ajout d'un *Controler*. Ainsi, cette nouvelle classe est dotée de la zone de texte de l'IHM et du *Buffer* de l'*EditorEngine*, et permet d'exécuter la commande qui va modifier le *Buffer*, puis de répercuter ces changements sur la zone de texte.

## 4 Version 3

Cette dernière version a été la plus simple des trois à implémenter. En effet, la majorité du code était déjà écrite, et les commandes *défaire* et *refaire* se basaient sur le même principe qu'*enregistrer*, à la différence que seul l'état du *Buffer* devait être conservé. Nous avons donc utilisé un *memento pattern*. Une nouvelle classe *Caretaker* s'occupe de sauvegarder les différents états du *Buffer* qui lui sont donné par l'*Observer*, c'est à dire *Buffer*.

Une fois cela mis en place, les commandes *undo* et *redo* ont été très faciles à développer. On ajoute à l'*EditorEngine* deux entier, *index*, qui représente la position courante dans le *memento*, et *indexMax*, qui représente la position maximale. Ceci est utile lors de l'enchaînement d'un ou plusieurs *undo*, puis d'une autre commande autre que *redo*. Dans ce cas là, il n'est pas logique de pouvoir refaire les commandes défaites, puisque l'on a ajouté autre chose. l'*indexMax* se ramène donc à l'*index*, et les commandes suivantes écrasent dans le *memento* les commandes que l'on ne pourra plus refaire.

Pour défaire ou refaire une action, il suffit alors de récupérer dans le *memento* l'état du *Buffer* qui nous intéresse, ou de l'ajouter, le tout à l'indice *index*.

# 5 Annexes

## 5.1 Diagrammes de classes

Pour plus de lisibilité, les diagrammes de classes ne montrent pas le contenu des classes. Celui-ci est donné après le diagramme de chaque version. Les classes identiques d'une version sur l'autre ne sont pas redonnées.

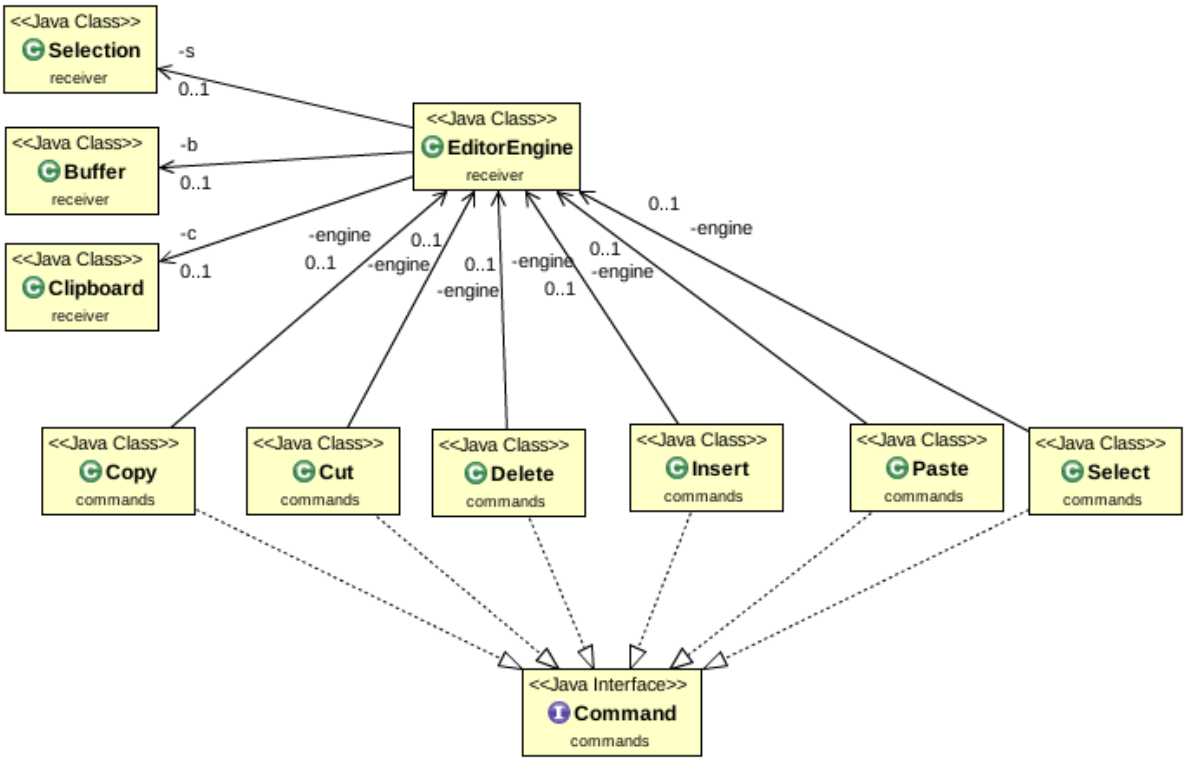


FIGURE 1 – Diagramme de classes de la version 1

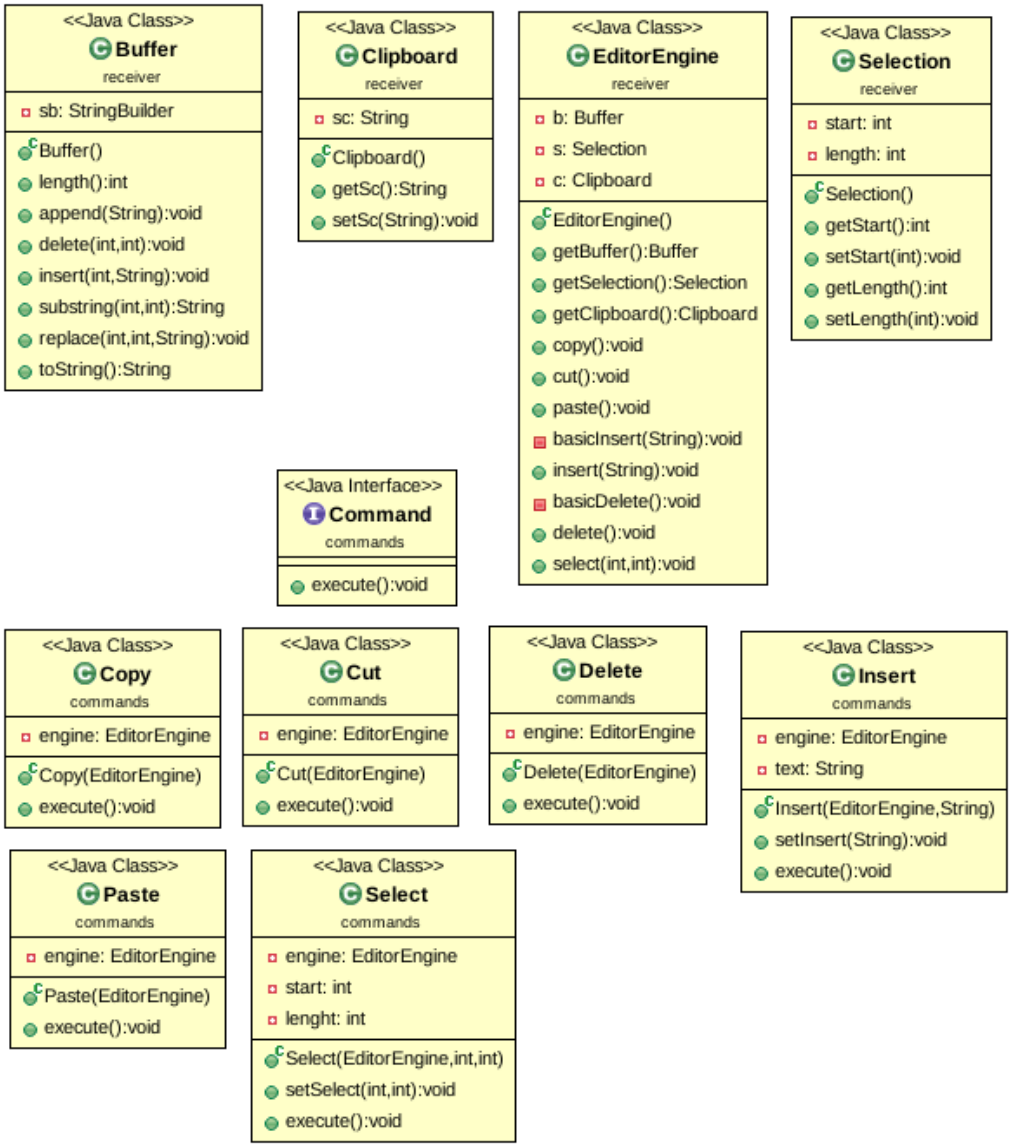


FIGURE 2 – Contenu des classes de la version 1

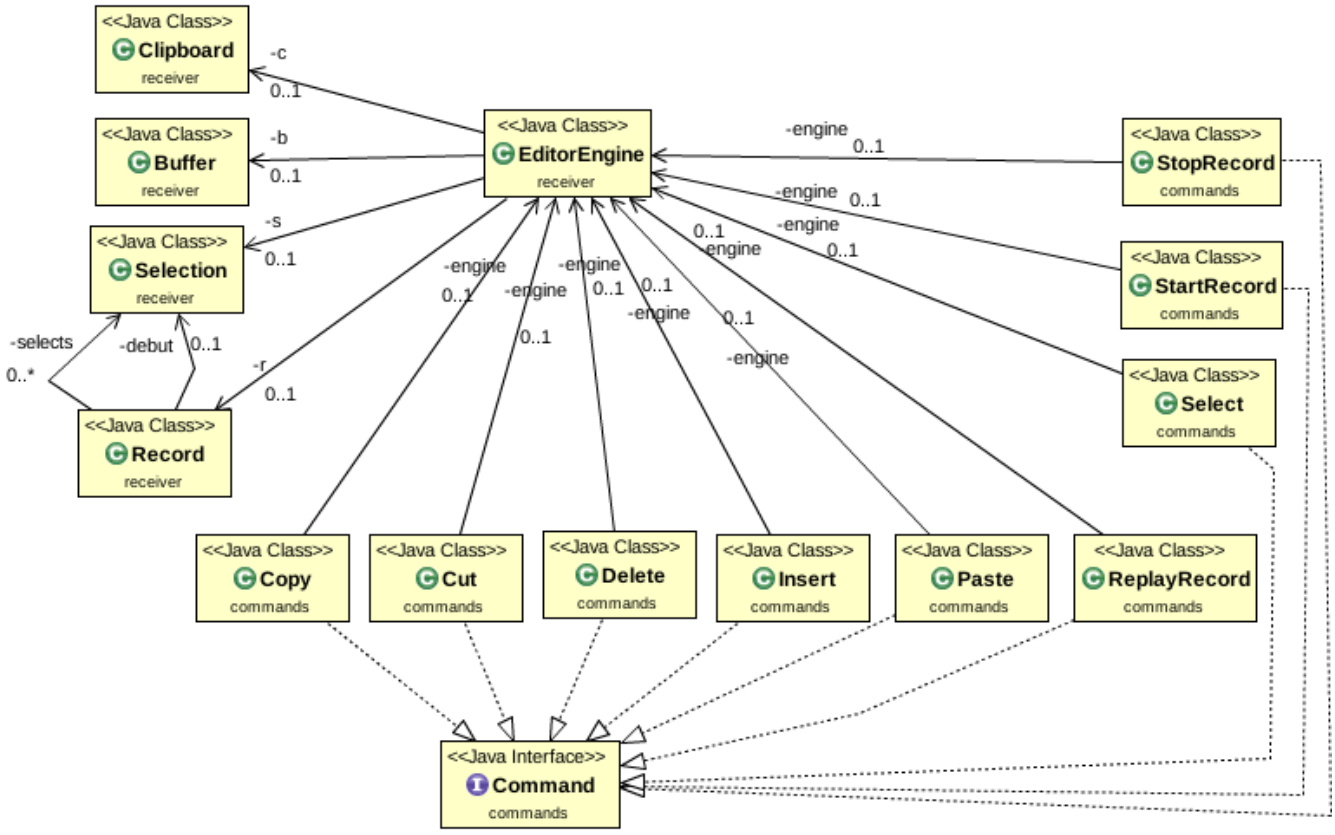


FIGURE 3 – Diagramme de classes de la version 2

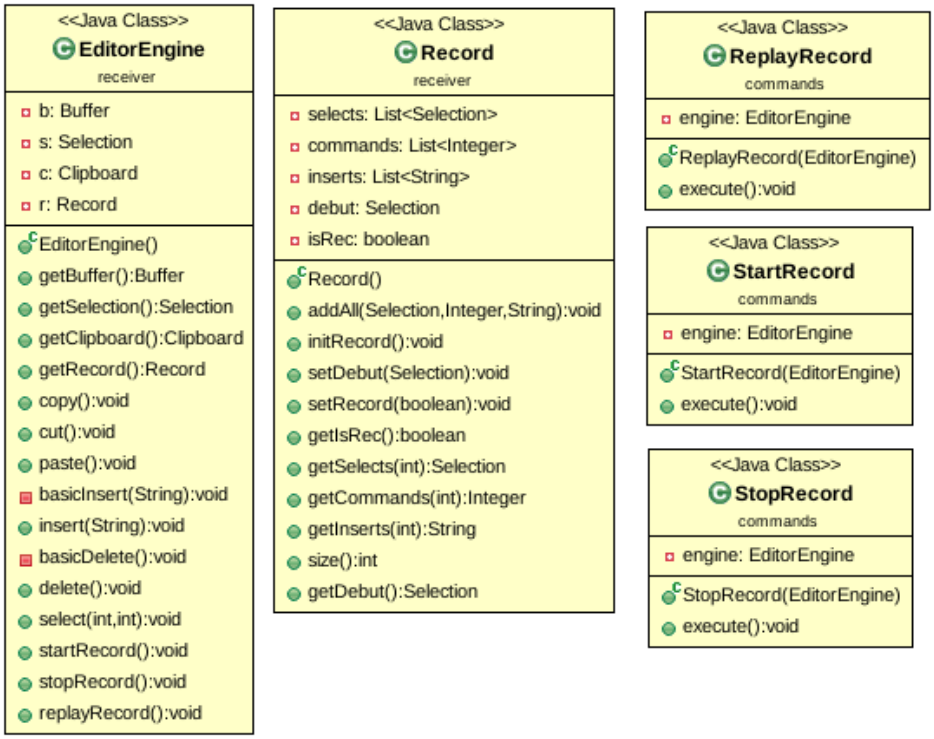


FIGURE 4 – Contenu des classes de la version 2

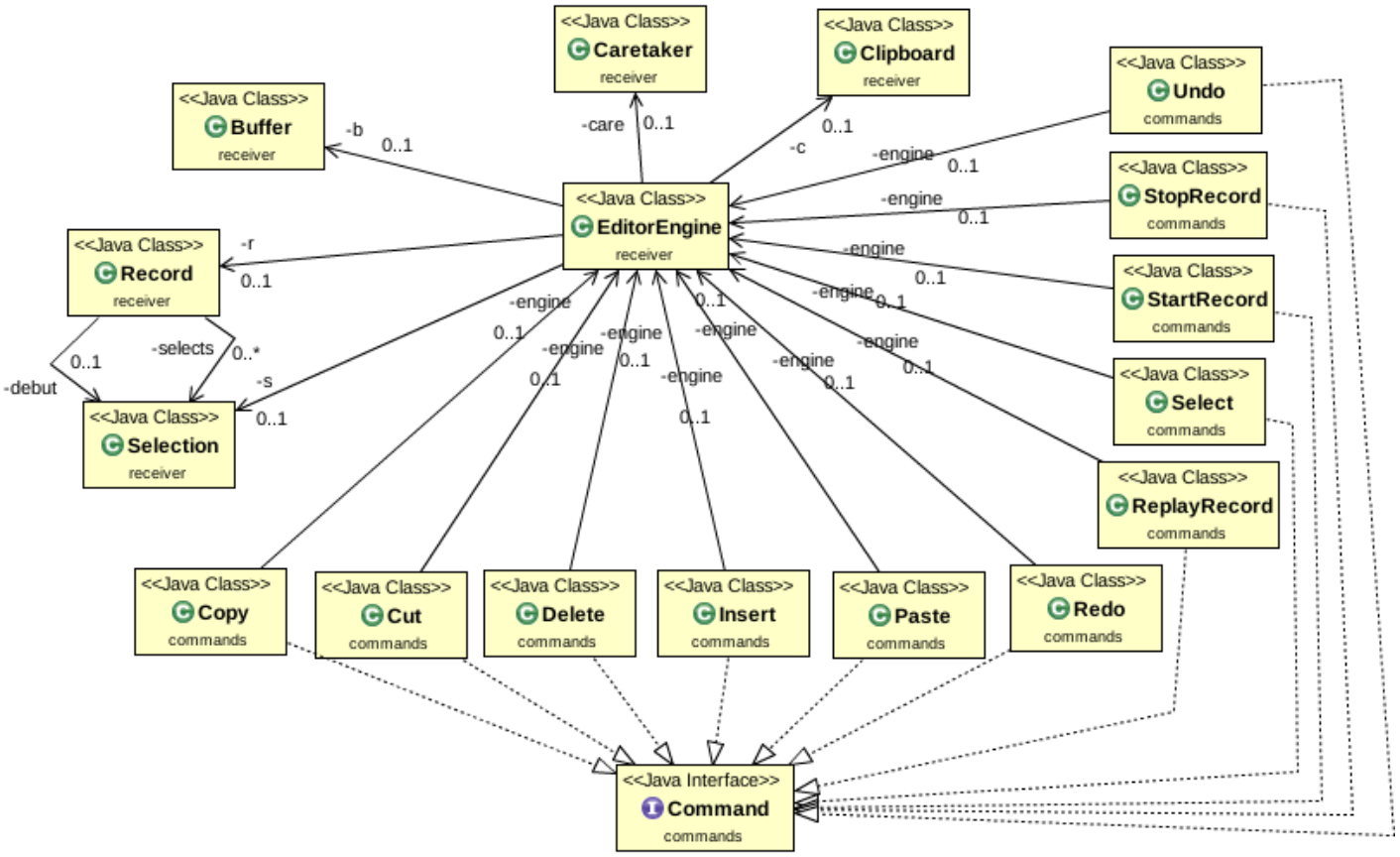


FIGURE 5 – Diagramme de classes de la version 3

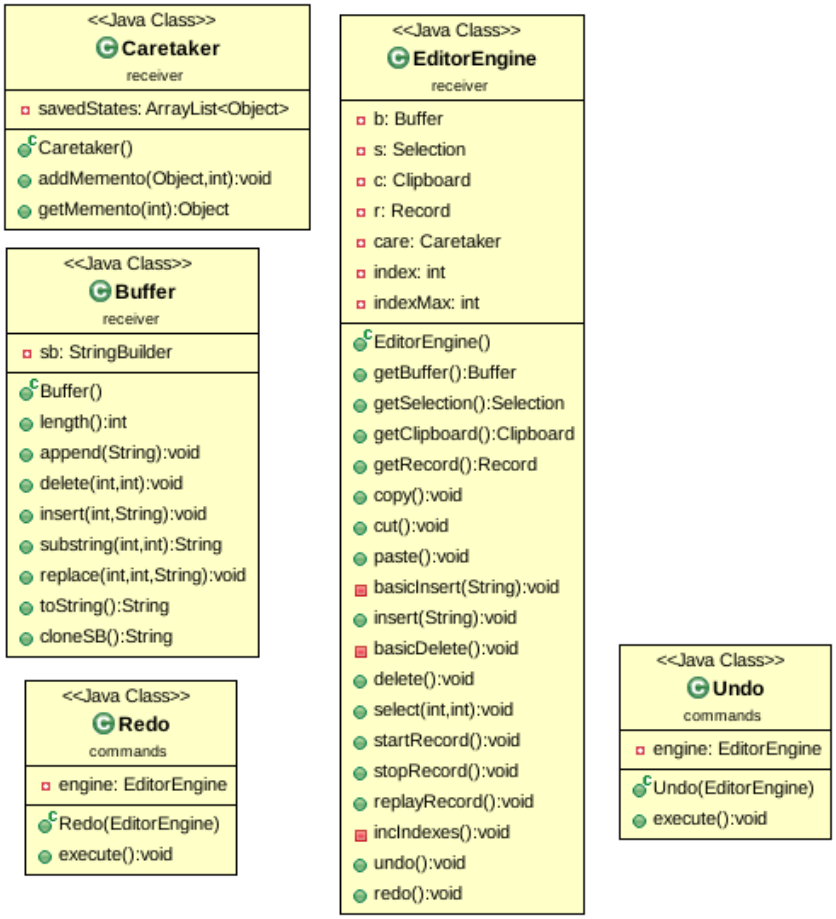


FIGURE 6 – Contenu des classes de la version 3

## 5.2 Diagrammes de séquence

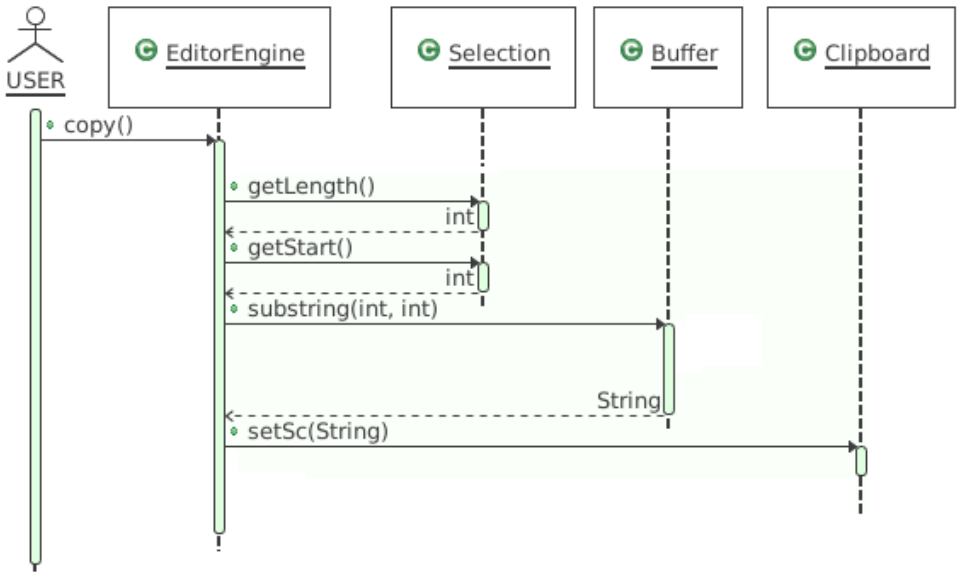


FIGURE 7 – Diagramme de séquence de la commande *Copier*

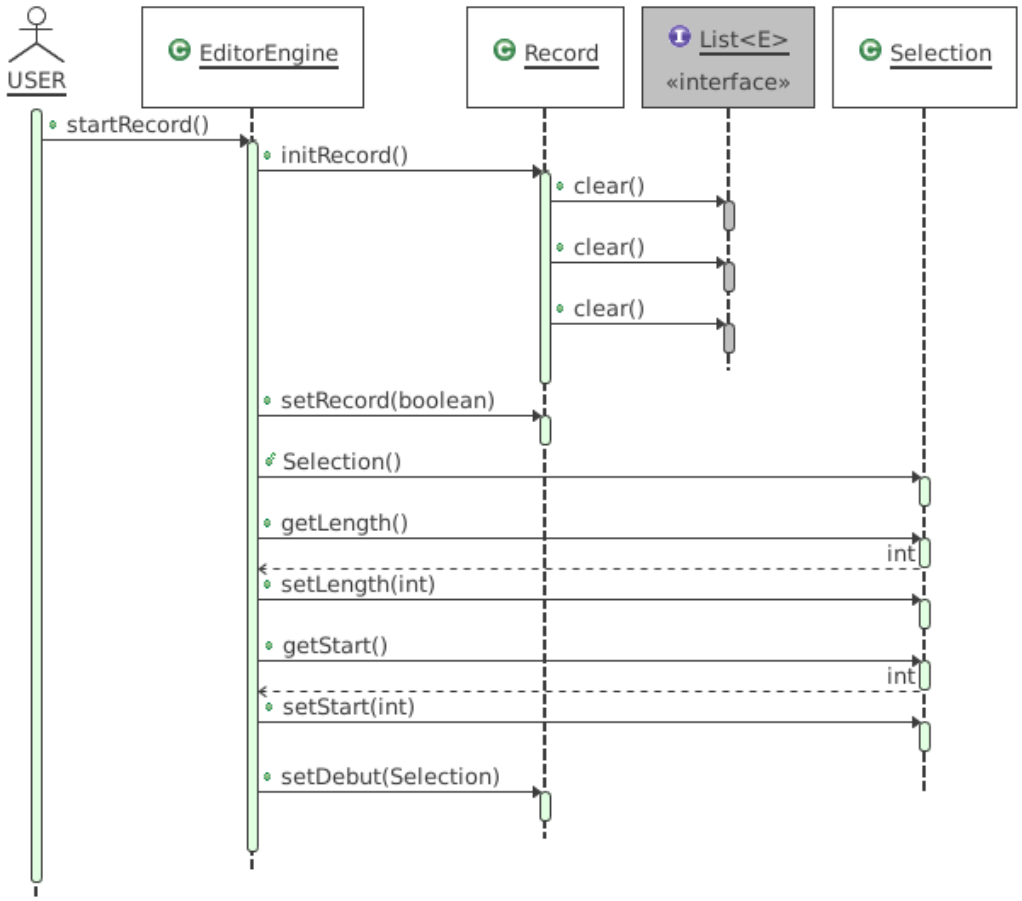


FIGURE 8 – Diagramme de séquence de la commande *Enregistrer*

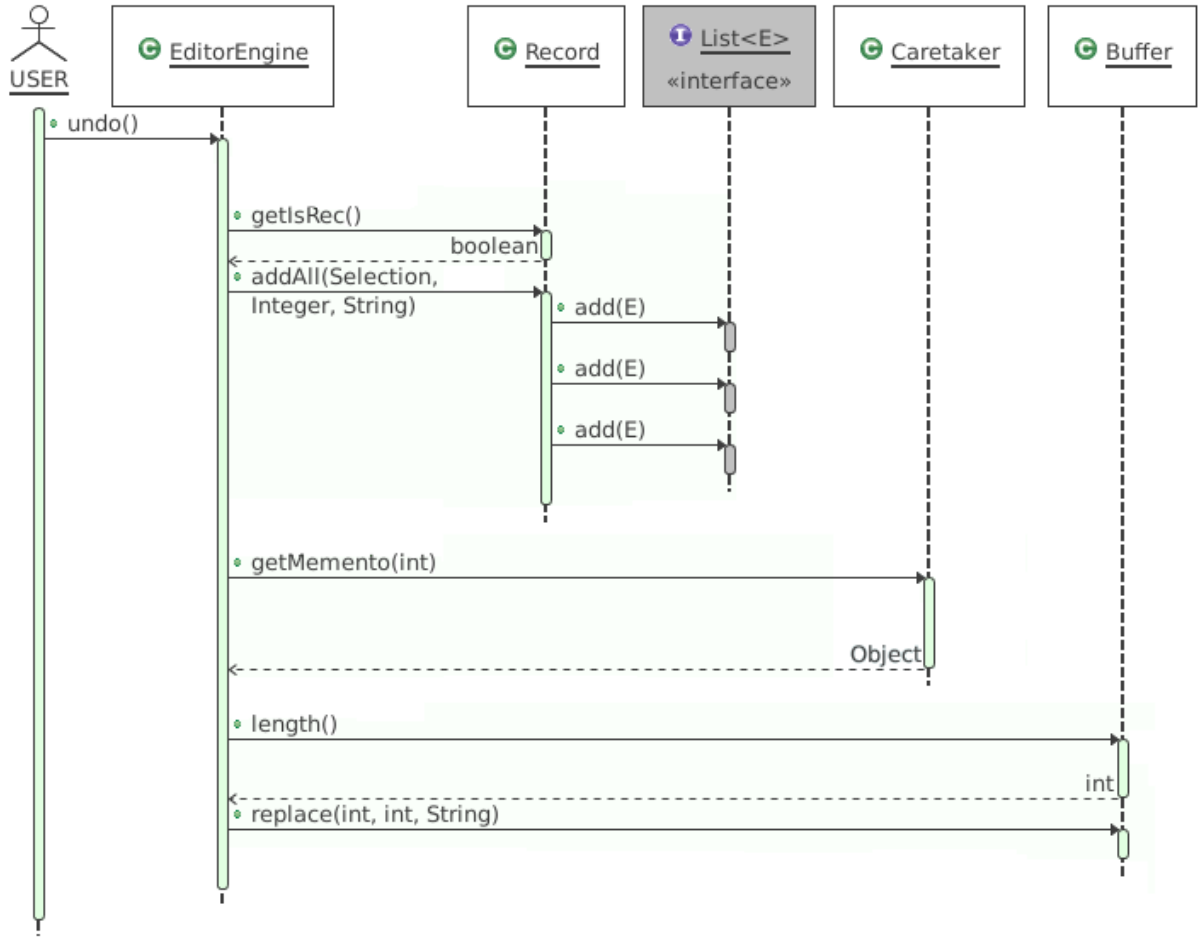


FIGURE 9 – Diagramme de séquence de la commande *Défaire*