

Rapport projet ACO

Université de Rennes 1
ISTIC - Master 1 MIAGE

2021-2022

Binôme
Eljeddi Yosser 1 A
Bah Moussa 1 B

Enseignant
Adrien Le Roch

Sommaire

I. Présentation de la première version du Mini Éditeur V1

1. Présentation de l'Interface de programmation (API)
2. Présentation de la première version V1
 - 2.1 Diagramme de classe de la V1
 - 2.2 Diagramme de cas d'utilisation de la V1

II. Présentation de la deuxième version du Mini Éditeur V2

1. Présentation de l'Interface de programmation (API) adapté pour V2
2. Présentation de la deuxième version V2
 - 2.1 Diagramme de séquence « Insert » de la V2
 - 2.2 Diagramme de classe de la V2
 - 2.3 Rapport des tests

III. Méthode d'exécution

IV. Conclusion

I. Présentation de la première version du Mini Éditeur V1

L'objectif de ce projet était de développer une application basée sur les fonctionnalités de base d'un mini éditeur de texte. Cette application devait offrir la possibilité à ses utilisateurs de faire une multitude d'actions du style Ecrire du texte, faire des manipulations diverses et variées sur ce texte tel que la suppression d'un élément du texte voulu par le l'utilisateur, la possibilité de copier une partie du texte en vue de la dupliquer par la suite ou encore la possibilité d'enregistrer certaines actions à un temps donné pour les rejouer par la suite, ...

Afin de pouvoir atteindre cet objectif et répondre aux contraintes imposées, le projet a été scindé en plusieurs parties qu'on va essayer de vous présenter à travers ce rapport.

Pour commencer, on vous invite à nous intéresser à la présentation de la V1

1. Présentation de l'Interface de programmation (API)

L'objectif de projet est de développer une application basée sur un mini éditeur de texte qui contient des différentes fonctionnalités de bases :

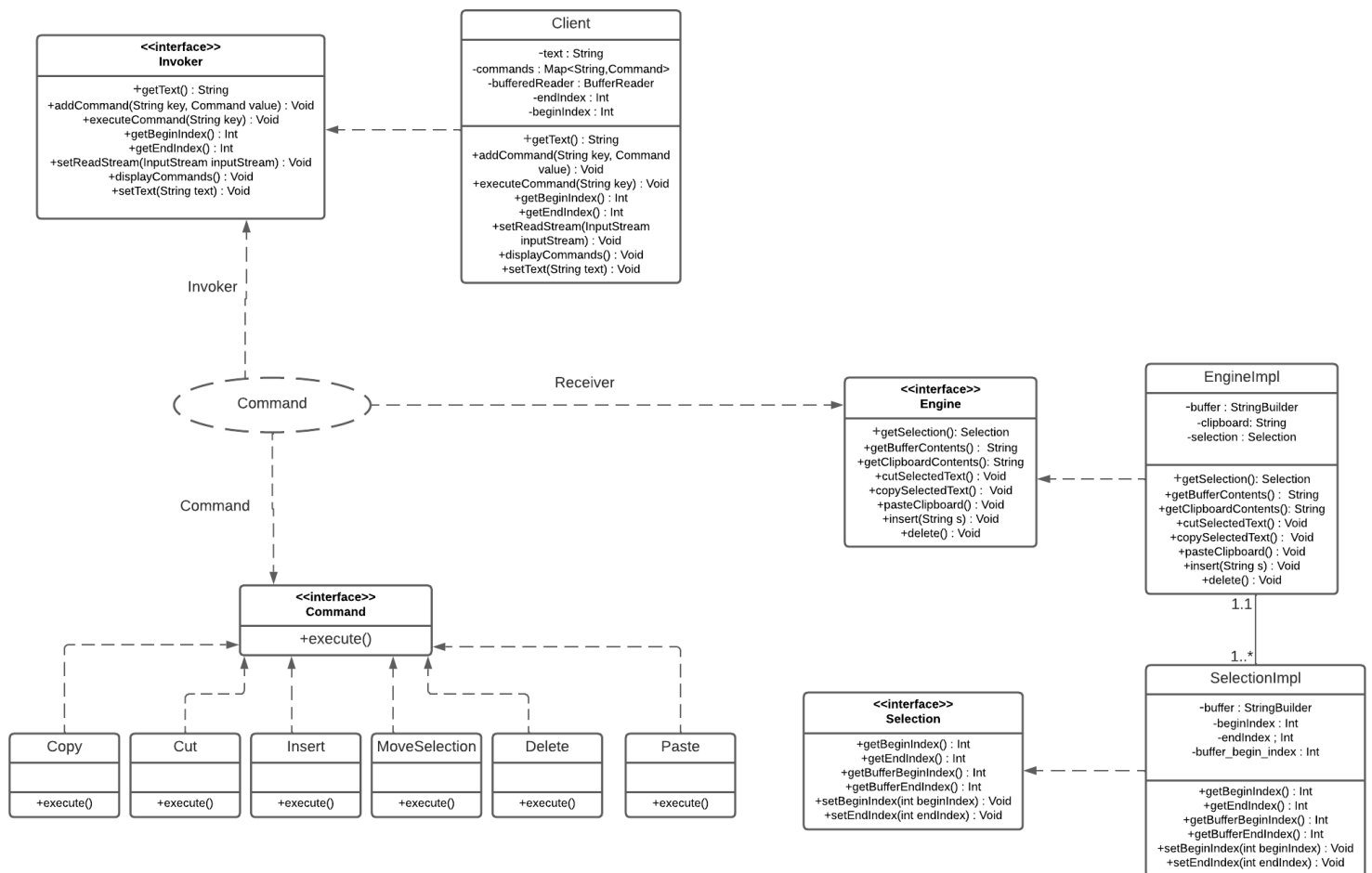
- **Copy** : Représente l'action de dupliquer un text sélectionné.
- **Cut** : Représente l'action de dupliquer un text sélectionné et le supprimé.
- **Paste** : Représente l'action de mettre le texte dupliqué dans un autre emplacement.
- **Delete** : Représente l'action de supprimer un text sélectionné.
- **Select** : Représente l'action de choisir le text que nous souhaitons modifié.

Ce projet a pour but aussi de mettre en pratique les connaissances théorique que nous avons appris sur les méthodes de conception afin de respecter les règles de programmation défini par des professionnels et éviter les BAD SMELL et appliquer les Design Patterns et les Bonnes Pratiques vues en cours.

2. Présentation de la première version V1

Pour la première version V1, nous avons implémenté le diagramme de classe ci-dessous :

2.1 Diagramme de classe de la V1



Afin d'atteindre les objectifs de la première version, nous avons implémenté des interfaces telles que « Engine », « Selection », « Invoker » et le Design Pattern « Command ».

Pour une meilleure clarification, nous avons :

- **EngineImpl** : C'est la classe qui implémente l'interface Engine, elle contient les méthodes
 - `getSelection()` : Retourne le texte sélectionné
 - `getBufferContents()` : Retourne le contenu du buffer
 - `getClipboardContents()` : Retourne le contenu du Presse-papiers
 - `cutSelectedText()` : La méthode pour l'action de Cut
 - `copySelectedText()` : La méthode pour l'action de Copy
 - `pasteClipboard()` : La méthode pour l'action de Paste
 - `insert(String s)` : La méthode pour l'action de Insert
 - `delete()` : La méthode pour l'action de Delete

- **SelectionImpl** : C'est la classe qui implémente l'interface Selection , elle contient les méthodes
 - getBeginIndex() : Retourne le premier index de la sélection
 - getEndIndex() : Retourne le dernier index de la sélection
 - getBufferBeginIndex() : Retourne le premier index du buffer
 - getBufferEndIndex() : Retourne le dernier index du buffer
 - setBeginIndex(int beginIndex) : Change la valeur du premier index de la sélection
 - setEndIndex(int endIndex) : Change la valeur du dernier index de la sélection
- **Client** : C'est la classe qui implémente l'interface Invoker, elle contient les méthodes
 - getText() : Retourne le texte inséré/modifié
 - addCommand(String key, Command value) : Ajouter une commande avec son identifiant key
 - executeCommand(String key) : Exécuter la commande en paramètre
 - getBeginIndex() : Retourne le premier index choisi par l'utilisateur
 - getEndIndex() : Retourne le dernier index choisi par l'utilisateur
 - setReadStream(InputStream inputStream) :
 - displayCommands() : Afficher des informations sur les commands
 - setText(String s) : Modifier/Ajouter un texte.
- **Copy/Cut/Paste/MoveSelection/Delete/Insert** : Ce sont des classes qui implémentent l'interface Command, ils contiennent la méthode Execute() qui permet à chaque classe d'exécuter l'action correspondante.

En effet, chaque implémentation de command fait référence à son Engine, et chaque classe qui implémente cette dernière performera son action.

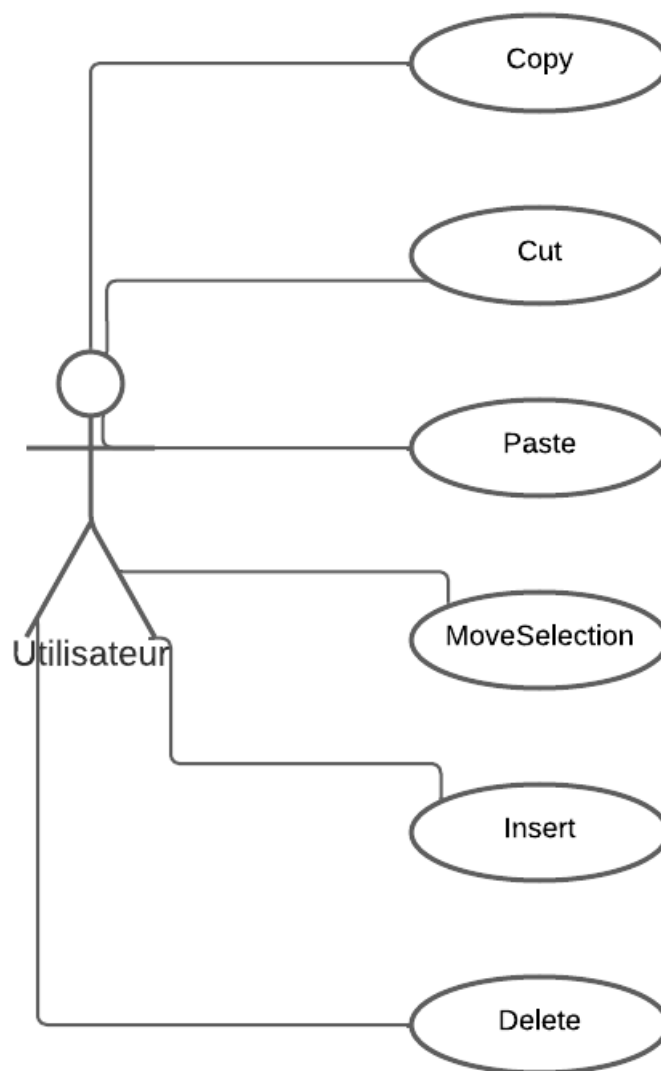
Quand une commande est créée, elle demande au constructeur en paramètre l'Engine et de cette façon la méthode execute() agit directement sur ce dernier.

Pour pouvoir exécuter les commandes, nous avons créé l'interface Invoker qui a la méthode executeCommand(String key) qui nous permettra de préciser quelle commande nous souhaitons exécuter.

Dans la classe Client (InvokerImpl) nous avons ajouté les champs privés text, beginIndex, endIndex qui sont nécessaires pour la commande Insert, et les méthodes getBeginIndex et getEndIndex et setText(String text).

2.2 Diagramme de cas d'utilisation de la V1

D'après le DCU ci-dessous, notre application permet à l'utilisateur de : Sélectionner, Copier, Coller, Couper, Supprimer et Insérer un texte



II. Présentation de la deuxième version du Mini Éditeur V2

1. Présentation de l'Interface de programmation (API) adapté pour V2

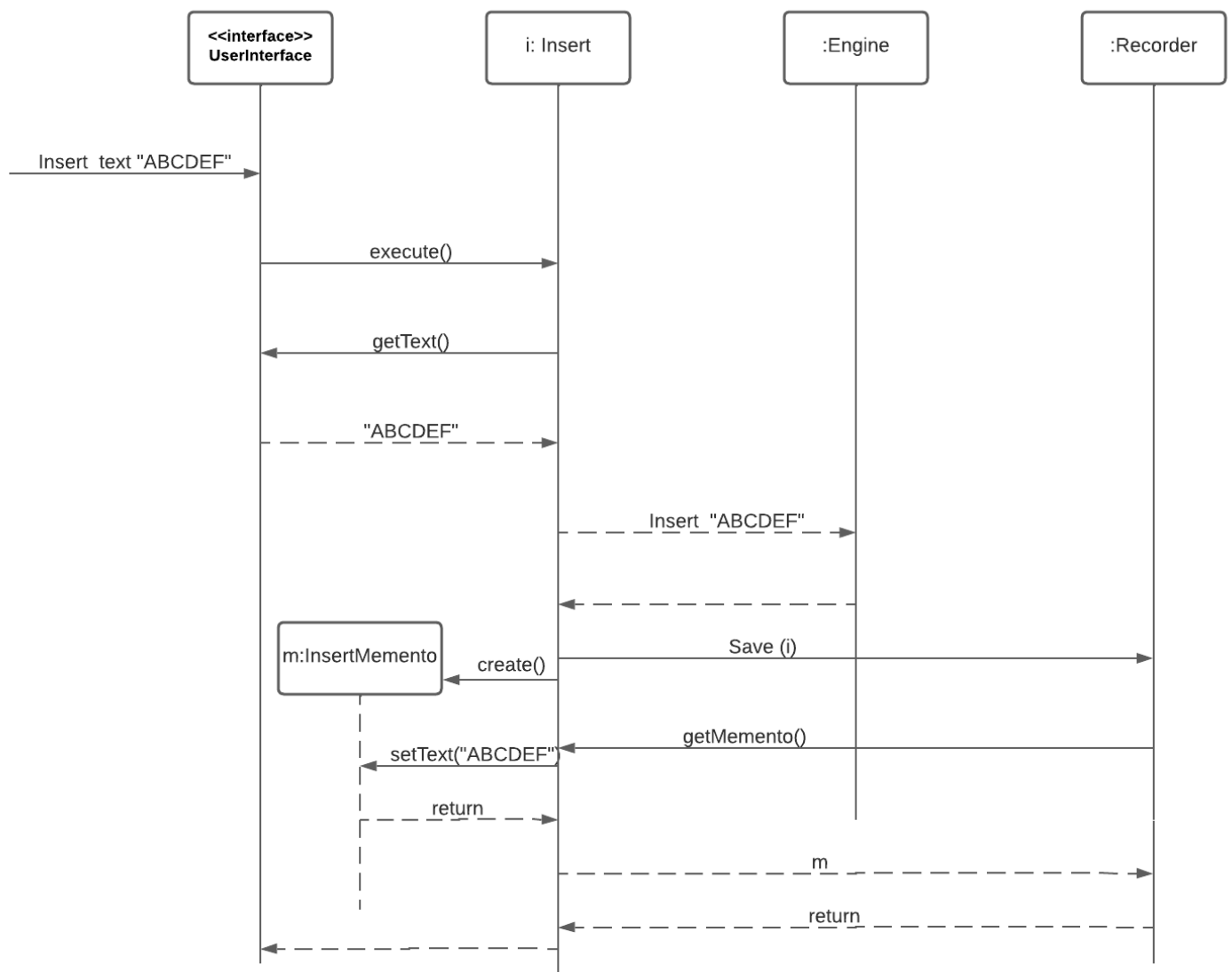
La deuxième version est identique à la première, sauf qu'elle offre plus de fonctionnalités à l'utilisateur , par exemple :

- La possibilité d'enregistrer une action effectuée
- La possibilité de sauvegarder cet enregistrement
- La possibilité de rejouer cette action

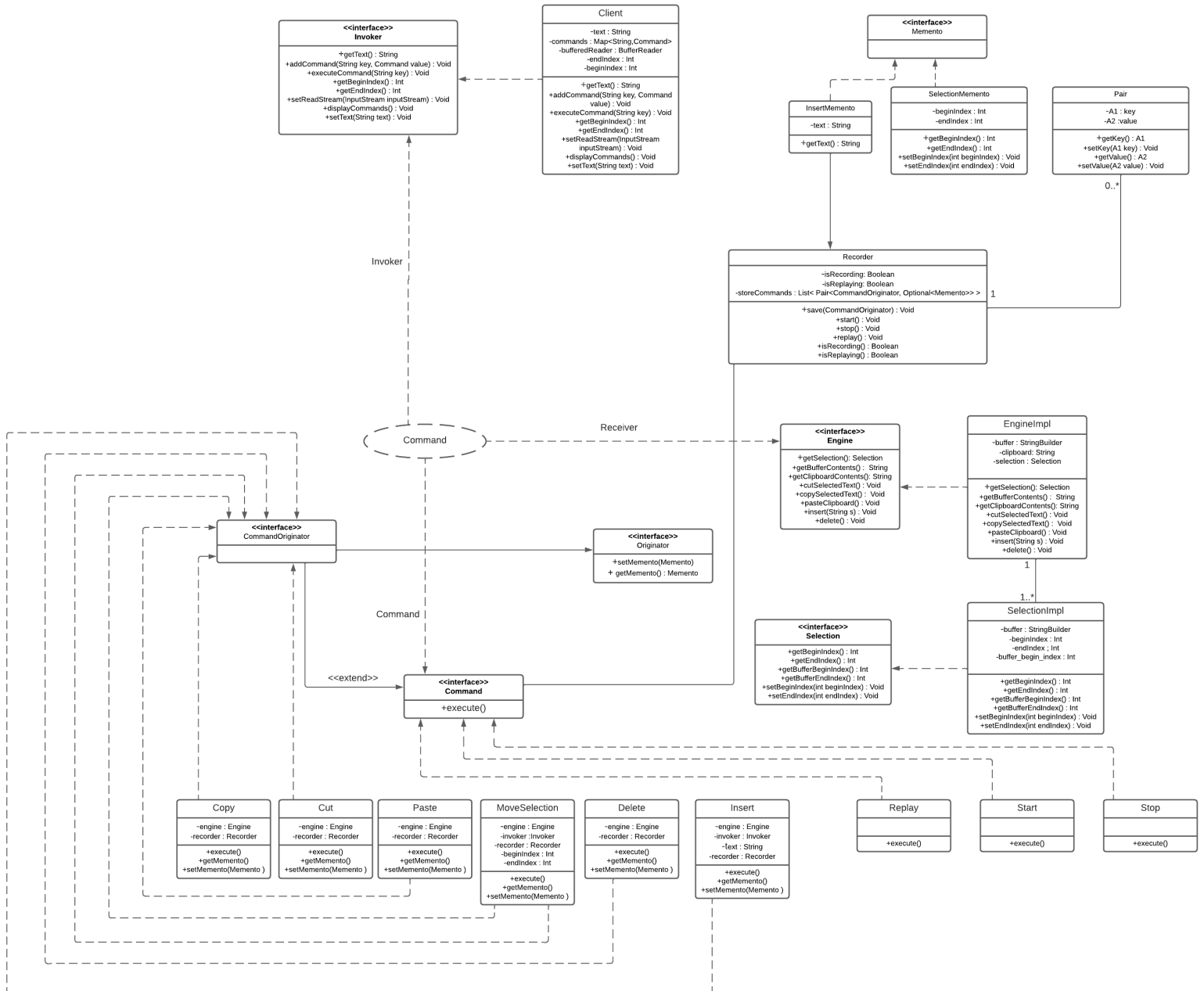
Pour ce faire, nous avons rajouter un autre Design Pattern, c'est le « Memento » qui nous permet d'enregistrer les différentes actions effectuées et les sauvegarder.

Nous pouvons expliquer son fonctionnement avec le diagramme de séquence ci-dessous :

2.1 Diagramme de séquence « Insert » de la V2



2.2 Diagramme de classe de la V2



La deuxième version V2 se base sur la version 1 pour implémenter ses fonctionnalités.

Design Pattern Memento :

Afin de stocker les paramètres des commandes, nous avons dû implémenter le Memento Design Pattern.

Pour cela, nous avons défini une interface vide « Memento » qui nous permet de créer des souvenirs en fonction des données qu'ils doivent stocker.

Nous avons associé un memento pour chaque commande.

La classe InsertMemento stocke le texte de l'insert avec la méthode getText().

La classe Selection Memento stocke la partie sélectionnée par les commandes correspondantes, elle nous permettra également de modifier la valeur du beginIndex et endIndex.

Recorder :

La classe Recorder contient des opérations pour enregistrer une commande avec son memento, elle démarre l'enregistrement avec Start(), l'arrête avec Stop() et refait les commandes enregistrées avec la méthode Replay().

Cette classe a deux listes pour stocker la commande et son memento correspondant (utilisé pour enregistrer les paramètres de la commande).

Lorsqu'une commande est exécutée, elle utilise la référence du Recorder pour invoquer la méthode save().

Ce dernier enregistre alors la commande, et lui demande son souvenir.

Si l'enregistrement n'a pas été lancé, nous sautons simplement la sauvegarde de la commande.

La méthode Replay() itère à travers toutes les commandes enregistrées, obtient le memento correspondant, le définit dans la commande (grâce à l'interface Originator) et exécute ensuite la commande.

Avant de faire un Replay des commandes enregistrées, nous devons arrêter l'enregistrement, sinon il aurait été infini.

Lorsque l'utilisateur démarre l'enregistrement, nous effaçons les commandes précédemment identiques afin de démarrer un nouveau enregistrement.

Les interfaces « Command » et « Originator »:

Command et Originator sont deux interfaces nécessaires à deux fins différentes.

Command fournit l'opération execute() qui nous permet d'exécuter la commande concrète.

Originator fournit l'opération `getMemento()` qui nous permet de créer un nouveau memento et `setMemento(Memento m)` qui permet de modifier un memento qui existe déjà.

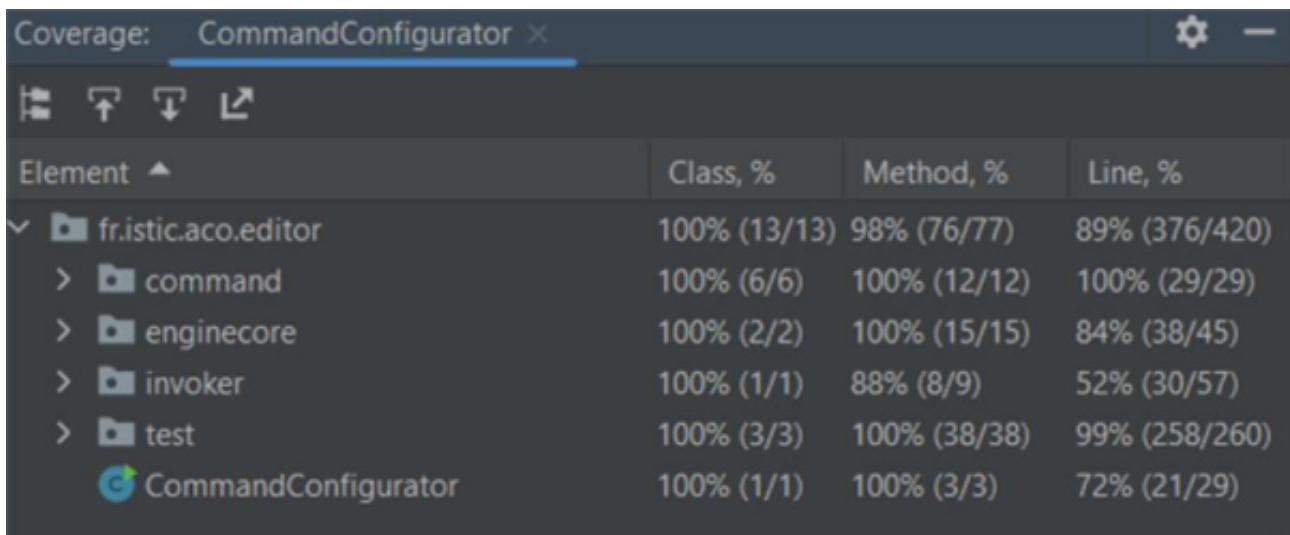
Pour avoir des commandes qui implémentent toujours les deux interfaces, nous avons créé une interface « `CommandOriginator` » unique qui étend `Command` et `Originator`.

2.3 Rapport des tests

La version 2 regroupe l'ensemble des classes du projet, en effets les tests unitaires de la deuxième version prennent en compte celui la première version V1.

Il faut noter que les tests sont les mêmes mais les jeux de données sont différents vu que les objets n'ont pas de la même structure.

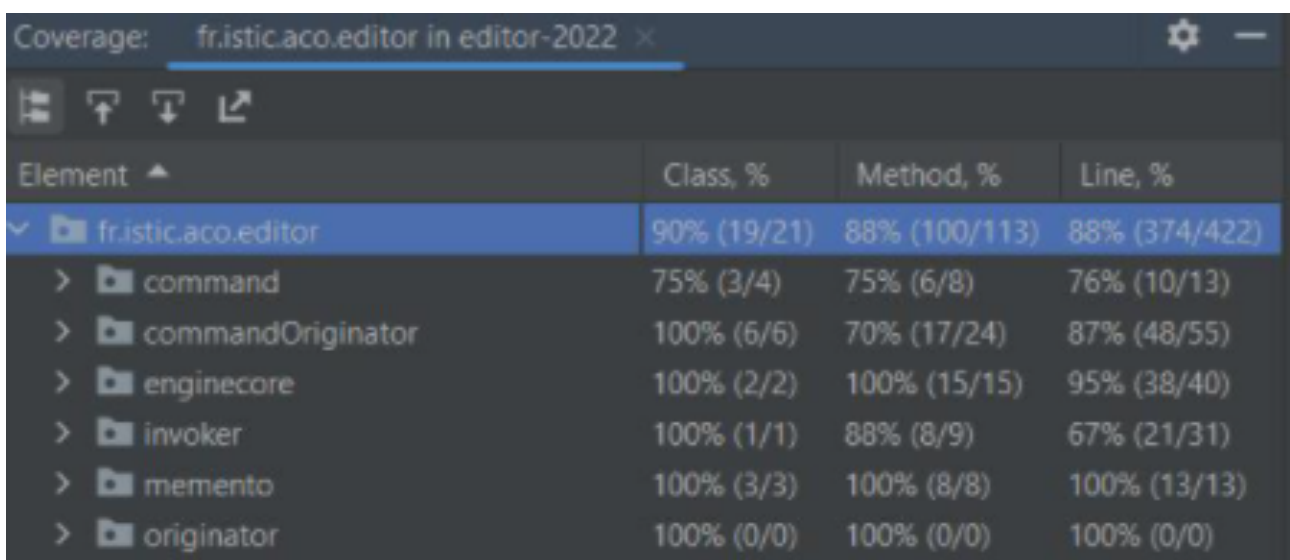
○ Coverage V1



The screenshot shows the 'Coverage: CommandConfigurator' window in IntelliJ IDEA. It displays a table with coverage data for various elements. The table has four columns: 'Element', 'Class, %', 'Method, %', and 'Line, %'. The elements listed are 'fr.istic.aco.editor', 'command', 'enginecore', 'invoker', 'test', and 'CommandConfigurator'. The coverage percentages are as follows:

Element	Class, %	Method, %	Line, %
fr.istic.aco.editor	100% (13/13)	98% (76/77)	89% (376/420)
command	100% (6/6)	100% (12/12)	100% (29/29)
enginecore	100% (2/2)	100% (15/15)	84% (38/45)
invoker	100% (1/1)	88% (8/9)	52% (30/57)
test	100% (3/3)	100% (38/38)	99% (258/260)
CommandConfigurator	100% (1/1)	100% (3/3)	72% (21/29)

○ Coverage V2



The screenshot shows the 'Coverage: fr.istic.aco.editor in editor-2022' window in IntelliJ IDEA. It displays a table with coverage data for various elements. The table has four columns: 'Element', 'Class, %', 'Method, %', and 'Line, %'. The elements listed are 'fr.istic.aco.editor', 'command', 'commandOriginator', 'enginecore', 'invoker', 'memento', and 'originator'. The coverage percentages are as follows:

Element	Class, %	Method, %	Line, %
fr.istic.aco.editor	90% (19/21)	88% (100/113)	88% (374/422)
command	75% (3/4)	75% (6/8)	76% (10/13)
commandOriginator	100% (6/6)	70% (17/24)	87% (48/55)
enginecore	100% (2/2)	100% (15/15)	95% (38/40)
invoker	100% (1/1)	88% (8/9)	67% (21/31)
memento	100% (3/3)	100% (8/8)	100% (13/13)
originator	100% (0/0)	100% (0/0)	100% (0/0)

Nous avons décidés de travailler avec un buffer car cela nous permettra de manipuler une classe et analyser son comportement dans les testes.

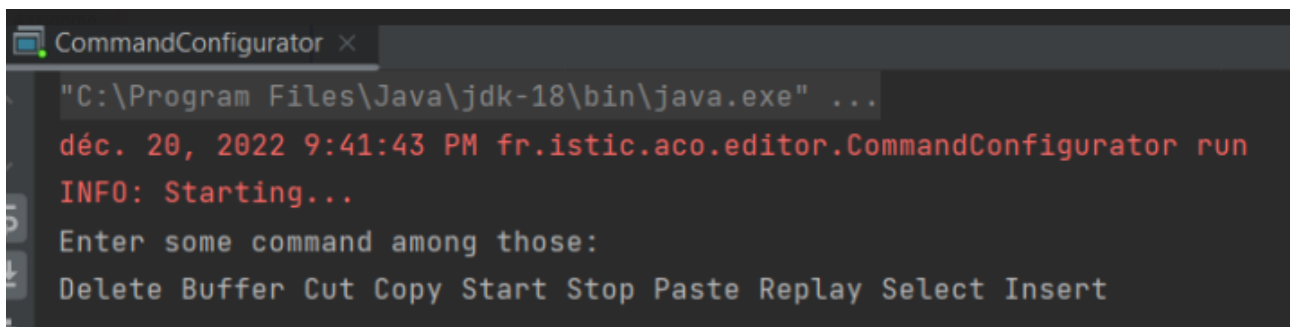
III. Méthode d'exécution

La procédure de l'exécution consiste à :

Pour la V1 :

- Se placer dans le répertoire de la version à utiliser
- Lancer l'éditeur, un menu s'affichera avec toutes les commandes à exécuter
- Sélectionner la commande que nous souhaitons exécuter.

Pour la V2 :

A screenshot of the CommandConfigurator V1 application. The window title is "CommandConfigurator". The main text area shows the command path "C:\Program Files\Java\jdk-18\bin\java.exe" followed by an ellipsis. Below this, it displays a timestamp "déc. 20, 2022 9:41:43 PM" and the command "fr.istic.aco.editor.CommandConfigurator run". It then shows "INFO: Starting..." and a prompt "Enter some command among those:". At the bottom, a list of commands is displayed: "Delete Buffer Cut Copy Start Stop Paste Replay Select Insert".

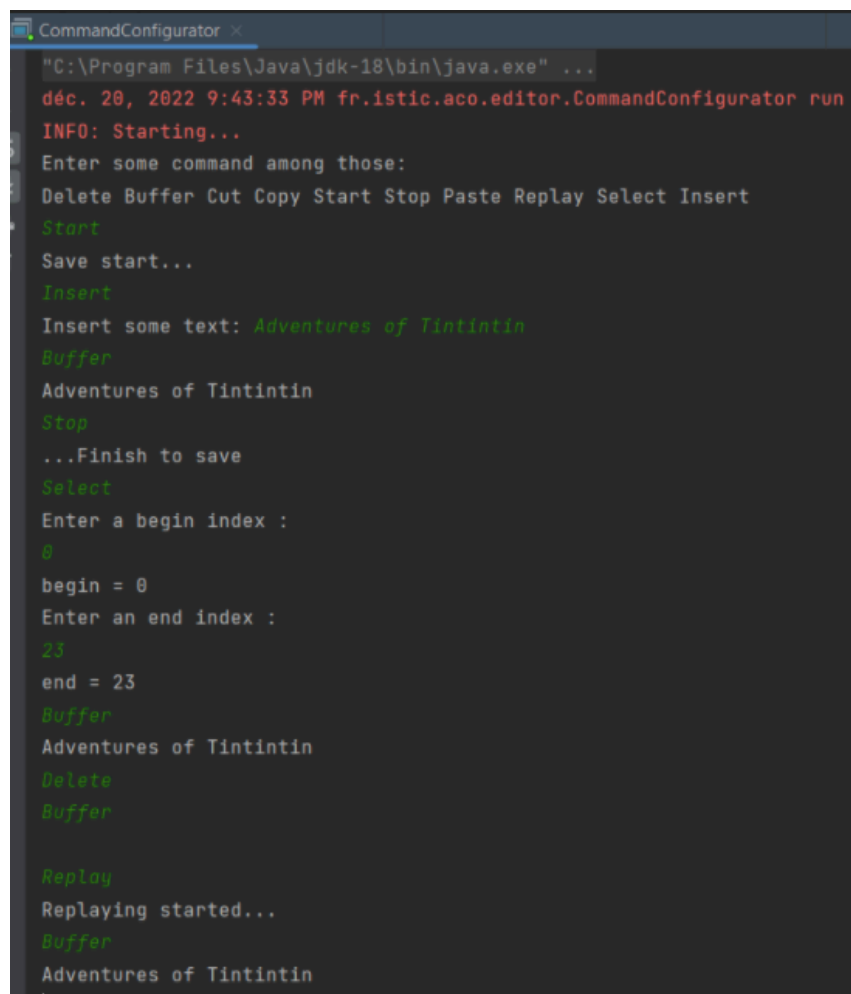
```
CommandConfigurator x
"C:\Program Files\Java\jdk-18\bin\java.exe" ...
déc. 20, 2022 9:41:43 PM fr.istic.aco.editor.CommandConfigurator run
INFO: Starting...
Enter some command among those:
Delete Buffer Cut Copy Start Stop Paste Replay Select Insert
```

-Il y'a l'enregistrement de la commande avec la méthode Start().

-Il y'a l'arrêt de l'enregistrement de la commande avec la méthode Stop().

-Il y'a le refait de la commande enregistré avec la méthode Replay().

-Vous pouvez vérifier à tout moment l'état du Buffer par simple appel de la commande **Buffer**.

A screenshot of the CommandConfigurator V2 application. The window title is "CommandConfigurator". The main text area shows the command path "C:\Program Files\Java\jdk-18\bin\java.exe" followed by an ellipsis. Below this, it displays a timestamp "déc. 20, 2022 9:43:33 PM" and the command "fr.istic.aco.editor.CommandConfigurator run". It then shows "INFO: Starting..." and a prompt "Enter some command among those:". A list of commands is displayed: "Delete Buffer Cut Copy Start Stop Paste Replay Select Insert". The user has entered "Start", followed by "Save start...", "Insert", and "Insert some text: Adventures of Tintintin". Then "Buffer" is entered, showing "Adventures of Tintintin" in the buffer. Next, "Stop" is entered, followed by "...Finish to save". Then "Select" is entered, prompting "Enter a begin index :", where "0" is entered, and "begin = 0". Then "Enter an end index :", where "23" is entered, and "end = 23". Then "Buffer" is entered again, showing "Adventures of Tintintin". Then "Delete" is entered, followed by "Buffer", showing "Adventures of Tintintin". Finally, "Replay" is entered, showing "Replaying started...", followed by "Buffer", showing "Adventures of Tintintin".

```
CommandConfigurator x
"C:\Program Files\Java\jdk-18\bin\java.exe" ...
déc. 20, 2022 9:43:33 PM fr.istic.aco.editor.CommandConfigurator run
INFO: Starting...
Enter some command among those:
Delete Buffer Cut Copy Start Stop Paste Replay Select Insert
Start
Save start...
Insert
Insert some text: Adventures of Tintintin
Buffer
Adventures of Tintintin
Stop
...Finish to save
Select
Enter a begin index :
0
begin = 0
Enter an end index :
23
end = 23
Buffer
Adventures of Tintintin
Delete
Buffer
Adventures of Tintintin
Replay
Replaying started...
Buffer
Adventures of Tintintin
```

IV. Conclusion

L'objectif de ce projet était de développer une application basée sur les fonctionnalités de base d'un mini éditeur de texte. Cette application devait offrir la possibilité à ses utilisateurs de faire une multitude d'actions décrits plus haut.

Pour ce faire, il a été choisi de répartir les tâches dans plusieurs versions, la version initiale étant l'implémentation des classes qui allaient constituer le cœur du système à savoir les interfaces Engine et Selection. Par la suite, dans la version 1, l'utilisation des principes du pattern de conception Command a permis l'implémentation des commandes qui font l'objet des appels et des actions afin d'encapsuler les informations nécessaires pour déclencher un évènement particulier à la demande de l'utilisateur car c'est avec ces commandes et seulement elles (en tapant leurs noms) que l'utilisateur va interagir avec le système. Dans la version 2, on offre la possibilité à l'utilisateur d'enregistrer des actions et de les rejouer par la suite par l'utilisation des principes du pattern Memento.

Nous vous avons apporté à travers ce rapport en tandem avec les codes sources présents sur le repo **gitlab** de notre projet, un cheminement de notre compréhension du sujet et des étapes entreprises pour la réalisation de ce programme ainsi qu'un aperçu de la couverture d'exécutions des tests de notre programme qui vous le verrez répondent aux contraintes imposées.

La réalisation de ce projet nous a permis de mettre en pratique des éléments que nous avons vu de façon théorique dans les cours mais aussi de pouvoir étendre nos connaissances sur le sujet des patterns de conception Command et Memento à travers nos cours et de nos recherches personnelles.