ESIR 2
INFO IoT

Goetgheluck Malo
Ezzonfari Ilyass

OMD

Rapport TP 2

Nous attestons que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée.

Mini-éditeur

Introduction

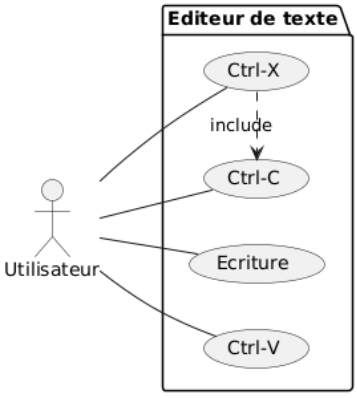
Le but de ce TP est de modéliser et réaliser un éditeur de texte basique. Nous avons plusieurs prérequis à respecter dans la réalisation de cet éditeur. Les objectifs de ce TP sont de présenter une technique de conception et de programmation à objets simple, de montrer les différents moyens de coordination des objets, de montrer la constructions parallèle des diagrammes statiques et dynamiques, de montrer la mise en œuvre des patrons de conceptions, et de montrer le passage des modèles de conception au programme. Pour réaliser ce TP, nous allons utiliser PlantUML afin de réaliser les diagrammes, et Java afin de programmer notre éditeur de texte.

Version n°1

Les attentes par rapport à notre éditeur de texte sont les suivantes :

- Le texte est contenu dans un buffer (zone de travail),
- Il existe une notion de sélection de texte, avec des commandes utilisateur permettant de déplacer le début et la fin de la sélection,
- Copie de la sélection dans le presse-papier,
- Copie de la sélection dans le presse-papier puis effacement de la sélection,
- Remplacement (« collage ») de la sélection par le contenu du presse-papier,
- L'interface homme-machine est d'un type quelconque (textuelle ou graphique)

Nous avons donc pour commencer réalisé le diagramme de cas d'utilisation.



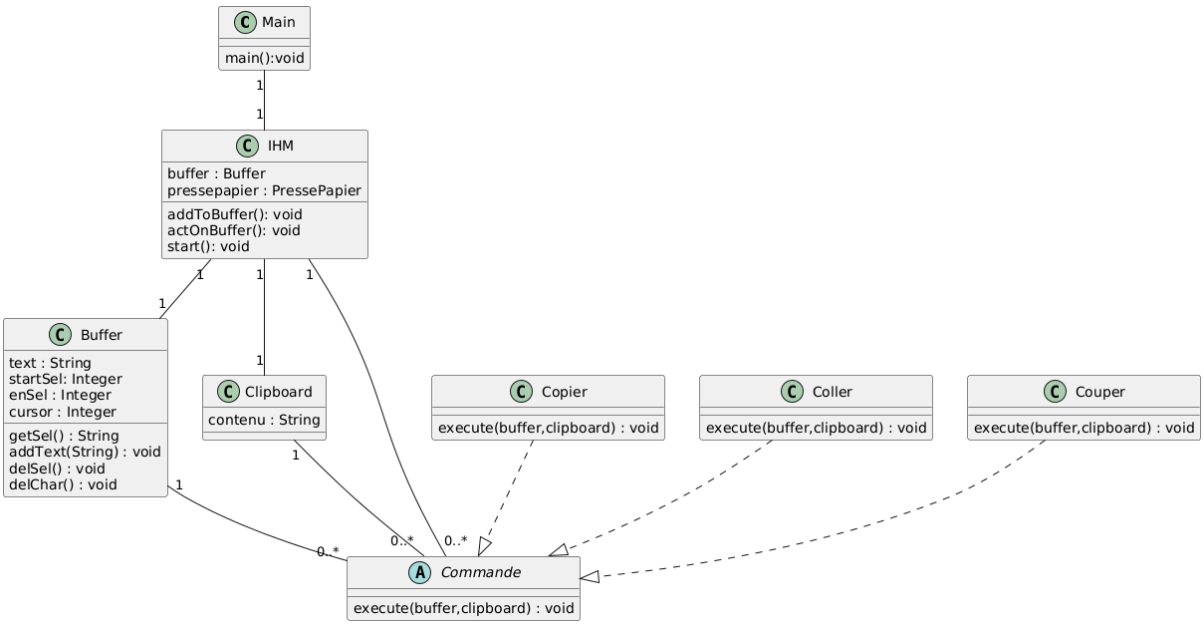
Et nous avons ensuite décris chaque cas d'utilisations :

Nom du cas d'utilisation : Ecriture.
Brève description : L'utilisateur écrit dans l'éditeur.
Acteurs : Utilisateur.
Contexte : L'utilisateur veut écrire dans l'éditeur de texte.

Données en entrée et pré-conditions : L'éditeur est ouvert. **Scénario principal** : L'éditeur saisit au clavier le texte voulu, qui s'affiche dans l'éditeur.
Nom du cas d'utilisation : Copier. **Brève description** : L'utilisateur copie une sélection de texte. **Acteurs** : Utilisateur. **Contexte** : L'utilisateur veut copier une sélection. **Données en entrée et pré-conditions** : L'éditeur est ouvert. **Scénario principal** : L'utilisateur sélectionne du texte et choisit de copier la sélection, le texte sélectionné sera donc copié dans le presse-papier.
Nom du cas d'utilisation : Coller. **Brève description** : L'utilisateur colle une sélection de texte. **Acteurs** : Utilisateur. **Contexte** : L'utilisateur veut coller une sélection. **Données en entrée et pré-conditions** : L'éditeur est ouvert. **Scénario principal** : L'utilisateur met le curseur sur l'emplacement ou il veut que le texte soit collé et colle le contenu du presse-papier.
Nom du cas d'utilisation : Couper. **Brève description** : L'utilisateur coupe une sélection de texte. **Acteurs** : Utilisateur. **Contexte** : L'utilisateur veut couper une sélection. **Données en entrée et pré-conditions** : L'éditeur est ouvert. **Scénario principal** : L'utilisateur sélectionne du texte et choisit de couper la sélection,la sélection va être effacé et le texte sera copié dans le presse-papier.

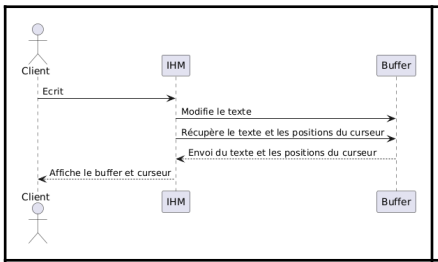
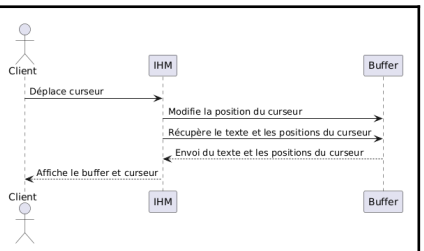
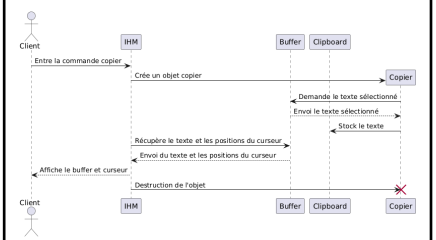
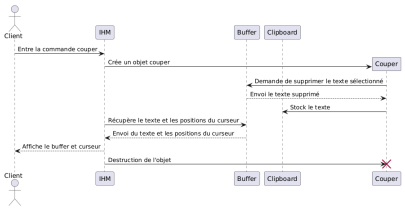
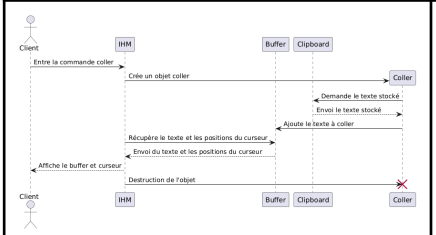
À l'aide du diagramme de cas d'utilisation et des scénarios d'utilisation, nous réalisons ensuite le diagramme de classe et les diagrammes de séquences.

Ce diagramme de classe montre l'architecture de notre éditeur de texte. La classe main contient la méthode principale qui crée les objets nécessaires : un buffer pour stocker le texte, un clipboard pour stocker le contenu temporaire, et une interface utilisateur graphique IHM. La classe IHM utilise ces objets pour interagir avec l'utilisateur et gérer le texte, le buffer stocke le texte actuel, ainsi que les indices de début et de fin pour la sélection, et fournit des méthodes pour obtenir et modifier cette sélection, le clipboard stocke le texte copié ou coupé et permet de récupérer ou de définir ce contenu. L'interface commande impose une méthode execute(Buffer, Clipboard) que les classes Copier, Coller, et Couper implémentent pour réaliser leurs actions spécifiques. Ces trois classes sont des implémentations concrètes du modèle de commande, chacune définissant comment elles interagissent avec le Buffer et le Clipboard. Pour les liens, la classe main crée et utilise IHM, qui elle même crée et manipule un buffer et un clipboard pour gérer le texte. Les classes Copier, Couper, et Coller implémentent l'interface Commande et dépendent du buffer et du clipboard pour exécuter leurs actions.



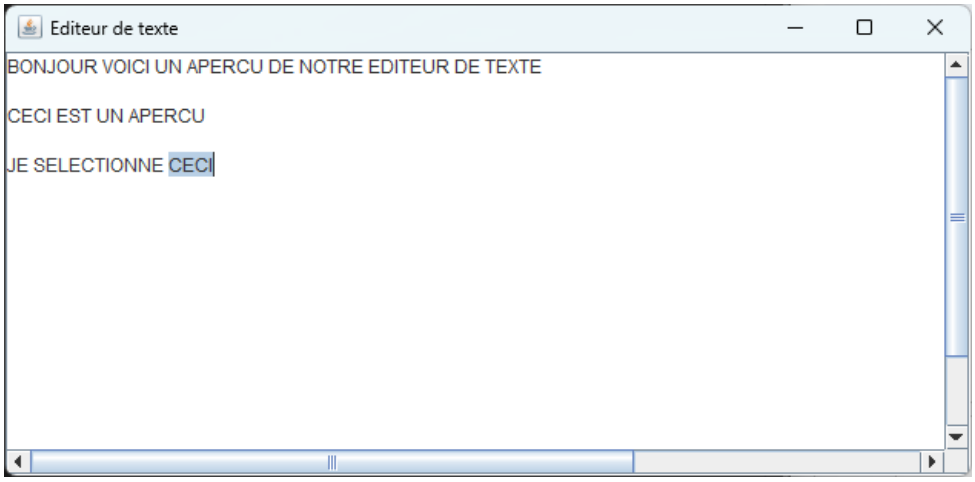
Notons que les getters et setters pour chacun des attributs des classes ne sont pas représentés pour des questions de lisibilité.

Les diagrammes de séquence représentent les étapes réalisées pour chaque cas d'utilisations.

<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Ecriture</td><td>Déplacement du curseur</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Copier</td><td>Couper</td></tr>
<tr><td> 
</td><td></td></tr>
<tr><td>Coller</td><td></td></tr>
</table>

Nous sommes ensuite passé à la partie développement de notre éditeur. Nous avons donc fait des classes pour chaque instance du diagramme de classe. Nous avons, à l'aide des librairie java.awt et javax.swing, réalisé une interface graphique, qui nous a permis d'afficher le texte, et de récupérer les interactions avec la souris pour bouger le curseur. Cependant, nous avons bloqué ou ignoré toutes autres fonctionnalités, car nous devions implémenter nous même le reste de l'éditeur de texte.

Voici un apercu de la fenêtre de notre éditeur de texte.



Comme vous l'avez certainement remarqué, nous avons fait le choix de n'implémenter que le texte majuscule. De plus, il apparait qu'un bruit venant de windows intervient dès que l'on veut supprimer un caractère, nous n'avons malheureusement pas trouvé comment régler ce problème. Nous avons pû utiliser des barres de défilement verticale et horizontale. Grâce au librairie utilisées, nous pouvons afficher le curseur et la zone de texte sélectionnée. La zone sélectionnée peut être définie grâce à la souris, ou alors en maintenant shift et en apuyant sur les flèches vers la gauche ou le bas.

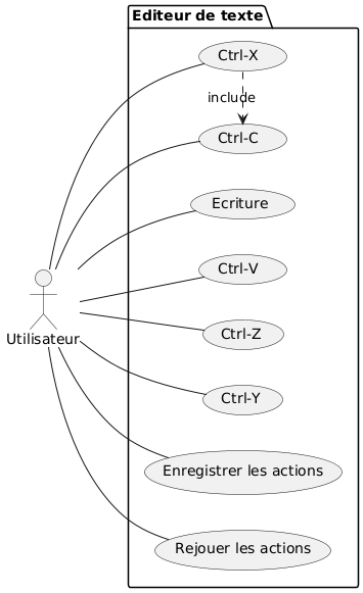
Cette version permet bien toute les fonctionnalités de base désirée.

Version n°2

Nous voulons maintenant implémenter de nouvelles fonctionnalités. Celles-ci sont listées ci-dessous.

- Enregistrer/rejouer les actions de l'utilisateur (e.g., script),
- Réaliser le défaire/refaire, avec une capacité quelconque dans le défaire (autrement dit on peut revenir au début).

Nous modifions donc le diagramme de cas d'utilisation afin qu'il réponde à ces nouvelles fonctionnalités.



Puis nous décrivons les nouveaux cas d'utilisations :

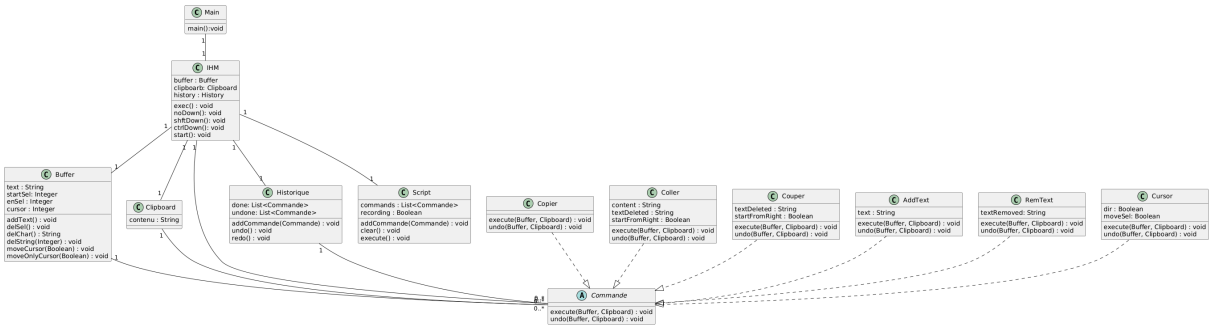
Nom du cas d'utilisation : Défaire. **Brève description** : L'utilisateur défait la dernière action effectuée dans l'éditeur de texte. **Acteurs** : Utilisateur. **Contexte** : L'utilisateur souhaite annuler une action qui a été effectuée par erreur ou qui doit être corrigée. **Données en entrée et pré-conditions** : Une ou plusieurs actions ont été effectuées. **Scénario principal** : L'utilisateur utilise Ctrl-Z pour annuler la dernière action effectuée, que ce soit une modification de texte, une suppression, un collage, etc.
Nom du cas d'utilisation : Refaire. **Brève description** : L'utilisateur refait une action qui a été annulée précédemment. **Acteurs** : Utilisateur. **Contexte** : L'utilisateur souhaite réappliquer une action qu'il a annulée précédemment en utilisant Ctrl-Z. **Données en entrée et pré-conditions** : Une ou plusieurs actions ont été annulées. **Scénario principal** : L'utilisateur utilise Ctrl-Y pour restaurer une action annulée par Ctrl-Z, comme une réécriture ou une suppression. **Variantes, cas d'erreurs** : Si aucune action n'a été annulée, la commande "Refaire" est impossible.
Nom du cas d'utilisation : Enregistrer les actions. **Brève description** : L'utilisateur enregistre une série d'actions effectuées dans l'éditeur de texte pour les rejouer plus tard. **Acteurs** : Utilisateur. **Contexte** : L'utilisateur souhaite enregistrer une série d'actions répétitives pour les rejouer plus tard de façon automatique. **Données en entrée et pré-conditions** : L'éditeur est ouvert. **Scénario principal** : L'utilisateur active l'enregistrement, effectue plusieurs actions

(écriture, copier, coller, etc.), puis arrête l'enregistrement. **Variantes, cas d'erreurs** : Si aucune action n'a été effectuée, l'enregistrement est vide.
Nom du cas d'utilisation : Rejouer les actions. **Brève description** : L'utilisateur rejoue une série d'actions qui ont été précédemment enregistrées. **Acteurs** : Utilisateur. **Contexte** : L'utilisateur souhaite reproduire automatiquement une séquence d'actions enregistrées dans l'éditeur. **Données en entrée et pré-conditions** : Une série d'actions a déjà été enregistrée. **Scénario principal** : L'utilisateur sélectionne une séquence enregistrée et la rejoue dans l'éditeur. Les actions se déroulent automatiquement dans l'ordre dans lequel elles ont été effectuées. **Variantes, cas d'erreurs** : Si aucune action n'a été enregistrée, il n'y a rien à rejouer.

Nous modifions ensuite le diagramme de classe grâce aux cas d'utilisations.

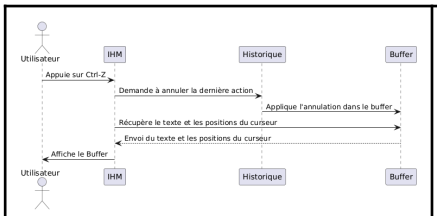
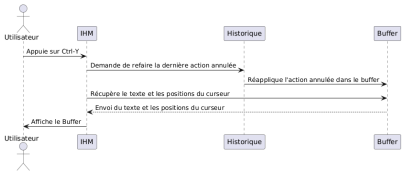
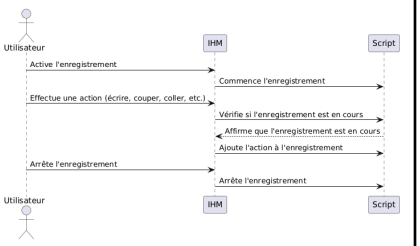
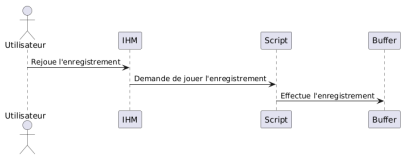
Dans cette version on ajoute des nouvelles fonctionnalités par rapport à la v1, la principale amélioration est l'ajout de la classe Historique, qui permet de gérer l'enregistrement des actions effectuées et leur annulation ou réapplication via défaire et refaire. Cela permet de mettre en place les fonctionnalités Ctrl-Z et Ctrl-Y. L'IHM est maintenant liée à l'Historique, ce qui lui permet d'ajouter chaque commande à l'historique après son exécution. Ainsi, le système peut rejouer ou annuler les commandes selon les interactions de l'utilisateur. Ces ajouts permettent une gestion complète des actions de l'utilisateur avec l'éditeur. Nous ajoutons aussi la classe Script, qui nous permet, à l'image de l'historique, d'enregistrer des actions lorsque nous voulons enregistrer un script.

Nous avons aussi modifié le fonctionnement d'écriture et du déplacement du curseur. En effet, nous passons maintenant par des commandes afin de pouvoir les stocker dans l'historique et le script.



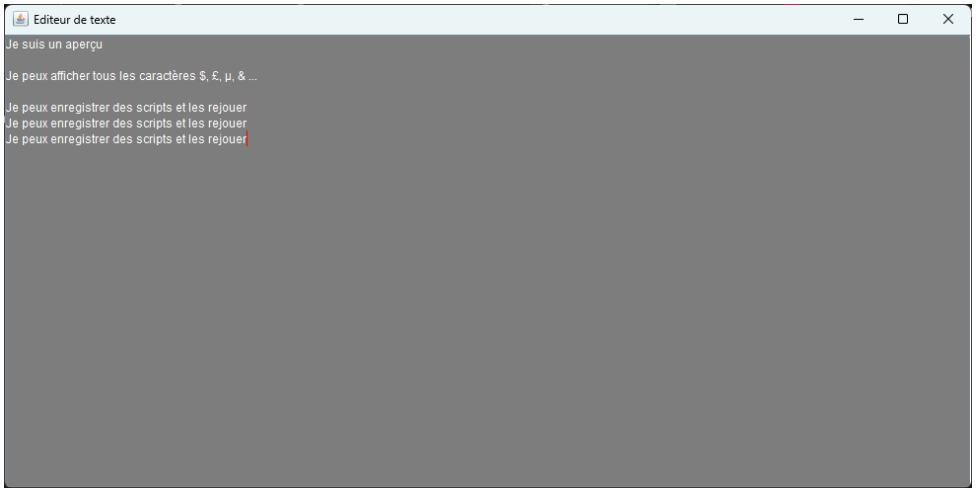
De nouveau, les getters et les setters ne sont pas représentés par problème de lisibilité.

Nous faisons ensuite les diagrammes de séquences pour chaque cas d'utilisations.

<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Défaire</td><td>Refaire</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Enregistrer le script</td><td>Rejouer le script</td></tr>
</table>

Nous sommes ensuite passé au développement de la v2. Nous avons tout d'abord dû restreindre les mouvements du curseur aux entrées claviers de l'utilisateur. Nous pouvons donc maintenant bouger le curseur seulement en appuyant sur les touches gauche et droite. Pour sélectionner, il suffit d'appuyer sur shift. De plus, nous pouvons sur la v2 taper n'importe quel caractère. Par contre, si nous appuyons sur une touche qui n'inclus par de caractère imprimable, notre éditeur affiche un caractère vide.
Nous pouvons donc maintenant appuyer sur ctrl et Z ou Y afin de défaire ou refaire une action. Nous pouvons aussi appuyer sur ctrl R afin de lancer l'enregistrement du script. Le curseur s'affiche alors en rouge. Lorsque l'utilisateur veut arrêter d'enregistrer, il appuie de nouveau sur ctrl R, le curseur repasse en noir. Pour rejouer l'enregistrement, l'utilisateur doit presser ctrl et E, ce qui va rejouer l'enregistrement. Si l'enregistrement est vide, ou qu'il n'y a pas eu d'enregistrement, alors rien ne se passera.
Nous avons changé la couleur du background pour que cela soit plus agréable pour nos yeux. Lorsque nous quittons l'éditeur, le programme s'arrête lui aussi.

Voici un apercu de la deuxième version de notre éditeur de texte.



Conclusion

Durant ce TP, nous nous sommes rendus compte de l'importance d'utiliser des outils de modélisation. En effet, se poser la question de comment nous pourrions faire un programme que nous pourrons améliorer facilement plus tard a été décisif dans le bon déroulé de notre TP. Nous pouvons prendre comme exemple les quelques modifications que nous avons dû réaliser entre la v1 et la v2 telles que la suppresion de la gestion du curseur par la souris, ou encore l'ajout des commandes addText et RemText afin de faciliter l'implémentation de l'historique et du script.

Pour conclure, ce TP nous aura permis de nouveau de nous rendre compte de l'importance des outils de modélisation. En effet, afin de produire un programme complet et surtout, facilement améliorable, il est très important de se poser la question de comment nous pouvons faire en sorte pour rendre facile l'implémentation de nouvelles fonctionnalités plus tard.