

Algorithmique avancé

Editeur Ligne

Berthet Laurent – Bugnard Julien

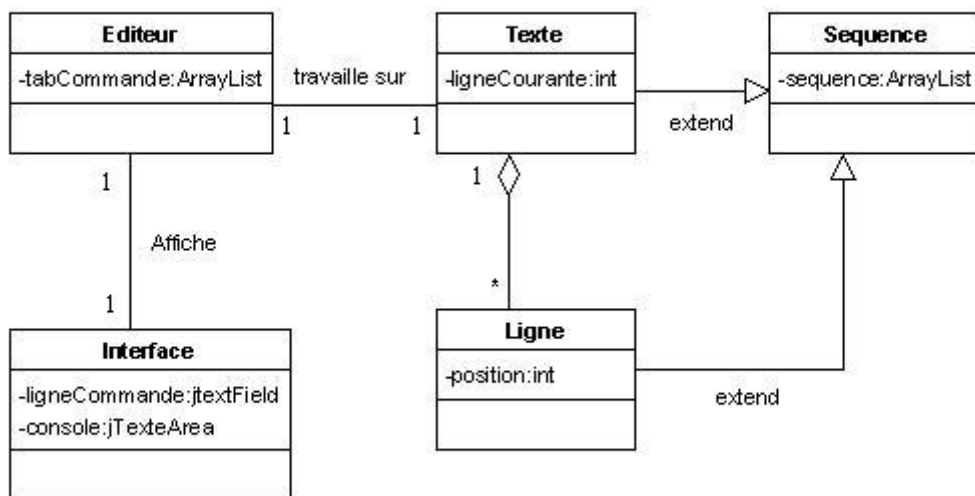
Plan :

Introduction :	2
I. Diagramme UML	2
II. Explications des Classes	3
1. Editeur :	3
2. Interface :	3
3. Texte :	4
4. Ligne :	4
5. Séquence :	5
6. Test :	5
Conclusion	5

Introduction :

Le but de ce sujet était de créer un éditeur ligne. Nous avions qu'une seule contrainte qui était le choix du langage (JAVA), pour le reste, nous étions libre dans nos choix. Nous avons donc décidé de créer notre propre interface en essayant de nous rapprocher d'une console quelconque.

I. Diagramme UML



II. Explications des Classes

1. Editeur :

Cette classe va gérer tout ce que l'on a tapé dans la console. L'éditeur a besoin de plusieurs attributs :

- Stack pileLigne : c'est la pile de l'éditeur qui permet de stocker des lignes lorsqu'on fait la commande K.
- Texte texte : c'est le texte sur lequel on travail
- JTextArea console : permet l'affichage des résultats des commandes.
- Boolean modeInsertion : permet de savoir si l'éditeur prend en compte les commandes.
- Int posHistorique : permet de retrouver une commande déjà entrée.
- ArrayList<String> historique : stocke toutes les commandes déjà entrées.
- ArrayList<String> tabCommande : pour avoir accès à la dernière commande

Pour pouvoir analyser une commande, on utilise la méthode analyseCommande (JTextArea console, JTextField ligneCommande). Cette méthode récupère, ce que l'on a entré dans ligneCommande, on le découpe en morceaux et on la stocke dans un tableau. Si l'éditeur est en mode insertion, on va insérer le contenu de ligneCommande. Si le premier élément de ce tableau est une commande, on analyse la suite. Sinon, ce n'est pas une commande donc on affiche un message d'erreur.

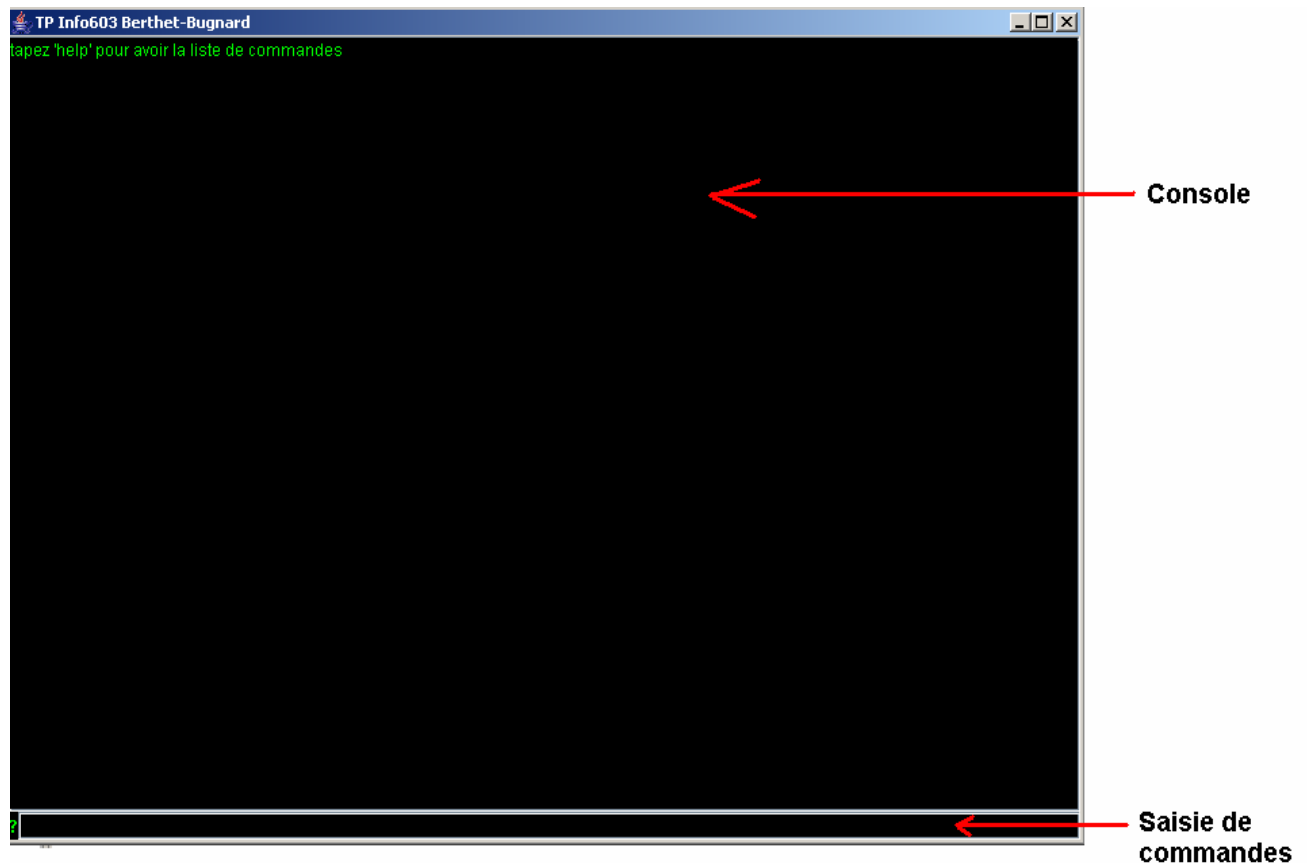
Suivant la commande que l'on a entrée, on appelle la méthode correspondante. Avant d'appeler la méthode, on vérifie que la commande est bien formée.

Nous avons choisit de ne pas expliquer les méthodes concernant les commandes car elles sont toutes commentées dans le code du programme. Nous pensons qu'il n'est pas nécessaire de répéter les explications.

2. Interface :

Cette classe affiche les résultats des commandes.

Notre fenêtre est décomposée en deux parties : la fenêtre d'affichage et la fenêtre permettant la saisie de commandes. Pour réaliser cette fenêtre, nous avons utilisé un JTextArea pour l'affichage et un JTextField pour la saisie des commandes.



Pour pouvoir gérer les commandes entrées, un `KeyListener` a été ajouté au `TextField`.
Suivant les touches sur lesquelles nous allons appuyées, une action différente sera exécutée.

Le `KeyListener` va effectuer une action pour les touches « entrée », « echap », « i » et les flèches haut et bas du clavier.

La touche « entrée » sert à valider une commande, c'est à dire qu'à la fin d'une commande, il faut appuyer sur « entrée ».

La touche « echap » sert à sortir du mode insertion.

La touche « i » permet de passer l'éditeur en mode insertion.

Les flèches "haut" et "bas" permettent d'afficher de sélectionner une commande que l'on a déjà tapé.

3. Texte :

Cette classe représente le texte sur lequel on travaille.

Un texte est représenté par une séquence de Lignes et une ligne courante. Lorsqu'on crée le texte, on initialise la ligne courante à -1 pour dire que le texte est vide.

On peut ajouter une ligne au texte et en supprimer une. Pour ajouter une ligne, il faut passer en paramètre la ligne que l'on veut ajouter, la ligne est ajoutée à la fin du texte. Pour supprimer une ligne, il faut passer le numéro de la ligne qui va être supprimée.

4. Ligne :

Cette classe représente une séquence de caractères.

Pour créer une ligne, il faut donnée une chaîne de caractères correspondant à la ligne que l'on veut créer. On peut aussi transformer une ligne en `String` pour afficher la ligne.

5. Séquence :

Cette classe représente une liste d'objets qui ont un ordre défini. La liste d'objets est un ArrayList.

6. Test :

Cette classe permet d'instancier nos objets. C'est elle qui contient le "main". Dans le main, on initialise un texte vide et l'éditeur associé au texte.

Conclusion

Nous pensons avoir réalisé un éditeur de texte comprenant toutes les fonctionnalités demandées. Nous en avons même rajoutées quelques unes supplémentaires afin d'améliorer l'ergonomie de l'éditeur.