Master 1 ALMA

Projet de Génie Logiciel

MINIED : Version 3

Étudiants :
Coraline MARIE et Vincent RAVENEAU

Intervenant :
Gerson SUNYÉ

13 décembre 2013

Sommaire

1 Introduction

Le Génie Logiciel est la discipline qui permet l'aboutissement d'un projet depuis son idée, jusqu'à son utilisation par un client. Cette discipline est donc indispensable car elle est à l'origine de tout logiciel, et permet sa conception de façon fiable et structurelle.

Actuellement en première année de Master Alma à l'Université de Nantes, nous avions pour projet de Génie Logiciel, la conception d'un éditeur de texte simplifié. Cet éditeur basique : MiniEd, devait disposer de diverses fonctionnalités, qui ont été développées dans trois versions différentes du logiciel.

Ce rapport présente donc la conception de MiniEd, au travers de ses trois versions fonctionnelles. Ces différentes versions disposent chacune de fonctionnalités et de commandes basiques, qui ont eu comme objectifs pédagogiques de nous sensibiliser sur le rôle du Génie Logiciel dans l'Informatique.

2 Conception

2.1 Dépendances externes

Avant même de concevoir un logiciel, il faut au préalable le penser, ainsi que définir ses besoins et ses différentes fonctionnalités. Dans cette optique, notre cahier des charges nous imposait de concevoir l'éditeur de texte dans le langage de programmation Scala, un langage récent qui allie de façon subtile la programmation orientée objet avec la programmation fonctionnelle. Ce langage ne fût pas choisi pour ses performances en programmation orientée objet, mais plutôt dans une optique pédagogique, car nous ne l'avions encore jamais utilisé avant le projet MiniEd.

Dans un soucis de fiabilité, de qualité et de performances, le code de l'éditeur de texte a été testé à l'aide de JUnit. Ce dernier est un framework de test unitaire qui fût créé pour le langage de programmation Java, et qui est aujourd'hui compatible avec le langage de programmation Scala.

Le cahier des charges de MiniEd nous demandait de modéliser une interface homme-machine. Dans un soucis d'esthétique, cette dernière est de type graphique, ce qui la rend plus intuitive et plus agréable pour les utilisateurs. Ainsi, pour concevoir l'interface, nous avons fait appel à la bibliothèque Java Swing qui est une bibliothèque graphique pour Java.

2.2 Fonctionnalités du logiciel

Le cahier des charges de MiniEd réclamait plusieurs fonctionnalités, qui ont été développées sur plusieurs versions fonctionnelles de l'éditeur. La programmation s'est donc déroulée en trois étapes : trois versions de MiniEd.

2.2.1 Version 1 : Les actions de base

Les actions de base de MiniEd, sont celles qui ont été mises en place pour la première version du logiciel. Comme on peut s'y attendre, celles-ci sont fondamentales et sont :

- La saisie de texte.
- Les actions Couper, Copier, Coller et Effacer.
- L'implémentation d'un buffer qui est notre zone de travail, et dans lequel est contenu le texte.
- La mise en place de la sélection qui donne la possibilité de sélectionner une partie, voire même la totalité du texte. Ainsi, le texte sélectionné peut être retravaillé à l'aide d'autres fonctionnalités.
- L'implémentation d'un presse-papier qui contient du texte que l'on a au préalable, couper ou copier, et qui peut ensuite être collé.

— La création de macros qui groupe certaines commandes utilisateur. Dans notre cas cette fonctionnalité était l'un des objectifs de la version 1, mais nous n'avons pas eu le temps de la mettre en place avant la version 2.
— La conception d'une interface homme-machine qui est de type graphique. Plusieurs interfaces peuvent être générées et elles sont toutes associées au même buffer.

2.2.2 Version 2 : L'enregistrement des actions de l'utilisateur

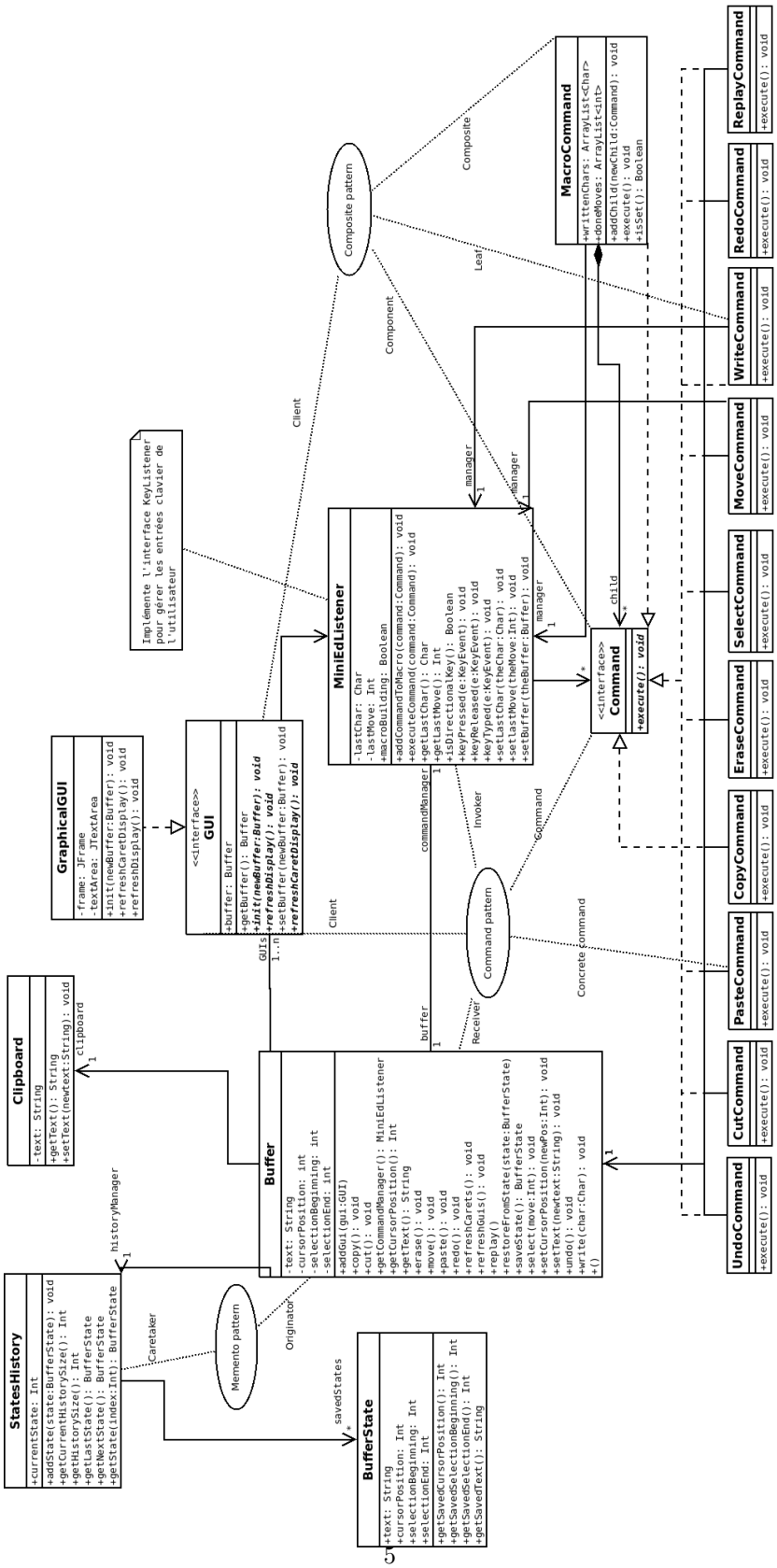
Pour la seconde version de MiniEd, le cahier des charges nous demandait de développer une nouvelle fonctionnalité. Il s'agissait de permettre l'enregistrement de toutes les actions de l'utilisateur, de façon à pouvoir les rejouer à volonté.

2.2.3 Version 3 : Le défaire / refaire

Enfin, la dernière fonctionnalité qui a été mise en place est le défaire / refaire. Il s'agit de l'équivalent des commandes "annuler" et "rétablir" que l'on trouve sur d'autres éditeurs de texte. Grâce à cette fonctionnalité, l'utilisateur peut désormais annuler pas à pas, toutes les actions qu'il a fait depuis le lancement de MiniEd (défaire), et s'il le souhaîte, il peut ensuite rétablir ce qu'il vient de défaire (refaire).

2.3 Schématisation

Dans un soucis de clarté et de compréhensibilité de notre cahier des charges, voici un diagramme UML décrivant la structure de notre application. Dans ce diagramme sont schématisées les classes nécessaires au bon fonctionnement de l'éditeur de texte ainsi que leurs dépendances.


Clipboard
-text: String
+getText(): String
+setText(newtext:String): void
clipboard
1
StatesHistory
+currentState: Int
+addState(state:BufferState): void
+getCurrentHistorySize(): Int
+getHistorySize(): Int
+getLastState(): BufferState
+getNextState(): BufferState
+getState(index:Int): BufferState
historyManager
1
Caretaker
Memento pattern
Originator
savedStates
*
BufferState
+text: String
+cursorPosition: Int
+selectionBeginning: Int
+selectionEnd: Int
+getSavedCursorPosition(): Int
+getSavedSelectionBeginning(): Int
+getSavedSelectionEnd(): Int
+getSavedText(): String
Buffer
-text: String
-cursorPosition: int
-selectionBeginning: int
-selectionEnd: int
+addGui(gui:GUI)
+copy(): void
+cut(): void
+getCommandManager(): MiniEdListener
+getCursorPosition(): Int
+getText(): String
+erase(): void
+move(): void
+paste(): void
+redo(): void
+refreshCarets(): void
+refreshGuis(): void
+replay()
+restoreFromState(state:BufferState)
+saveState(): BufferState
+select(move:Int): void
+setCursorPosition(newPos:Int): void
+setText(newtext:String): void
+undo(): void
+write(char:Char): void
+()
buffer
1
Receiver
Command pattern
Invoker
commandManager
1
Client
GUIs
1..n
GraphicalGUI
-frame: JFrame
-textArea: JTextArea
+init(newBuffer:Buffer): void
+refreshCaretDisplay(): void
+refreshDisplay(): void
<<interface>>
GUI
+buffer: Buffer
+getBuffer(): Buffer
+init(newBuffer:Buffer): void
+refreshDisplay(): void
+setBuffer(newBuffer:Buffer): void
+refreshCaretDisplay(): void
Implémente l'interface KeyListener
pour gérer les entrées clavier de
l'utilisateur
Client
MiniEdListener
-lastChar: Char
-lastMove: Int
-macroBuilding: Boolean
+addCommandToMacro(command:Command): void
+executeCommand(command:Command): void
+getLastChar(): Char
+getLastMove(): Int
+isDirectionalKey(): Boolean
+KeyPressed(e:KeyEvent): void
+KeyReleased(e:KeyEvent): void
+KeyTyped(e:KeyEvent): void
+setLastChar(theChar:Char): void
+setLastMove(theMove:Int): void
+setBuffer(theBuffer:Buffer): void
manager
1
manager
1
manager
1
Command
+execute(): void
<<interface>>
child
*
Command
Concrete Command
Composite pattern
Component
Client
Leaf
Composite
MacroCommand
+writtenChars: ArrayList<Char>
+doneMoves: ArrayList<int>
+addChild(newChild:Command): void
+execute(): void
+isSet(): Boolean
UndoCommand
+execute(): void
CutCommand
+execute(): void
PasteCommand
+execute(): void
CopyCommand
+execute(): void
EraseCommand
+execute(): void
SelectCommand
+execute(): void
MoveCommand
+execute(): void
WriteCommand
+execute(): void
RedoCommand
+execute(): void
ReplayCommand
+execute(): void

3 Développement du logiciel

3.1 Les classes

Comme il est possible de le voir sur le diagramme UML précédent, l'éditeur de texte est relativement complexe, et composé de multiples classes.

3.1.1 L'interface graphique

L'interface graphique est composée par quatre classes :
— Les classes `GUI`, et `GraphicalGUI` qui composent l'interface graphique (gestion de la fenêtre ...)
— Les classes `MiniEdListener` et `TextualGUI` qui s'occupent de l'interface textuelle (traitement des entrées clavier ...)

3.1.2 Les commandes

En ce qui concerne les commandes de l'éditeur de texte, elles ont chacune leur propre classe dépendante de la classe abstraite `Command`.
— La classe `CutCommand` : appelle la méthode `cut` de la classe `Buffer`
— La classe `PasteCommand` : appelle la méthode `paste` de la classe `Buffer`
— La classe `MoveCommand` : appelle la méthode `move` de la classe `Buffer`
— La classe `EraseCommand` : appelle la méthode `erase` de la classe `Buffer`
— La classe `SelectCommand` : appelle la méthode `select` de la classe `Buffer`
— La classe `CopyCommand` : appelle la méthode `copy` de la classe `Buffer`
— La classe `WriteCommand` : appelle la méthode `write` de la classe `Buffer`

3.1.3 Le buffer

La classe `Buffer` permet de gérer le texte de l'éditeur.

3.1.4 Le presse-papier

La classe `Clipboard` offre les fonctionnalités d'un presse-papier et permet ainsi le coupage, la copie et le collage d'une sélection de texte.

3.1.5 Macros et enregistrement

Les macros et l'enregistrement des actions de l'utilisateur sont gérées par les classes `MacroCommand`, `BufferState` et `StatesHistory`. Celles-ci permettent de mettre en place la gestion des macros utilisateur, et d'enregistrer les actions effectuées par celui-ci. Les macros sont une suite de commandes que l'utilisateur pourra définir lors de l'utilisation du logiciel, afin de les rejouer plus tard. Ici, elles sont appelées via le raccourci `ctrl + M`, qui fonctionne de la façon suivante :
— Le premier appel permet de débuter l'enregistrement de la macro.
— Le deuxième appel permet de terminer l'enregistrement de la macro. Toute action effectuée par l'utilisateur depuis le début de l'enregistrement est ainsi contenue dans la macro.

— Les appels suivants exécutent le contenu de la macro à l'emplacement du pointeur.

L'enregistrement des actions de l'utilisateur est également effectué automatiquement tout au long de l'exécution du programme. Ces actions peuvent être rejouées avec le raccourci `ctrl + P`. Cependant, aucun changement visuel ne se produit lors de cette action, le temps nécessaire à l'ordinateur pour rejouer les actions étant bien trop faible pour que l'action soit perçue.

3.1.6 Défaire / Refaire

Les fonctionnalités de défaire / refaire sont accessible grâce à deux classes : `UndoCommand` et `RedoCommand`. Elles permettent à l'utilisateur de revenir en arrière, c'est à dire d'annuler chaque étape de saisie jusqu'au début, ou de rétablir ce que l'utilisateur vient de défaire.

3.2 Patrons de conception

Afin de concevoir notre logiciel, nous avons utilisé plusieurs Design Pattern :

Le premier Pattern que nous utilisons est le Pattern Commande. Celui-ci est un patron de conception de type comportemental qui encapsule la notion d'Invocation. Il permet de séparer le code de l'appel des commandes, avec le code des commandes elles-mêmes.

Le second Pattern que nous utilisons est le Pattern Composite. Ce pattern est un patron de conception structurel qui permet de simplifier l'utilisation de plusieurs objets similaires. Dans notre cas, nous avons utilisé le Pattern Composite pour concevoir une macro permettant la stockage de commandes.

Le troisième Pattern que nous utilisons est le pattern Memento. Il permet de conserver l'état d'un objet pour le restaurer ultérieurement. Ce Pattern a été utilisé afin de réaliser l'enregistrement des actions effectuées par l'utilisateur. En effet, à chaque action l'état du buffer est enregistré en mémoire, afin d'établir un historique.

3.3 Tests et fiabilité

Aujourd'hui, la qualité d'un logiciel se juge autant sur sa fiabilité que sur ses compétences. C'est pourquoi nous avons développé plusieurs classes de test améliorant la fiabilité de notre logiciel. Dans ce but, nous avons mis en place des tests unitaires à l'aide de JUnit, ce qui permet de tester le bon fonctionnement de nos classes et de leurs méthodes.

4 Conclusion

Malgré sa simplicité, MiniEd reste un projet d'apprentissage. Ce travail nous a donc enseigné plusieurs choses, comme par exemple le langage Scala, tout en nous permettant de mettre en application les différentes notions étudiées en Génie Logiciel. Nous avons conçu MiniEd depuis la base et mis en place les concepts d'UML, de multiclasses et de Design Pattern au service de notre programme.

De plus, le fait de séparer la conception en trois versions fonctionnelles de l'éditeur, nous a aidé à réaliser un logiciel fiable et performant. En effet, chaque version se devait d'être fonctionnelle et de répondre précisément au cahier des charges, ce qui impliquait une vérification et des tests pour chaque rendu.

MiniEd nous a permis d'utiliser de nouvelles connaissances étudiées en Génie Logiciel, mais pas seulement. Nous nous sommes servis pour ce projet d'autres notions vues dans d'autres modules d'enseignement, comme *Concepts et Outils de Développement*, ou *Vérifications et Tests*.