Scala Basic Editor

Projet de Génie Logiciel

https://github.com/masterALMA2016/scalaBasicEditor

Anthony Pena Jérémy Bardon

Sommaire

1 Introduction

Durant ce projet nous avons mis en place une structure permettant de réaliser un éditeur de texte. Celle-ci a été définie à l'aide de l'outil Papyrus [1] et ensuite implémenté en Scala [2]. Les classes et leurs méthodes ont été testées via des tests unitaires JUnit afin de valider leurs fonctionnements et limiter les bugs.

2 La structure générale

2.1 Editeur

La classe éditeur est une classe assez basique, elle est utilisée pour contenir les différents éléments liés à un éditeur selon les spécifications qui ont été définies. Elle contient principalement deux `Buffer` (voir 2.2) – un pour la zone de texte et un pour le presse-papier –, un `Curseur` (voir 2.3) et une `Macro` (voir 4).

2.2 Les buffers

La classe `Buffer` est une classe générique de gestion d'un buffer de texte avec les fonctionnalités classiques comme l'ajout de texte – à la fin ou à une position donnée – le remplacement d'une portion de texte, la suppression, etc. Dans cette implémentation d'éditeur, le `Buffer` est utilisé sous la forme de 2 instances : une pour la « Zone de texte » et une pour le « Presse-papier ». Les deux éléments ayant des caractéristiques proches (stockage du texte et manipulation de base), il était plus simple de créer une seule classe pour les deux.

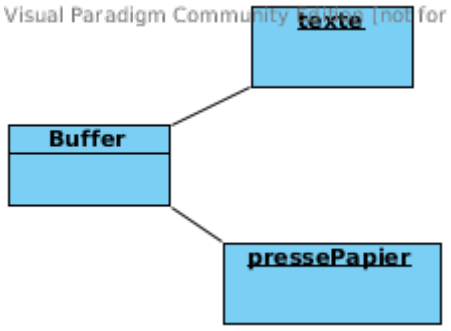

FIGURE 1 – Diagramme d'objet des deux instances de Buffer

2.2.1 Zone de texte

La zone de texte est la partie où sera stocké le texte saisie par l'utilisateur. Si on avait développé une interface graphique, cette objet deviendrait une copie côté moteur du contenu affiché à l'utilisateur.

1. Un plugin pour Eclipse, plus d'information ici : `http://www.eclipse.org/papyrus/`
2. Langage de programmation fonctionnel, exécuté sur JVM, plus d'information ici : `http://scala.com/fr/`

2.2.2 Presse-papier

Le presse-papier est un buffer dans lequel est stocké temporairement le contenu copié par l'utilisateur. Ici nous avons choisi de créer notre propre presse-papier pour simplifier l'implémentation. Si nous avions réalisé complètement l'éditeur nous aurions utilisé à la place l'interface système et son presse-papier afin qu'il soit commun à tous les logiciels.

2.3 Le curseur

Un `Curseur` est utilisé pour marquer la position à partir de laquelle le texte saisi doit être ajouté. Le `Curseur` peut aussi servir à définir une sélection. En interne, le `Curseur` possède deux positions – `début` et `fin` – lorsqu'on est en « mode saisie » (pas de sélection), `fin` est passé à -1 mais dans le cas où une sélection est faite, on assigne une valeur à `fin`.

Nous avons choisi d'implémenter la sélection comme une action en deux étapes : la première où on place le curseur à une extrémité de la sélection voulue, et la seconde qui consiste à glisser le curseur vers l'autre extrémité de la sélection. Dans le cas où on fait une sélection « négative » (la seconde extrémité est à gauche du curseur), la sélection est valide et les positions sont inversées.

3 Les fonctionnalités

Pour l'implémentation des différentes fonctionnalités nous avons choisi d'utiliser le `Pattern Commande` [3]. Il nous a permis de séparer données et comportements, ce qui était nécessaire au vu du nombre de fonctionnalités, afin d'éviter d'avoir une classe "dieu" qui ferait tout ou presque. Ce découpage permet aussi une plus grande évolutivité puisqu'une fonctionnalité est contenue dans une classe elle devient par nature facilement extensible pour améliorer son comportement ou en ajouter de nouveaux. Cela donne aussi la possibilité d'effectuer, de manière isolée, des tests sur les différentes fonctionnalités.

Le pattern commande comprend 4 éléments :

- **Commande** est une interface que doivent implémenter les classes de commande.
- **Commande réelle** est un ensemble de classes qui implémente la classe Commande.
- **Invocateur** est la classe sur laquelle on va appeler toutes les commandes comme de simples méthodes. En interne, ce seront les commandes réelles qui seront appelées.
- **Récepteur** est la classe sur laquelle va être appliqué les commandes.

Dans notre implémentation nous avons utilisé un vocabulaire plus spécifique au projet. Nous avons remplacé commande par `Action`, les commandes réelles sont les classes `Copier`, `Coller`, `Effacer`, `Selectionner`, `Inserer`, `Deplacer`, `Remplacer` et `DeplacerCurseur`, et pour finir le récepteur est la classe `Editeur`.

4 Les Macros

Pour les macros nous avons choisi d'utiliser le Pattern Composite [4], qui est approprié du fait qu'une macro n'est qu'une action composée d'une succession d'actions.

Dans l'implémentation actuelle, le système ne prend qu'une macro en compte, ce qui est suffisant pour tester le fonctionnement des macros, mais on pourrait envisager de permettre l'enregistrement de plusieurs macros. On pourrait aussi donner la possibilité de composer les

3. Une définition complète est disponible à cette adresse : `https://fr.wikipedia.org/wiki/Commande_(patron_de_conception)`

4. Une définition complète est disponible à cette adresse : `https://en.wikipedia.org/wiki/Composite_pattern`

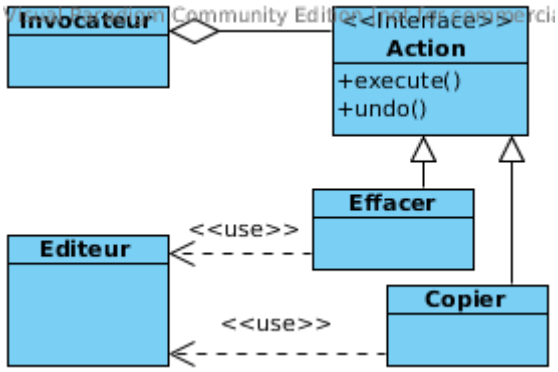

FIGURE 2 – Diagramme de classe des fonctionnalités

macros entre-elles, une macro étant défini comme une action à travers le Pattern Composite, cela ne pose aucun problème.

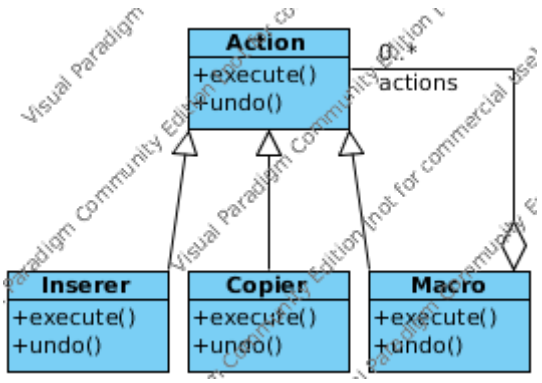

FIGURE 3 – Diagramme de classe des macros

En utilisant ce pattern, on ouvre la possibilité d'étendre la liste des actions possibles sans avoir à modifier la classe `Macro` du fait qu'une macro est composée d'une série d'action, sans connaitre l'implémentation de chaque action. Il suffit donc que chaque nouvelle action hérite de la classe `Action` ou d'une de ses sous-classes (c'est le cas de `Coller` qui hérite de `Insérer`, la logique étant différente mais la mécanique identique).

5 Évolutions

5.1 Interface

Pour aller plus loin dans ce projet, il pourrait être intéressant de créer une interface d'édition. Actuellement seul des tests vérifient que les différents éléments sont fonctionnels mais une interface en ligne de commande – via ncurses par exemple – ou graphique permettrait de vérifier que l'implémentation actuelle est fonctionnelle et le projet deviendrait alors utilisable.

5.2 Nouvelles fonctionnalités

Il pourrait être intéressant d'étendre les possibilités de l'éditeur. Par exemple en ajoutant la possibilité de sauvegarder plusieurs macros et en permettant de leurs assigner des raccourcis clavier. On pourrait aussi ajouter un fichier de configuration contenant des snippets [5] de code.

6 Conclusion

Tout au long de ce projet nous avons pu mettre en pratique les compétences acquises au cours du module de Génie Logiciel et ainsi mettre en application une partie des éléments théoriques comme les Design Pattern dans un projet concret.

Ce projet nous a aussi permis de découvrir un nouveau langage de programmation et d'utiliser un maximum ces capacités pour mettre en oeuvre notre projet. Le langage Scala permet de produire des binaires portables compatibles avec la JVM, tout en réduisant considérablement la longueur du code, ce qui augmente sa clareté. Nous avons aussi mis en application le module de Tests et Vérifications à travers un certain nombre de tests unitaires, qui nous ont permis de vérifier le bon fonctionnement de nos classes.

5. `https://en.wikipedia.org/wiki/Snippet_(programming)`