Description de l'architecture du projet

I- Les bibliothèques

J'ai utilisé 3 bibliothèques afin de réaliser mon application. J'ai tout d'abord utilisé JUnit afin de réaliser mes tests unitaires. Je m'en suis servi quand j'ai utilisé mes différents patterns afin que ça me permette d'aller plus vite dans leur implémentation.

J'ai ensuite utilisé la bibliothèque JFoenix. Cette bibliothèque rentre totalement dans ma philosophie de faire du material design en interface. Je pense qu'il faut faire des interfaces épurées pour que l'application soit sympa a utilisé. J'ai donc utilisé cette bibliothèque magique qui m'a permis de faire deux datepicker : un qui m'envoie le time et l'autre la date. En plus, l'interface de ces datepicker est très classe, ce qui est essentiel dans l'adoption d'une application.

Pour finir, j'ai utilisé une autre bibliothèque qui se nomme TrayNotification. C'est grâce à elle que mon programme envoie une notification très sympa visuellement à l'utilisateur. C'est une bibliothèque super complète qui permet d'avoir des notifications personnalisées allant de l'apparence à son temps d'affichage, etc…

II- Les patterns

Je me suis fait vraiment plaisir sur cette partie. Effectivement, j'ai réalisé 3 patterns très utiles dont je vais vous expliquer ici :

Le premier que je souhaite évoquer est le pattern commande. 3 lignes de codes me permettent d'utiliser ce pattern dans mon Controller. C'est magique.

// dans le controller :

Invoker control = new Invoker();
Receiver receiver = new Receiver();

//dans la méthode commandNotif :

Command cmd = new ActivateNotification(receiver,titre,message);
control.setCommand(cmd);
control.pressButton();

Ce pattern est composé d'une classe Recepteur que j'ai nommé ici Receiver, d'une classe Invocateur que j'ai nommé ici Invoker, d'une interface Command que j'ai nommé Command ainsi que d'une ConcreteCommand nommée ici : ActivateNotification.

Ce pattern : « encapsule une requête comme un objet, autorisant ainsi le paramétrage des clients par différentes requêtes, files d'attente et récapitulatifs de requêtes, et de plus, permettant la réversion des opérations. » (d'après goprod.bouhours.net).

J'utilise également un pattern Singleton. Ce pattern je l'utilise avec ma classe Converter. Lorsque je récupère la date et l'heure bindée sur mon datePicker dans mon élément, j'obtiens des jours et des minutes… Afin de réaliser une notification d'application à un temps voulue, je suis obligé de convertir cette période en une période en milliseconde. J'utilise donc un convertisseur. Afin d'optimiser mon application, j'ai décidé que mon objet Converter dispose d'une seule et unique instance. Effectivement, une seule instance suffit à faire toutes mes conversions nécessaires pour le temps de toutes les notifications qui devront s'exécuter. Deux lignes de code dans le Controller suffisent.

```
Converter cv = Converter.getInstance();
temps=cv.miliSec(date, time);
```

Pour finir, je souhaite parler de ce fameux pattern Memento. Celui que j'ai mal implémenté dans le code que je vous ai présenté durant mon oral. Celui qui m'a fait plancher des heures entières sur mon clavier d'ordinateur. Je peux enfin dire que j'ai pris ma revanche dessus. J'ai réussi à placer ce pattern Memento pour mon action Undo et Redo.

```
newV.texteProperty().addListener((oVal,oldVal,newVal)->{

String texteActuel=listeDesElements.getSelectionModel().getSelectedItem()
.getTexte();
if (texteActuel.charAt(texteActuel.length()-1) == ' ') {

Integer emplacement=listeDesElements.getSelectionModel()
.getSelectedIndex();
originator.setState(texteActuel);
originator.setNum(emplacement);
careTaker.add(originator.saveStateToMemento());
Integer nbActual=listeDesElements.getSelectionModel() .getSelectedItem()
.getNombreSauv();
listeDesElements.getSelectionModel().getSelectedItem().setNombreSauv(
nbActual+1);

}

});
```

Dans mon controller, je vais tout d'abord écouter si ma textProperty va changer, c'est-à-dire si on écrit à l'intérieur **en rouge.**

Puis à chaque fois que le dernier élément écrit à l'intérieur est un espace **en jaune**, je vais le sauvegarder en 3 lignes de codes : avec un originiator qui va dans un premier lieu programmer à la fois le texte présent et l'emplacement et un caretaker qui dans un second lieu va ajouter le texte dans un structure à un emplacement précis **en bleu.**
Pourquoi l'emplacement ? Comme précisé précédemment, je travaille avec un master détail, nous avons donc pour chaque élément de la liste une position. J'ai donc créer dans mon code behind une structure (une map) qui va stocker l'emplacement en clé et en valeur une ArrayList pour les éléments ajoutés peu à peu. L'emplacement permet au bouton undo et redo de ne pas faire n'importe quoi en fonction de la note utilisé.

Maintenant il ne reste plus qu'a modifié la propriété de notre élément pour lui indiquer le nombre de sauvegarde qu'il y a eu pendant sa saisie **en vert.**

Dans le code de la fonction undo, nous avons juste besoin de deux lignes de code afin d'utiliser ce pattern, ces lignes sont notées **en bleu.** La première permet de récupérer l'élément présent dans la structure à l'emplacement précis (cet emplacement a été donné en fonction du positionnement dans la liste). Nous utilisons les commandes **en vert** pour gérer le nombre de sauvegarde qui est une propriété d'un élément.

```
String Integer nbActual=listeDesElements.getSelectionModel()
.getSelectedItem() .getNombreSauv();

if (nbActual >0) {
listeDesElements.getSelectionModel().getSelectedItem()
.setNombreSauv(nbActual-1);
Integer emplcamnt = listeDesElements.getSelectionModel().
getSelectedIndex();

originator.getStateFromMemento(careTaker.get(nbActual-
1,emplcamnt));
listeDesElements.getSelectionModel().getSelectedItem().setTexte(origina
tor.getState());

}
```

Je tiens à rappeller que pour ce pattern : l'Auteur se nomme l'Orignator, le Surveillant se nomme Caretaker et le Memento est le Memento.

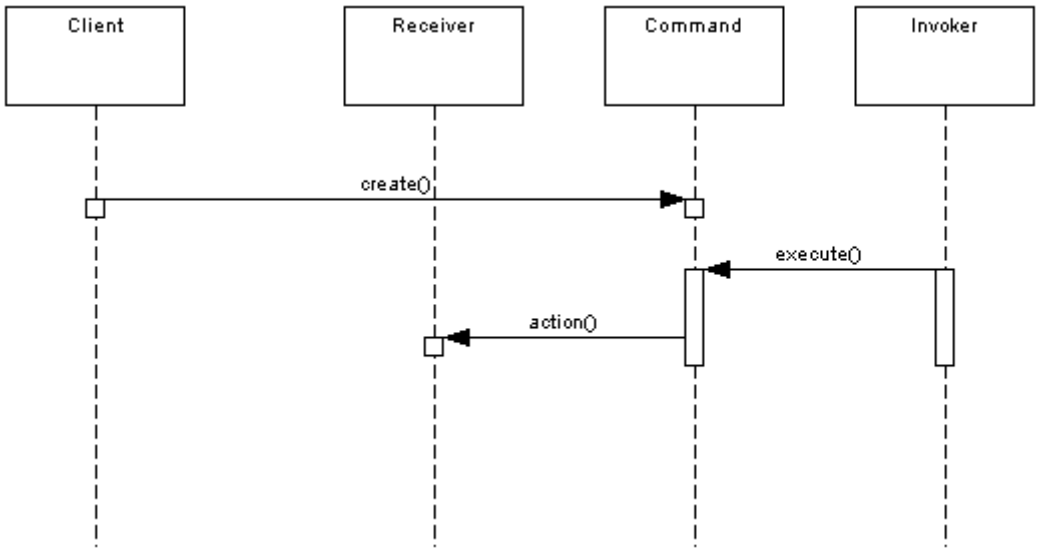
Pour finir comme précédemment, nous utilisons assez souvent la fonction addListener qui nous permet d'écouter quand un élément change. Cela est possible grâce à un Observateur.

Source : Je tiens quand même à référencer le site DZone qui m'a aidé à implémenter deux des trois patterns

https://dzone.com/articles/design-patterns-memento

https://dzone.com/articles/design-patterns-command

Voici comment fonctionne le pattern Command :



Avec son diagramme de classe :

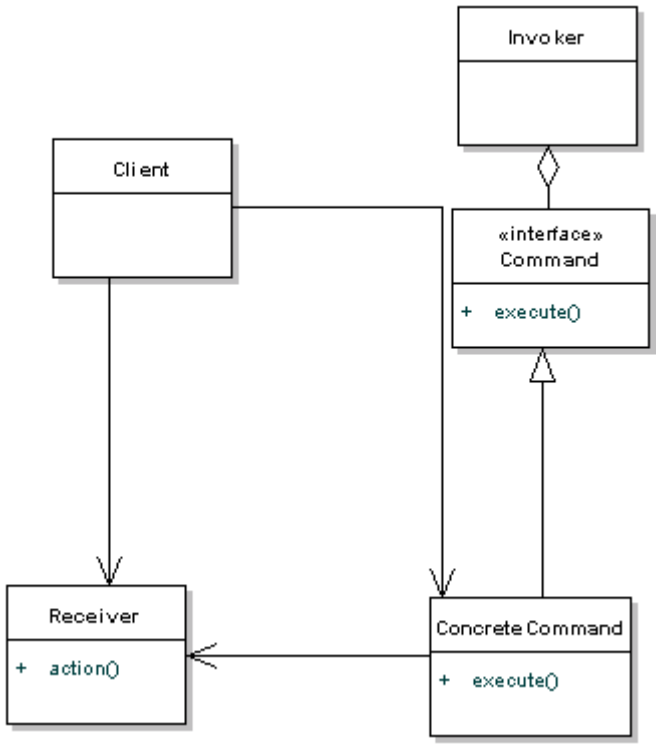


Diagramme de classe du pattern Memento :