TP : Actions, Undo/Redo, nouveaux composants

Objectifs

- se familiariser avec les actions et les opérations annulation / restauration,
- se familiariser avec la création de nouveaux composants.

1 Actions

Question 1. En partant du fichier UndoGUI.java [1], associez une action à chaque bouton et chaque item du menu Edition. Par défaut les boutons Annuler et Refaire seront désactivés. Associez aussi une action au champ texte pour ajouter un item dans la liste en appuyant sur la touche entrée (quand le champ texte a le focus) ou en cliquant sur le bouton Ajouter. Associez comme bon vous semble des raccourcis (voire des icônes) à chaque action. Vous déclarerez les actions correspondantes à l'ajout, la suppression, l'annulation et la restauration comme classes internes privées internes de la classe UndoGUI. La méthode `setEnabled` de la classe `AbstractAction` permet de désactiver une action et par conséquent de griser un bouton par exemple.

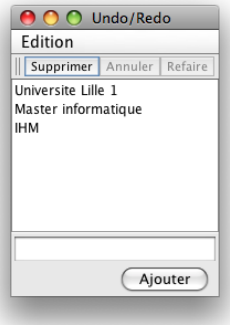

FIGURE 1 – Illustration des actions et de undo / redo.

2 Undo/Redo

Question 2. Créez deux classes, `AddUndoableListe` et `DelUndoableListe`, qui héritent de AbstractUndoableEdit [2]. `AddUndoableListe` conservera les informations nécessaires pour annuler et restaurer une opération d'ajout sur la liste. Vous devez pour cela enregistrer une référence du modèle associé à la liste (`DefaultListModel`), l'élément qui a été ajouté et la position à laquelle il a été ajouté. Vous redéfinirez les méthodes `undo` et `redo` pour utiliser ces informations afin d'annuler et restaurer l'opération d'ajout. Réalisez le même type de travail pour la classe `DelUndoableListe`.

Question 3. Après avoir ajouté une variable d'instance de type `UndoManager` à la classe `UndoGUI`, modifiez les méthodes `actionPerformed` de vos actions pour enregistrer les opérations annulables. Pour les actions Supprimer et Ajouter, vous utiliserez la méthode `addEdit` de UndoManager [3] pour enregistrer une opération annulable (utiliser `AddUndoableListe` et `DelUndoableListe`) dans l'historique d'opérations annulables de `UndoManager`. Pour les actions Annuler et Refaire, vous utiliserez les méthodes `undo` et `redo` de UndoManager.

1. `UndoGUI.java`
2. `http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/undo/AbstractUndoableEdit.html`
3. `http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/undo/UndoManager.html`

Question 4. Les boutons Annuler et Refaire restent grisés même si les opérations annulables sont bien enregistrées par le UndoManager [4]. Ajoutez une méthode `updateUndoRedoButtons` à la classe `UndoGUI` pour mettre à jour l'état d'activation des boutons en fonction des possibilités d'annulation et de restauration offertes par le `UndoManager`. Les méthodes `canUndo` et `canRedo` de la classe `UndoManager` permettent d'accéder à ces états. Appelez cette méthode pour les actions où cela est nécessaire.

Question 5. Ajoutez la méthode `getPresentationName` dans vos classes `AddUndoableListe` et `DelUndoableListe` qui retourne une chaîne de caractères représentant l'opération. Modifiez la méthode `updateUndoRedoButton` pour prendre en compte cette information sous forme de tooltip pour les boutons Annuler et Refaire. Utilisez pour cela la propriété SHORT_DESCRIPTION des actions et la méthode `getUndoPresentationName` de `UndoManager` pour obtenir la chaîne de caractères correspondant à l'action Undo.

Question 6. Nous allons maintenant ajouter le Drag and Drop à la liste pour importer/exporter des items de la liste mais également réarranger la liste. Ajoutez le support au Drag and Drop en utilisant la classe `TransfertListe` définie dans le fichier TransfertHandlerListe.java [5]. Utilisez le DropMode INSERT sur la liste.

Pour que les actions Annuler et Refaire gardent un sens, il est également nécessaire de supporter les actions d'annulation/restauration lors des transferts. Pour y parvenir, une solution est de passer une instance de `UndoManager` au `TransferHandler`.

Question 7. Utilisez cette méthode pour mémoriser l'historique des opérations de transfert (nous nous contentons pour l'instant de traiter uniquement le transfert de données entre la liste et des composants externes. La réorganisation de la liste sera traitée à la question 9). Notez que pour mettre à jour l'état d'activation des boutons Annuler et Refaire, il est nécessaire soit de passer les actions au `TransferHandler` ou de déclarer le `TransferHandler` en tant que classe interne de la classe `UndoGUI`.

Une autre solution est d'attacher le `UndoManager` à la `JList` en tant que `listener`. La classe `JList` ne supportant pas ce type de listener par défaut (contrairement aux composants texte qui héritent de la classe AbstractDocument [6]), il est nécessaire de créer une classe qui hérite de JList et qui définit les méthodes permettant d'attacher des listeners de type UndoManager :

```
class JListe extends JList {

        JListe(DefaultListModel listModel) {
                super(listModel);
        }

        public void addUndoableEditListener(UndoableEditListener listener){
                listenerList.add(UndoableEditListener.class,listener);
        }

        public void removeUndoableEditListener(UndoableEditListener listener){
                listenerList.remove(UndoableEditListener.class,listener);
        }
        public void fireUndoableEditUpdate(UndoableEditEvent e){
                Object[]listeners=listenerList.getListenerList();
                for(int i=listeners.length-2;i>=0;i-=2){
                        if(listeners[i]==UndoableEditListener.class)
                        ((UndoableEditListener)listeners[i+1]).undoableEditHappened(e);
                }
        }
}
```

4. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/undo/UndoManager.html
5. TransfertHandlerListe.java
6. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/text/AbstractDocument.DefaultDocumentEvent.html

Il est désormais possible dans la classe TransferHandler d'émettre un message qui sera reçu par le UndoManager, en utilisant la méthode `fireUndoableEditUpdate` :

```
UndoableEditEvent ue = new UndoableEditEvent(this,new AddUndoableListe(listModel,
                                                    data, insertedindex));
list.fireUndoableEditUpdate(ue);
```

A la réception du message, la méthode `undoableEditHappened` de la classe `UndoManager` ajoute alors l'opération dans l'historique.

Question 8. Mettez en oeuvre cette deuxième solution et mettez à jour les états d'activation des boutons Annuler et Refaire dans la méthode `undoableEditHappened`. Pour cela, écrivez une classe qui hérite de `UndoManager` et redéfinissez la méthode `undoableEditHappened`.

Il nous reste à traiter l'opération de réorganisation de la liste. Une solution pourrait être d'ajouter la prise en charge d'une opération à la classe `UndoableListe` pour gérer le déplacement d'un élément dans la liste. Cette opération peut être vue comme la suppression d'un élément et l'insertion d'un autre élément, ce qui correspond à une action composée.

Question 9. Utilisez CompoundEdit [7] pour créer un événement composé de la suppression et de l'insertion d'un élément pour gérer le drag and drop dans la liste. N'oubliez pas d'utiliser la méthode `end` de `CompoundEdit` pour préciser que l'opération composée est achevée. Un clic sur le bouton Annuler doit alors permettre de remettre à sa position initiale un élément qui a été déplacé.

Question 10. Remarquez que vous avez certainement des comportements erratiques quand vous combinez des réorganisations de la liste et des importation/exportation de données de la liste. Proposez une méthode pour résoudre ce problème.

3 Création d'un nouveau composant : le `JRangeSlider`

Nous proposons de créer un nouveau composant, que nous appellerons le `JRangeSlider`, qui permet de sélectionner un intervalle de valeurs plutôt qu'une seule valeur comme c'est le cas pour le `JSlider`.

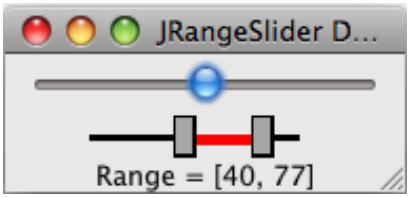

FIGURE 2 – Le JRangeSlider.

Pour respecter l'architecture Swing [8], notre `JRangeSlider` doit comporter :

- Une classe qui fournit l'API pour créer, modifier et obtenir l'état de base du composant. Nous appellons cette classe `JRangeSlider`. Cette classe est fortement inspirée de la classe JSlider [9].
- L'interface du modèle et l'implémentation par défaut du modèle qui prend en charge la logique du composant et les notifications. Nous nous contentons ici de créer la classe qui implémente le modèle que nous appelons `RangeSliderModel`. Cette classe est fortement inspirée de la classe BoundedRangeModel [10].
- L'UI delegate qui prend en charge le placement des différentes parties du composant, la gestion des événements et l'affichage. Nous appelons cette classe `BasicRangeSliderUI`. Elle est fortement inspirée de BasicSliderUI [11].

7. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/undo/CompoundEdit.html
8. http://java.sun.com/products/jfc/tsc/articles/architecture/
9. JSlider.java
10. BoundedRangeModel.java
11. BasicSliderUI.java

Les fichiers JRangeSlider.java[12], RangeSliderModel.java[13] et BasicRangeSliderUI.java[14] fournissent une version de base de `JRangeSlider`. Le fichier GUITest.java[15] permet de tester le composant, comme illustré figure 2.

Question 11. Etudiez ces différentes classes pour en comprendre le fonctionnement. Seule la poignée de gauche est manipulable. Rendez la poignée de droite manipulable également (seul le fichier BasicRangeSliderUI.java est à modifier).

12. `JRangeSlider.java`
13. `RangeSliderModel.java`
14. `BasicRangeSliderUI.java`
15. `GUITest.java`