TP : Gestion des événements

Objectifs
- se familiariser avec la gestion des événements.

1 Avant propos

La classe à la base de la hiérarchie des événements est la classe EventObject [1] qui se trouve dans le package java.util. Attention : Ne confondez pas avec la classe Event du package java.awt [2], qui était utilisée dans la première gestion des événements proposée par Sun (JDK1.0) et qui est maintenue uniquement dans un souci de compatibilité ascendante.

2 Où et comment gérer les événements ?

Le but de cette partie est d'étudier les différentes manières de gérer l'événement de clic sur un bouton. Nous utiliserons une interface avec un bouton qui permet d'incrémenter la valeur d'un label.

Question 1. En utilisant la classe `JFrame`, le gestionnaire de placement `BoxLayout`, les classes `JLabel` et `JButton`, réalisez une interface similaire à celle présentée figure 1

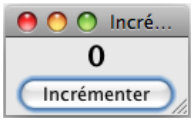

FIGURE 1 – Application incrémenter

JButton [3] possède une méthode `addActionListener` qui lui permet de s'abonner à l'événement "clic sur le bouton de la souris". Le paramètre de cette méthode est un objet de type ActionListener [4]. ActionListener est une interface qui doit être implémentée. La seule méthode à implémenter est la méthode `actionPerformed` qui est la méthode qui sera appelée quand un clic aura lieu.

Question 2. Ecrivez une classe `ReponseAuClic` qui implémente la méthode `actionPerformed` de l'interface `ActionListener` pour afficher "Clic sur le bouton" sur la console quand la méthode est appelée. Abonnez votre bouton à cette classe et vérifiez que le message s'affiche bien à l'écran à chaque fois que vous cliquez sur le bouton.

Nous souhaitons que la méthode `actionPerformed` modifie le texte du label. Pour que le listener puisse modifier cet objet, il est nécessaire de passer en paramètre du constructeur `ReponseAuClic` une référence de l'objet JLabel qu'il devra modifier.

Question 3. Modifier votre classe `ReponseAuClic` en ajoutant le JLabel en paramètre du contructeur et en modifiant la méthode `actionPerformed` pour récupérer et modifier le texte du label. Vous utiliserez la méthode `getText()` pour récupérer le texte du label et la méthode Integer.parseInt [5] pour convertir la chaîne de caractères en entier. La méthode `setText()` permet de modifier le texte du label.

En suivant ce principe, l'écriture d'applications simples va demander l'écriture d'un nombre important de classes de listener, ce qui peut rapidement devenir difficile à gérer. On remarque aussi que certains listeners sont étroitement liés au composant source et ne seront probablement jamais réutilisés pour un autre composant. Une autre solution est d'utiliser les classes internes.

1. http://java.sun.com/javase/6/docs/api/java/util/EventObject.html
2. http://java.sun.com/javase/6/docs/api/java/awt/event/package-summary.html
3. http://java.sun.com/javase/6/docs/api/javax/swing/JButton.html
4. http://java.sun.com/javase/6/docs/api/java/awt/event/ActionListener.html
5. http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Integer.html

2.1 Classes internes

Depuis le JDK1.1, il est possible d'imbriquer les définitions de classes les unes dans les autres. Une classe définie à l'intérieur d'une autre est appelée classe interne. Cette classe est généralement déclarée privée pour limiter son accès à la classe englobante.

Cette façon de faire permet de simplifier l'écriture du listener car nous n'avons plus besoin de passer le JLabel en paramètre du constructeur. La classe privée a en effet accès aux membres, même privés, de la classe englobante.

Question 4. Ecrivez une classe `FenetreIncrementer` avec le JLabel déclaré comme membre privé. Reprenez et modifiez la classe `ReponseAuClic` écrite à la question précédente en l'incluant dans la classe `FenetreIncrementer` (vous la définirez comme classe privée).

La classe `ReponseAuClic` n'étant utilisée qu'une seule fois, il est possible de la définir directement à l'endroit où une instance de cette classe est nécessaire. Par ailleurs, puisque cette classe est amenée à n'être utilisée qu'une seule fois, il est même inutile de lui donner un nom explicite. Nous pouvons dans ce cas utiliser une classe interne anonyme.

2.2 Classes internes anonymes

Le mécanisme des classes internes anonymes est adapté au cas de figure où l'on a besoin à un instant donné d'une instance de classe, et on est certain de ne pas avoir à créer d'autres instances ailleurs, ni à référencer cette classe plus loin. Le mécanisme consiste à appeler le constructeur et à construire en même temps la classe. L'appel au constructeur ne nécessite pas le nom de la classe : on indique le nom de la classe héritée ou de l'interface implémentée par la classe interne anonyme.

Par exemple, la déclaration d'une classe interne anonyme pour la gestion du clic sur le bouton donne quelque chose comme ceci :

```
lebouton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        /**
                         * Mise a jour du label
                         */
                    }
                });
```

L'instruction `new ActionListener()` instancie un objet d'une classe qui n'a pas de nom, autrement dit est anonyme ; dérive de `Object` ; et implémente l'interface `actionPerformed`. L'instance a été créée en utilisant le constructeur par défaut, c'est-à-dire le constructeur sans paramètre.

De la même manière il est possible de créer des classes internes anonymes dérivées d'autres classes.

Pour éviter de rendre difficile la lecture du code, il est d'usage d'appeler une méthode de la classe englobante pour chaque méthode implémentée dans la classe anonyme :

```
lebouton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        /**
                         * Mise a jour du label
                         */
                        miseAJourDuLabel(e);
                    }
                });
```

Question 5. Testez l'utilisation des classes internes anonymes dans votre exemple précédent.

Cette méthode de gestion des événements qui est utilisée dans les outils de génération d'interfaces comme NetBeans.

2.3 Les adapters

Pour fermer la fenêtre en cliquant sur le bouton de fermeture, nous avons vu précédemment qu'il est possible d'utiliser la méthode `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)` afin de provoquer la fin de l'exécution du programme quand cet événement survient. Pour gérer soi même cet événement, il est possible d'écrire une classe qui implémente l'interface WindowListener [6] avec la méthode `windowClosing`. L'inconvénient est que cette interface spécifie sept méthodes et nous souhaitons uniquement implémenter la méthode `windowClosing`.

Pour éviter d'avoir à implémenter six méthodes vides, Java fournit des classes d'implémentation par défaut des listeners. Ce sont les `adapters`. Les adapters implémentent des versions vides de toutes les méthodes de l'interface. Il nous suffit donc d'hériter de la classe WindowAdapter [7] et de redéfinir seulement la méthode nécessaire.

Question 6. Ecrivez une classe `FermetureFenetre`, qui hérite de `WindowAdapter`, et qui redéfinit `windowClosing` pour afficher le message "Fenetre en cours de fermeture" avant de terminer l'exécution du programme par un `System.exit(0)`.

3 La méthode `getSource()`

Créer une classe par événement peut conduire à un nombre de classes important et par conséquent augmenter le temps de chargement de l'application et l'espace mémoire utilisé. Il peut donc être utile de gérer les événements de plusieurs composants à l'intérieur d'une même classe.

Chaque méthode d'un gestionnaire d'événement a un seul argument qui est un objet qui hérite de la classe EventObject [8]. La classe `EventObject` définit une méthode très utile, `getSource()`, qui donne l'objet à l'origine de l'événement.

Question 7. Réalisez une interface similaire à celle de la figure 2, où le clic sur un bouton affiche son nom dans le label supérieur. Vous utiliserez une seule classe pour gérer les événements des trois boutons et la méthode `getSource()` pour identifier le bouton qui a reçu l'événement.

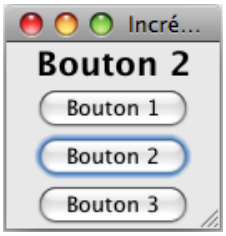

FIGURE 2 – Utilisation de getSource() pour gérer les événements de plusieurs objets dans une seule classe.

4 Application : Ardoise magique

4.1 La classe `Graphics`

Graphics [9] est une classe qui permet de dessiner en tracant des trais, cercles, arc de cercles dans une interface.

Chaque composant possède une instance de Graphics pour se dessiner. Pour obtenir une instance de `Graphics` sur une instance de `Component`, il faut utiliser la méhode `getGraphics()`. Cette instance représente un rectangle dans lequel il est possible de dessiner. La taille de ce rectangle est la taille réelle du composant. `Graphics` possède un système de coordonnées géométriques orthogonales dont l'origine se situe en haut à gauche du rectangle dont nous venons de parler. Pour que les modifications apportées sur

6. http://java.sun.com/javase/6/docs/api/java/awt/event/WindowListener.html
7. http://java.sun.com/javase/6/docs/api/java/awt/event/WindowAdapter.html
8. http://java.sun.com/javase/6/docs/api/java/util/EventObject.html
9. http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Graphics.html

l'objet `Graphics` puissent persister après le rafraîchissement, il faut écrire une sous classe du composant et redéfinir la méthode `paintComponent` qui est appelée par Swing pour dessiner le composant. Un exemple pour dessiner un cercle dans un JPanel [10] :

```
public class MonPanel extends JPanel {
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                g.drawOval(10,10,50,50);
        }
}
```

4.2 Mise en oeuvre

Question 8. En utilisant la classe `ArdoiseMagique` définie dans le fichier ArdoiseMagique.java [11], construisez une interface qui permet de dessiner des traits en utilisant le bouton gauche de la souris. Un appui sur le bouton droit permettra d'effacer le dessin. Notez que la classe définie dans le fichier ArdoiseMagique.java n'applique pas le patron de conception MVC puisqu'il n'y a pas de séparation nette entre le Modèle et la Vue (cela fera l'objet du prochain TP). Note : Utilisez la méthode `getInsets` de JFrame pour obtenir la hauteur de la barre de décoration de la fenêtre et ainsi faire correspondre la position du pointeur à celle du trait tracé. Attention : veillez à appeler la méthode après avoir appelé la méthode `pack()` de JFrame.

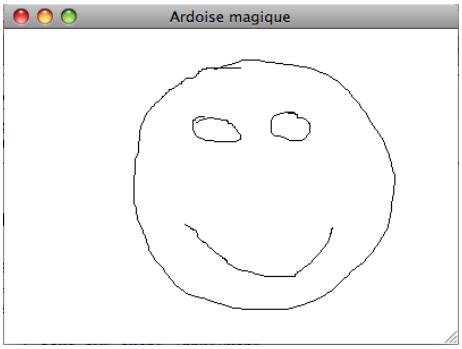

FIGURE 3 – Ardoise magique.

5 Sélecteur de couleur

Le but de cet exercice est de réaliser une interface qui permet de choisir une couleur en spécifiant chacune de ses composantes en utilisant soit les potentiomètres, les champs de textes décimaux ou le champ hexadécimal. La modification de l'état d'un composant doit modifier les autres composants en conséquence.

Question 9. En utilisant les classes JSlider [12], JTextField [13] et `JPanel`, réalisez une interface similaire à la figure ci-dessous.

Question 10. Gérez les événements liés aux différents composants en utilisant les méthodes de votre choix parmi celles présentées précédemment. La méthode toHexString [14] (classe Integer) permet de convertir un entier en chaine hexadécimale et la méthode parseInt [15] (classe Integer) permet de convertir une chaîne de caractères en entier. Pour gérer la couleur, vous pourrez utiliser la classe Color [16]. La méthode

10. http://java.sun.com/javase/6/docs/api/javax/swing/JPanel.html
11. ArdoiseMagique.java
12. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JSlider.html
13. http://java.sun.com/javase/6/docs/api/javax/swing/JTextField.html
14. http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Integer.html#toHexString(int)
15. http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Integer.html#parseInt(java.lang.String)
16. http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Color.html

decode[17] vous permettra de convertir une chaine hexadécimale représentant une couleur en objet couleur (Color).

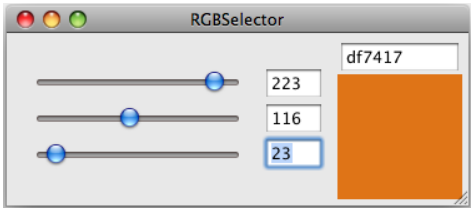

FIGURE 4 – Sélecteur de couleur.

Pour information, Swing propose en standard la classe JColorChooser[18] qui permet de choisir de manière simple une couleur.

17. http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Color.html#decode(java.lang.String)
18. http://java.sun.com/docs/books/tutorial/uiswing/components/colorchooser.html