Université d'Aix-Marseille III - Licence de math-info 2ème année

I4 : Programmation objet - L'interface graphique avec Swing

Swing est une bibliothèque de classes qui permet de créer une interface utilisateur graphique (GUI). Avec ce type d'application, nous nous trouvons dans le cadre de ce qu'on appelle la *programmation événementielle*. On crée et assemble différents composants (fenêtres, boutons, champs textuels, menus, etc) pour composer l'interface et on associe des méthodes aux différents événements pouvant survenir (appui sur un bouton, fermeture de fenêtre, entrée d'un texte dans un champ textuel, sélection d'un item dans un menu, etc).

Ce qui suit n'est qu'une présentation très partielle et succinte de ce qu'on peut faire avec Swing, avec pour seul objectif que vous puissiez résoudre les exercices qui vous seront donnés en TP. Pour avoir une vue générale et complète de Swing, on pourra par exemple consulter le tutorial à l'adresse :

`http://java.sun.com/docs/books/tutorial/uiswing/`

Voici un exemple de programme Java utilisant Swing :

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

class Fenetre implements ActionListener {
    private JLabel label = new JLabel("0");
    private int nbClics;

    public void actionPerformed(ActionEvent e) {
        nbClics++;
        label.setText("" + nbClics);
    }

    public void lanceFenetre() {
        JFrame frame = new JFrame("Compteur");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.getContentPane().setLayout(new FlowLayout());
        JButton bouton = new JButton("Incrementation");
        frame.getContentPane().add(bouton);
        frame.getContentPane().add(label);

        bouton.addActionListener(this);

        frame.pack();
        frame.setVisible(true);
    }
}

public class TesteFenetre {
    public static void main(String[] args) {
        Fenetre f = new Fenetre();
        f.lanceFenetre();
    }
}
```

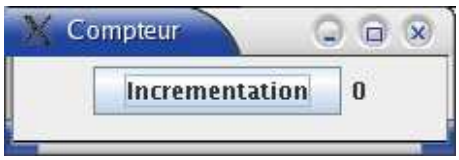

FIG. 1 – *Résultat de l'exécution du programme TesteFenêtre. A chaque fois qu'on cliquera sur le bouton "incrémentation", on augmentera le nombre situé à gauche d'une unité. Le programme s'arrêtera quand on fermera la fenêtre.*

Création et affichage d'une fenêtre

Il est conseillé d'utiliser le squelette de programme suivant :

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

class Fenetre ... {
    ...
    public void lanceFenetre() {
        JFrame frame = new JFrame("Titre");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ...
        frame.pack();
        frame.setVisible(true);
    }
}

public class TesteFenetre {
    public static void main(String[] args) {
        Fenetre f = new Fenetre();
        f.lanceFenetre();
    }
}
```

`JFrame frame = new JFrame("Titre");` permet de créer une fenêtre dont le titre sera "Titre", en créant l'instance `frame` de type `JFrame`. `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);` indique que la fenêtre se fermera si on clique sur le bouton de fermeture standard des fenêtres. Après qu'on ait ajouté tous les composants de la fenêtre, `frame.pack();` calcule sa structure interne. `frame.setVisible(true);` permet de rendre la fenêtre visible.

Ajout de composants dans une fenêtre

```
frame.getContentPane().setLayout(new FlowLayout());
JButton bouton = new JButton("Incrementation");
frame.getContentPane().add(bouton);
frame.getContentPane().add(label);
```

Les instances de `JFrame` (ici : `frame`) sont composés notamment d'un *panneau "contenu"*, de type `Container`, destiné à recevoir des composants, c'est-à-dire des instances de classes dérivées de `Component`. `Container` est elle-même une classe qui hérite de `Component`. L'accesseur `Container getContentPane()` retourne le panneau "contenu" de `frame`. `frame.getContentPane().setLayout(new FlowLayout());` choisit le *gestionnaire de mise en forme (layout)* de type `FlowLayout` pour le panneau. Les gestionnaires de mise en forme

déterminent la façon dont les composants seront disposés automatiquement sur le panneau. Ici, ce sera fait en alignant les composants horizontalement au fur et à mesure qu'ils sont ajoutés, en passant à la ligne suivante dès que le bord droit de la fenêtre est atteint. Lorsque la fenêtre sera redimensionnée, les composants se repositionneront automatiquement suivant le même principe mais en fonction des nouvelles dimensions de la fenêtre. Il existe d'autres types de "layout" : `BoxLayout`, `GridLayout`, `BorderLayout`, etc. La méthode `void add(Component)` permet de rajouter un composant au panneau en respectant le "layout". Ici, on a ajouté un bouton, puis un label, tous deux des instances de classes qui dérivent de `Component`.

Liaison événement-action

Une interface graphique sert à recevoir des entrées via des *événements*, à réagir à ces événements en exécutant certaines actions (des méthodes sont appelées) et à afficher graphiquement le résultat de ces actions. Par exemple, on associe une méthode à un événement de type "action" pouvant survenir à une instance de `JButton` grâce à la méthode `void addActionListener(ActionListener)` (définie dans la classe `JButton`). Le paramètre de cette méthode est une instance d'une classe implémentant l'interface `ActionListener` et définissant la méthode `public void actionPerformed(ActionEvent)`. C'est cette méthode qui sera appelée lorsque qu'on appuiera sur le bouton. Il est possible que plusieurs composants (par exemple, deux boutons) associent leur événement "action" à une même instance de `ActionListener`. Il faut alors pouvoir déterminer quel est le composant source de l'événement à l'intérieur de la méthode `actionPerformed`. C'est la méthode `Object getSource()` de la classe `ActionListener` retournant l'instance de l'objet sur lequel est apparu l'événement qui permet de le déterminer. `ActionListener` hérite de `EventListener`, qui est la racine de la hiérarchie des "event listeners". Ses autres sous-classes (interfaces) sont : `ComponentListener` pour réagir quand un composant change de taille, de position ou de visibilité, `FocusListener` pour réagir quand un composant obtient le focus ou le perd, `KeyListener` pour que le composant qui a le focus réagisse à l'appui ou au relachement d'une touche, `MouseListener` pour réagir quand la souris entre ou sort de l'aire d'affichage du composant ou qu'on clique dessus, qu'on appuie ou qu'on relache un bouton de souris, `MouseMotionListener` pour réagir au déplacement de la souris au dessus du composant, qu'un bouton de la souris soit pressé ou non, etc. Chacune de ces interfaces liste sa série de méthodes à définir en réaction aux événements. Le paramètre passé à l'exécution de ces méthodes est une instance d'une sous-classe de `EventObject` et permet de récupérer des informations complémentaires sur l'événement qui est survenu (ex : coordonnées du pointeur de la souris lorsqu'on clique sur la souris).

Exemple

L'application graphique suivante est constitué d'une fenêtre contenant un bouton Oui, un bouton Non et deux labels. Le premier label affiche le nom du bouton au-dessus duquel se trouve la souris. Le second label affiche le nom du dernier bouton sur lequel on a cliqué. La classe `Fenetre` implemente `ActionListener` et `MouseListener`. Les 5 méthodes de l'interface `MouseListener` sont donc définies mais 3 d'entre elles ne font rien. Voici l'état de l'application lorsque le dernier bouton sur lequel on a appuyé est le bouton Non et que le pointeur de la souris se trouve en dehors des zones occupés par les boutons :

FIG. 2 –

Voici le programme qui permet de réaliser cette application (cf au verso) :

```
package applicationgraphique;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

class Fenetre implements ActionListener, MouseListener {
   private JButton boutonOui = new JButton("Oui");
   private JButton boutonNon = new JButton("Non");
   private JLabel etat = new JLabel("????");
   private JLabel choix = new JLabel("???");

   // La méthode à définir quand
   // on implémente ActionListener :
   public void actionPerformed(ActionEvent e) {
      if(e.getSource() == boutonOui)
         choix.setText("Oui");
      if(e.getSource() == boutonNon)
         choix.setText("Non");
   }

   // Les 5 méthodes à définir quand
   // on implémente MouseListener :
   public void mouseEntered(MouseEvent e){
      if(e.getSource() == boutonOui)
         etat.setText("Oui");
      if(e.getSource() == boutonNon)
         etat.setText("Non");
   }

   public void mouseExited(MouseEvent e){
      etat.setText("???");
   }

   public void mouseClicked(MouseEvent e){}
   public void mousePressed(MouseEvent e){}
   public void mouseReleased(MouseEvent e){}

   public void lanceFenetre(){
      JFrame frame = new JFrame("Boutons");
      frame.setDefaultCloseOperation(
         JFrame.EXIT_ON_CLOSE);

      Container panneau = frame.getContentPane();
      panneau.setLayout(new FlowLayout());
      panneau.add(boutonOui);
      panneau.add(boutonNon);
      panneau.add(etat);
      panneau.add(choix);

      boutonOui.addActionListener(this);
      boutonOui.addMouseListener(this);
      boutonNon.addActionListener(this);
      boutonNon.addMouseListener(this);

      frame.pack();
      frame.setVisible(true);
   }
}

public class TesteFenetre {
   public static void main(String[] args) {
      Fenetre f = new Fenetre();
      f.lanceFenetre();
   }
}
```