Interfaces graphiques avec l'API Swing

Les Swing

Les classes graphiques Swing dérivent de la classe **`JComponent`** , qui hérite elle-même de la classe **AWT** (Abstract Window Toolkit).

Tous les composants Swing commencent par la lettre **`"J"`** . C'est la principale différence entre les composants AWT et les composants Swing.

La classe **`JComponent`** et les contrôles GUI (Graphical User Interface) se trouvent dans le package **`javax.swing`**

Les composants Swing se répartissent :

- en conteneurs de plus haut niveau (**`JFrame`**, **`JWindow`**, **`JApplet`** et **`JDialog`**)
- en conteneurs légers (les autres contrôles GUI Swing).

AWT et Swing

Les composants AWT sont des composants " lourds" c-à-d des contrôles produits par la machine virtuelle à destination du système d'exploitation.

Si on crée par exemple un bouton Button tiré du module java.awt sous Windows NT, la machine virtuelle génère un bouton NT et lui communique tous les paramètres nécessaires à son initialisation . L'aspect du bouton, comme des autres composants de ce type, dépend du système d'exploitation utilisé.

Les composants Swing sont des composants " légers " c-à-d directement dessinés par la machine virtuelle. Le composant aura toujours le même aspect quelque soit la plateforme utilisée. On trouve dans les Swing plus de fonctionnalités. Pour les Swing, un conteneur de plus haut niveau se compose d'une " fenêtre visible ", la **ContentPane**, placée au dessus de la fenêtre native . Les composants GUI doivent se placer dans cette ContentPane.

Création d'une fenêtre Swing

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

public class Swing01 extends JFrame{
  public Swing01 (String titre) {
       this.setTitle (titre);
       this.setSize (250,200);
       Container contenu = this.getContentPane( );
       contenu.setBackground (Color.yellow);
  }

  public static void main( String [] args) {
       Swing01 fen = new Swing01("Ma Fenêtre Swing");
       fen.setVisible (true);
  }
}
```

La même fenêtre avec AWT

```
import java.awt.*;

public class AWT01 extends Frame{
  public AWT01 (String titre) {
       this.setTitle (titre);
       this.setSize (250,200);
       this.setBackground (Color.yellow);
  }

  public static void main( String [] args) {
       AWT01 fen = new AWT01("Ma Fenêtre AWT");
       fen.setVisible (true);// pour rendre la fenêtre visible
  }
}
```

Remarques

Les classes **Color** et **Container** sont présentes dans le module **java.awt** , c'est pourquoi il faut toujours importer ce package. Dans la gestion des interfaces graphiques, il ne s'agit pas simplement de construire des composants, mais il faut aussi pouvoir interagir avec eux en produisant des évènements. Il s'agit de la programmation évènementielle qui nécessitent les classes de gestion d'évènements présentées dans les packages **java.awt.event** et **javax.swing.event**

En somme, il faut importer au minimum , les quatre packages suivante:

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

Ajout d'un composant léger: un JButton

```
import java.awt.*;
import javax.swing .*;
public class Swing02 extends JFrame{
     public Swing02 (String titre) {
           this.setTitle (titre); this.setSize (250,200);
           Container contenu = this.getContentPane( );
           contenu.setBackground (Color.yellow);
           JButton bouton = new JButton ("Copier");
           bouton.setBackground (Color.green);
           contenu.add (bouton);
     }
     public static void main( String [] args) {
          new Swing02("Ma Fenêtre Swing").setVisible (true);
     }
}
```

Ajout du JButton

La création d'un bouton nécessite l'usage d'un constructeur de la classe **JButton**.

Ici, on utilise le constructeur **JButton (String nomBoutton).**

```
JButton bouton = new JButton ("Copier");
```

On donne une couleur au bouton avec la méthode **setBackground (Color couleur)** appliqué à l'objet bouton.

```
bouton.setBackground (Color.green);
```

Et on ajoute le composant à la partie contenu de la fenêtre native (le ContenPane) en utilisant la méthode **add (Component comp):**

```
contenu.add (bouton);
```

Remarques sur l'ajout du bouton

A l'affichage de la fenêtre, il faut remarquer que seule la couleur verte (celle du bouton apparaît) et non celle de la fenêtre (couleur jaune).

En fait, le bouton occupe par défaut tout le ContenPane. Ceci s'explique par le fait que chaque composant de plus haut niveau dispose de ce que l'on nomme un gestionnaire de mise en forme (**Layout Manager**) qui permet de disposer les différents composants dans le ContenPane.

Pour **JFrame**, le gestionnaire est la classe **BorderLayout**. Avec ce gestionnaire, le composant occupe toute la fenêtre. Donc même si on rajoute un deuxième bouton à la fenêtre, il va se substituer au premier et on ne verra donc que le dernier composant ajouté. Pour visualiser les deux composants, il faut indiquer leur position car BorderLayout place les composants aux quatre points cardinaux (**North, West, East, South**) et au centre (**Center**).

Le gestionnaire de JFrame: BorderLayout

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

public class Swing03 extends JFrame{
      public Swing03 (String titre) {
              this.setTitle (titre); this.setSize (250,200);
              Container contenu = this.getContentPane( );
              contenu.setBackground (Color.yellow);


              JButton bouton = new JButton ("Copier");
              bouton.setBackground (Color.green);
              contenu.add (bouton, BorderLayout.SOUTH);
              JButton bout = new JButton ("Effacer");
              bout.setBackground (Color.green);
              contenu.add (bout, BorderLayout.NORTH) ;

}
```

Gestion de l'interface MouseListener

L'interface **`MouseListener`** permet de traiter les clics de la souris sur la fenêtre. On va se contenter d'afficher les coordonnées du point où l'on clique.

En Java, tout évènement possède ce qu'on appelle une **source**. Il s'agit de l'objet ayant donné naissance à cet évènement : bouton, menu, fenêtre…

Pour traiter un évènement, on associe à la source un objet de son choix dont la classe implémente une interface particulière à une catégorie d'évènement.

Cet objet est un écouteur de cette catégorie d'évènement. Chaque méthode proposée par l'interface correspond à une catégorie d'évènement.

Gestion de l'interface MouseListener

Il existe une catégorie d'évènement souris que l'on peut traiter avec un écouteur de souris, c'est-à-dire un objet d'une classe implémentant l'interface MouseListener.

Cette interface possède cinq méthodes: **`mouseClicked`**, **`mouseEntered`**, **`mouseReleased`**, **`mouseExited`** et **`mousePressed`**.

Pour prendre en compte la gestion du clic, seul l'évènement clic nous intéresse et ce dernier correspond à la méthode **`mouseClicked`**. Mais comme on implémente une interface, on est obligé de redéfinir toutes les méthodes de cette dernière.

Gestion de l'interface MouseListener

```
import java.awt .*; import java.awt.event.*; import javax.swing.*;
public class Swing04 extends JFrame implements MouseListener{
      public Swing04(String titre) {
              this.setTitle(titre);  this.setSize(250,200);
              Container contenu = this.getContentPane();
              contenu.setBackground(Color.yellow);

              this.addMouseListener ( this );
              /*la fenetre est son propre écouteur d'événement souris*/
      } /*redefinition obligatoire de toutes les méthodes de l'interface*/
      public void mouseClicked( MouseEvent e){
              System.out.println ("Vous avez cliqué sur le point de
                                  coordonnées : "+e.getX()+" "+e.getY());
      }
      public void mouseReleased( MouseEvent e) { }
      public void mouseExited( MouseEvent e) { }
      public void mousePressed( MouseEvent e) { }
      public void mouseEntered( MouseEvent e) { }
}
```

Les classes adapter

On constate que dans l'exemple précèdent, nous n'avions eu besoin que de la méthode **mouseClicked** mais on était obligé de redéfinir les autres méthodes de l'interface puisque Java l'impose lors de l'implémentation d'une interface.

Il existe une classe particulière appelée **MouseAdapter** qui implémente toutes les méthodes de l'interface **MouseListener** :

```
class MouseAdapter implements MouseListener{
     public void mouseReleased ( MouseEvent e) { }
     public void mouseExited ( MouseEvent e) { }
     public void mousePressed ( MouseEvent e) { }
     public void mouseEntered ( MouseEvent e) { }
     public void mouseClicked ( MouseEvent e) { }
}
```

Les classes adapter

Comme **`MouseAdapter`** est une classe et non une interface, on pourra désormais en dériver simplement, ce qui nous permettra d'utiliser que les méthodes que nous souhaitons exploiter (en les redéfinissant).

Presque toutes les interfaces Listener disposent d'une classe Adapter. Les interfaces Listener qui n'ont qu' un seul type d'évènement à traiter, donc une seule méthode ne disposent pas de classe adaptateur. Par exemple l'interface **`ActionListener`** qui gère la catégorie d'évènements action.

Voici comment on peut refaire le premier exemple en ne tenant compte que de la méthode **`mouseClicked`**.

Les classes adapter

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

public class Swing04 extends JFrame{
     public Swing04(String titre) {
               this.setTitle(titre);
               this.setSize(250,200);
               Container contenu = this.getContentPane();
               contenu.setBackground(Color.yellow);

               Ecouteur ecout = new Ecouteur();
               this.addMouseListener ( ecout );

     }
}
```

Les classes adapteurs

```
class Ecouteur extends MouseAdapter{

    /*on ne redéfinit que la méthode mouseClicked*/
    public void mouseClicked( MouseEvent e){
        System.out.println ("vous avez clique au point de
            coordonnes : "+e.getX()+" +e.getY());
    }
}
```

Si on utilise ici la classe MouseAdapter ai lieu de l'interface alors la fenêtre ne peut plus être son propre écouteur. Ceci impliquerait de dériver la classe Swing05 en même temps de JFrame et de MouseAdapter, ce qui est interdit.

Ecouteur avec une classe anonyme

```
class Swing06 extends JFrame {
    public Swing06(String titre) {
       this.setTitle (titre);
       this.setSize (250,200);
       Container contenu = this.getContentPane();
       contenu.setBackground (Color.yellow);

       //gestion de l'écouteur avec une classe anonyme
       this.addMouseListener (new MouseAdapter (){
            public void mouseClicked( MouseEvent e){
              System.out.println ("vous avez clique au
                                   point de coordonnes :
                                  "+e.getX()+" "+e.getY() );
           }
       } ) ;
    }
}
```

Mettre fin à l'application

Le simple clic sur le bouton de fermeture de la fenêtre ne permet de mettre fin à l'application. Il rend simplement la fenêtre invisible.

Le clic de fermeture est équivalent à faire:

```
new Swing02("Ma Fenêtre Swing").setVisible (false);
```

Autrement dit le processus qui gère l'application tourne toujours en tâche de fond. Pour l'arrêter, il faut interrompre le compilateur, ce qui n'est pas optimal. Il faut toujours gérer la fin de l'application par des instructions .

Pour ce faire, on va voir un premier cas d'utilisation de la gestion des évènements avec la classe **`java.awt.event.WindowListener`** dans l'implémentation d'une classe anonyme.

Mettre fin à l'application

```
class Swing07 extends JFrame {
      public Swing07 (String titre) {
              this.setTitle (titre); this.setSize (250,200);
              Container contenu = this.getContentPane( );
              contenu.setBackground (Color.yellow);

              /* pour mettre fin a l'application dès qu'on clique sur le
              bouton de fermeture*/

              this.addWindowListener (new WindowAdapter ( ){
                      public void windowClosing (WindowEvent e){
                              System.exit (0);
                      }
              } );

       }

 }
```

Action sur un bouton

Un bouton génère une catégorie d'évènement appelée **action** que l'on traite avec un écouteur qui est un objet implémentant l'interface **ActionListener**.

Cette dernière ne possède qu'une seule méthode :

```
public void actionPerformed (ActionEvent ev).
```

Comme illustration, on considérera un bouton et deux zones de texte, l'une contenant un texte et l'autre étant vide; le clic sur le bouton entraînera la copie du contenu de la première zone de texte dans la seconde, et le vidange de celle-là.

On supposera que la fenêtre est l'objet écouteur des clics sur le bouton.

Action sur un bouton

```
import java.awt.event.*;import java.awt.*;import javax.swing.*;
public class Swing07 extends JFrame implements ActionListener{
      JTextField texteinitial, textefinal;
      JButton bouton;
      public Swing07 (String titre) {
             this.setTitle(titre); this.setSize(250,100);
             Container contenu = this.getContentPane();
             contenu.setBackground (Color.yellow);
             bouton = new JButton("Copier");
             bouton.setBackground(Color.green);
             contenu.add(bouton,BorderLayout.SOUTH);
             texteinitial = new JTextField("texte initial",15);
             contenu.add( texteinitial, BorderLayout.NORTH );
             textefinal = new JTextField("",15);
             contenu.add( textefinal, BorderLayout.CENTER);
             bouton.addActionListener (this);
}
```

Action sur un bouton

```
   /*redéfinition de la méthode actionPerformed*/
   public void actionPerformed(ActionEvent e){
        if ( e.getSource() == bouton){
               textefinal.setText( texteinitial.getText ( ) );
               texteinitial.setText("");
        }
   }

   public static void main(String[] args) {
        Swing07 fen = new Swing07("Ma Fenêtre Swing");
        fen.setVisible(true);
   }
}
```

Pour déterminer la source du clic, on utilise la méthode **getSource ( )** qui fournit une référence de type Object sur l'objet ayant déclenché l'évènement.

Les gestionnaire de mise en forme

Le rôle d'un gestionnaire de mise en forme est de permettre une disposition des composants selon le choix de l'utilisateur. Parmi les gestionnaires de mise en forme on peut citer:

FlowLayout : représente les composants sur une même ligne, les uns à la suite des autres; s'il n'y a plus d'espace en fin de ligne, il passe à la ligne suivante.

CardLayout : permet de disposer des composants suivant une pile, à la manière d'un paquet de cartes, un seul composant étant visible à la fois,

BoxLayout : dispose les composants sur une seule ligne ou sur une seule colonne,

GridBagLayout : dispose les composants sur une grille, la taille d'un composant dépend du nombre de cellules que le composant occupe.

GridLayout : dispose les composants sur une grille, les composants de même colonne ayant la même taille.

Exemple d'utilisation de FlowLayout

```
public class Flow extends JFrame {
      JTextField texteinitial;
      JButton bouton;
      JTextField textefinal;
      public Flow(String titre) {
              this.setTitle(titre);
              this.setSize(250,150);
              Container contenu = this.getContentPane();

              contenu.setLayout (new FlowLayout ( ));

              bouton = new JButton("Copier");
              bouton.setBackground(Color.green);
              contenu.add(bouton);
              texteinitial = new JTextField("texte initial",15);
              contenu.add(texteinitial);
              textefinal = new JTextField("",15);
              contenu.add(textefinal);
}
```

Exemple d'utilisation de GridLayout

```
public class Grid extends JFrame{
      JTextField texteprenom, textenom;
      JLabel prenom ,nom;
      public Grid(String titre) {
              this.setTitle(titre);
              this.setSize(250,150);
              Container contenu = this.getContentPane();

              contenu.setLayout(new GridLayout(2,2));

              prenom = new JLabel("prénom");
              nom = new JLabel("nom");
              texteprenom = new JTextField("texte initial",15);
              textenom = new JTextField("",15);


              contenu.add(prenom); contenu.add(texteprenom );
              contenu.add(nom); contenu.add(textenom );
}
```

Aucun gestionnaire de mise en forme

Il se peut , lors de la construction d'une interface graphique, que le programmeur ne veuille utiliser aucun de gestionnaires prédéfinies. Cela voudra dire qu'il prend ses propres dispositions pour ajouter les composants lui-même à l'endroit où il voudrait bien les placer.

Dans ce cas, il faut signaler qu'on n'utilise aucun gestionnaire en faisant:

```
objet_conteneur.setLayout(null) ;
```

et après d'utiliser la méthode `setBounds(int a, int b, int c, int d )`

Où:

a = abscisse du point d'insertion du composant,

b = ordonnée du point d'insertion du composant,

c = largeur du composant,

d = hauteur du composant.

Les JPanel

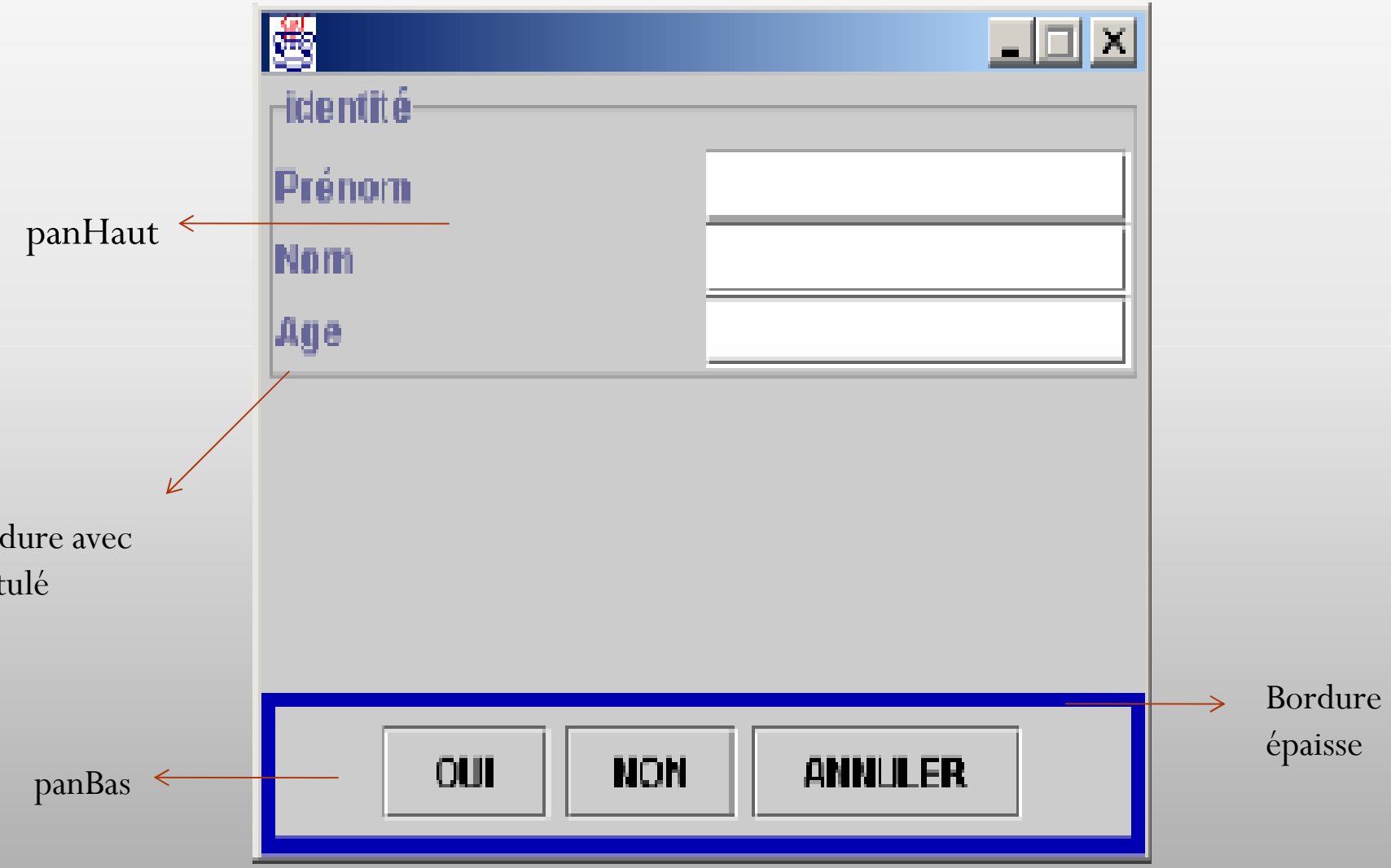
Si ont veut ranger en même temps et directement dans un **JFrame** des composants suivant une grille avec par exemple **GridLayout** et d'autres composants selon une ligne horizontale avec **FlowLayout**, cela va s'avérer impossible puisqu'on ne peut pas appliquer deux gestionnaires simultanément.

L'astuce qu'il faut utiliser est de créer deux panneaux, l'un pour le premier groupe de composants, le second pour le deuxième groupe.

Les panneaux sont des containers puisqu'ils servent à contenir des composants légers. Un panneau est une sorte de sous fenêtre sans titre, ni bordure.

Le gestionnaire par défaut de **JPanel** est **FlowLayout**.

Exemple de JPanel



Exemple de JPanel

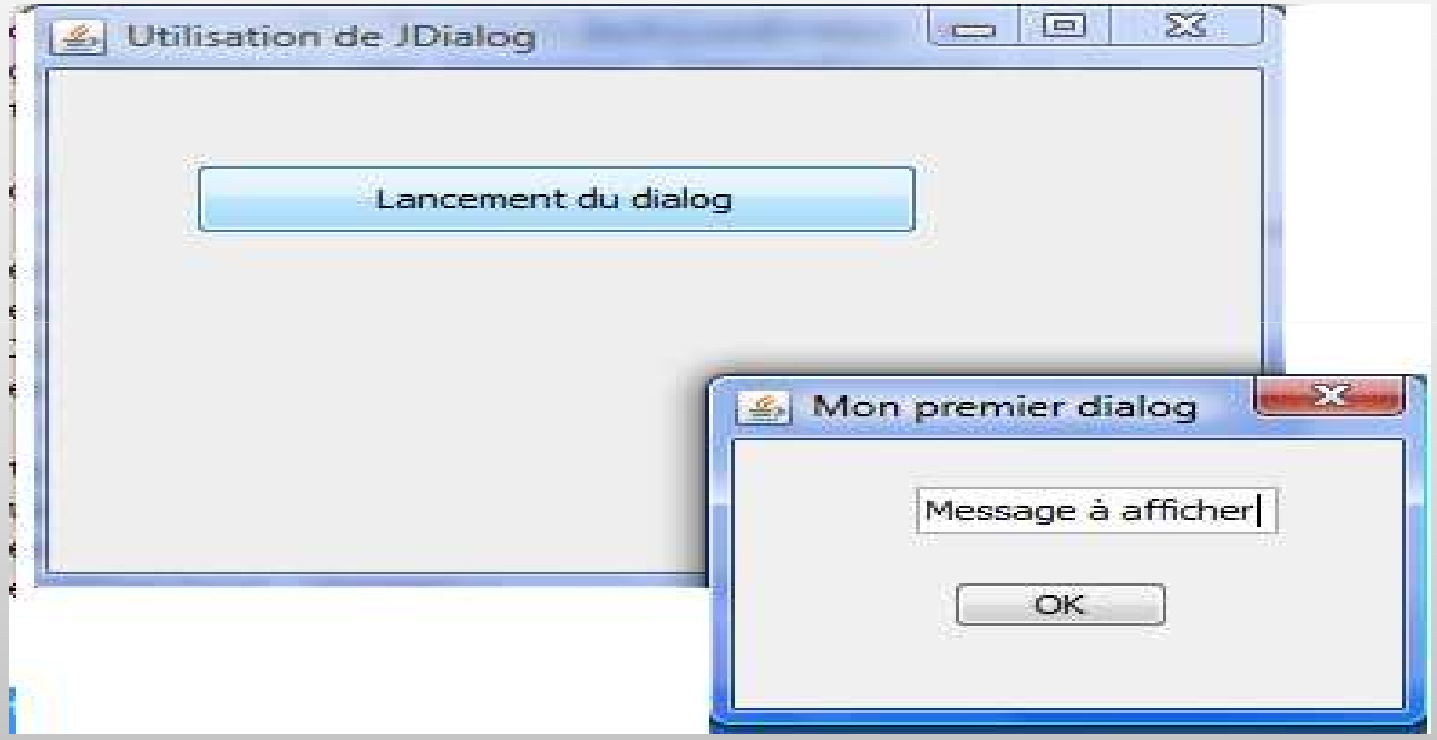
```
import java.awt .*; import javax.swing .*; import javax.swing.border .*;
public class JPanneau extends JFrame {
      JPanel panHaut, panBas;
      public JPanneau() {
            /*initialisation du JFrame*/
            super();
            this.setTitle("Panneau");
            this.setSize (new Dimension (300,250));
            this.setResizable(false); //on ne pourra pas agrandir la
                                      //fenetre
            /*recupération du ContentPane*/
            Container contenu = this.getContentPane ();
            /*creation des JPanel avec leur Layout Manager*/
            panHaut = new JPanel(new GridLayout (3,2));
            panBas = new JPanel ( );
            /*ajout des panneaux au ContentPane,l'un au nord, l'autre
            au sud*/
            contenu.add (panHaut, BorderLayout.NORTH);
            contenu.add(panBas, BorderLayout.SOUTH);
```

Exemple de JPanel

```
        /*ajout de trois label et de trois zones de texte à panHaut*/
        panHaut.add( new JLabel ("Prénom")); panHaut.add (new JTextField());
        panHaut.add( new JLabel("Nom")); panHaut.add(new JTextField());
        panHaut.add (new JLabel("Age")); panHaut.add(new JTextField());

        /*ajout de trois boutons à panBas*/
        panBas.add ( new JButton("OUI"));
        panBas.add ( new JButton("NON"));
        panBas.add ( new JButton("ANNULER"));

        /*ajout d une bordure avec intiutlé à panHaut*/
        panHaut.setBorder ( new TitledBorder("Identité"));

        /*ajout d une bordure epaisse à panBas*/
        Border b = BorderFactory.createLineBorder (Color.blue.darker() ,5) ;
        panBas.setBorder(b);
    }
```

Les boites de dialogues

La classe **JDiaolog**



La classe JDialog

```
public class Dialog extends JFrame implements ActionListener{
	JDialog dialog;
	JButton lancer, ok;
	public Dialog (String title) {
		try {
			UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.
				WindowsLookAndFeel");
		}
		catch (Exception e) {
			e.printStackTrace();
		}
		this.setTitle( title);
		this.setSize(350,250);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Container c = this.getContentPane();
		c.setLayout (null);
		lancer = new JButton ("Lancement du dialog");
		lancer.addActionListener (this);
		lancer.setBounds (40,40,200,30);
		c.add (lancer);
	}
```

La classe JDialog

```
public void actionPerformed(ActionEvent e){
      if (e.getSource() == lancer) lanceDialog ();
      if (e.getSource() == ok)     dialog.dispose ( );
}

public void lanceDialog() {
      dialog = new JDialog(this,"Mon premier dialog",true);
      dialog.setBounds (170,170,200,150);
      dialog.getContentPane().setLayout (null);
      JTextField text = new JTextField("Message à afficher") ;
      dialog.getContentPane().add(text).setBounds (50,20,100,20);
      ok = new JButton("OK");
      ok.addActionListener (this);
      ok.setBounds (60,60,60,20);
      dialog.getContentPane().add (ok);
      dialog.setVisible (true);
}
```

Commentaires

Dans l'instruction :

`dialog = new JDialog( this, "Mon premier dialog", true);` on a trois arguments:

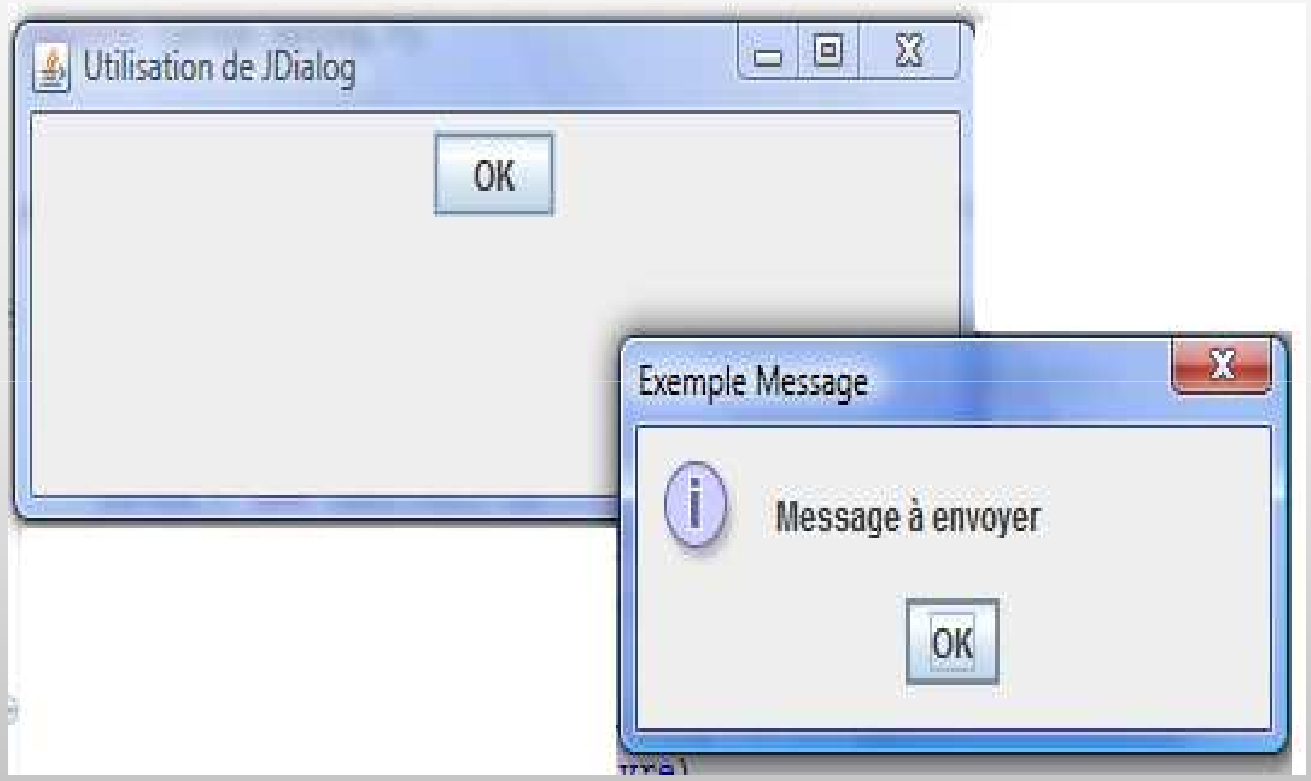
`this` désigne la fenêtre propriétaire (parent) c-à-d celle contenant le `Jdialog ;`

`"Mon premier dialog"` désigne le titre de la boîte de dialogue; `true` la boîte de dialogue est modale c-à-d une fois lancée, l'utilisateur ne peut pas agir sur d'autres que ceux intégrés dans la boîte de dialogue.

Remarque : il est possible (de la même façon qu'on utilise la classe `JFrame`) de créer une classe qui dérive de `JDialog` et d'y ajouter toutes les fonctionnalités dont on souhaite disposer.

Il est aussi possible de créer des boîtes de dialogue sans faire usage de la classe `JDialog`.

La classe JOptionPane



La classe JOptionPane

```
public class Message extends JFrame implements ActionListener{
      JButton ouvre;
      public Message (String titre) {
            super(); this.setTitle(titre);
            this.setSize(400,150);
            this.getContentPane().setLayout( new FlowLayout());
            ouvre = new JButton("OK");
            ouvre.addActionListener(this);
            this.getContentPane().add(ouvre);
      }
      public void actionPerformed(ActionEvent e){
            if (e.getSource() == ouvre)
                  JOptionPane.showMessageDialog(this, "Message à
                  envoyer", "ExempleMessage",
                  JOptionPane.INFORMATION_MESSAGE, null);
      }
}
```