

# Cours de Programmation Orientée Objet JAVA

**Demba SOW**

*Docteur en  
Mathématiques et  
Cryptologie*

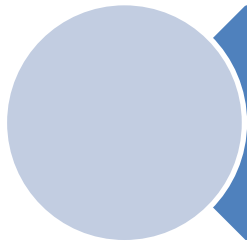
**L.A.C.G.A.A.**

**F.S.T. / U.C.A.D.**





**Swing GUI** **Les**



# Les Swing GUI

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

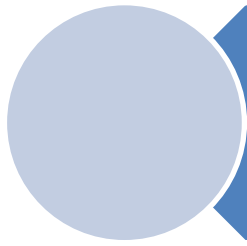
**Quelle est la différence entre les composants AWT et les composants Swing ?**

Tous les composants Swing commencent par la lettre " **J** " .

La classe JComponent et les contrôles GUI (Graphical User Interface) se trouvent dans le paquetage **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  d des contr  les produits par la machine virtuelle    destination du syst  me d'exploitation. Si vous cr  ez par exemple un bouton **Button** tir   du module **java.awt** sous Windows NT, la machine virtuelle g  n  re un bouton NT et lui communique tous les param  tres n  cessaires    son initialisation . L'aspect du bouton, comme des autres composants de ce type, d  pend du syst  me d'exploitation utilis  .

Les composants Swing sont des composants " **l  gers** " c  d **directement dessin  s** par la machine virtuelle. Le composant aura toujours le m  me aspect quelque soit la plateforme utilis  e. On trouve dans les Swing plus de fonctionnalit  s.

Pour les Swing, un conteneur de plus haut niveau se compose d'une " fen  tre visible " , la **ContentPane**, plac  e au dessus de la fen  tre native . Les composants GUI doivent se placer dans cette ContentPane.

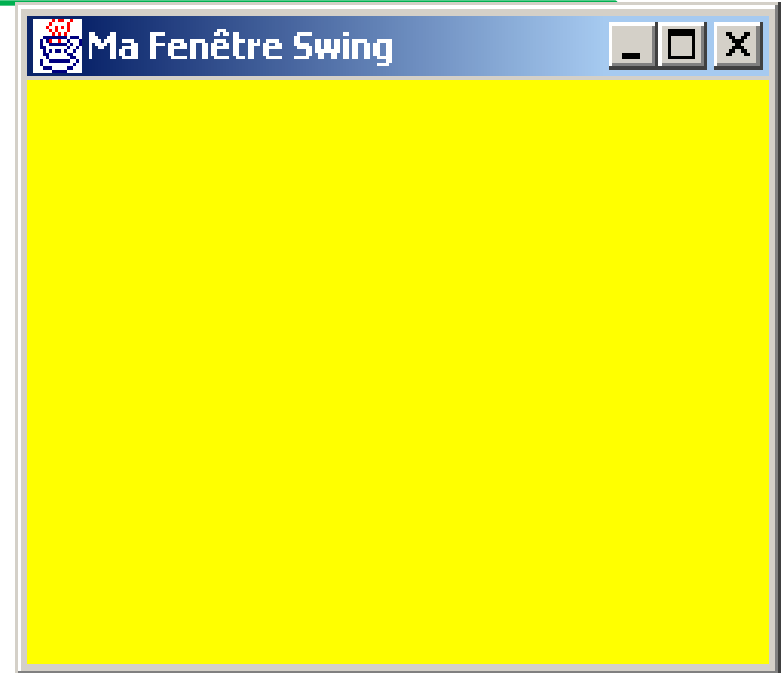
# Création d'une fenêtre Swing

```
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);
    }
}
```



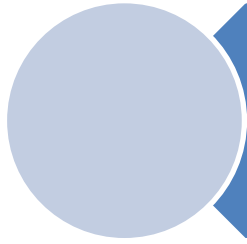
# La même fenêtre en AWT

```
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 modules **java.awt.event.\*** et **javax.swing.event.\***

En somme, il faut importer au minimum , les quatre bibliothèques suivantes:

**java.awt.\***

**java.awt.event.\***

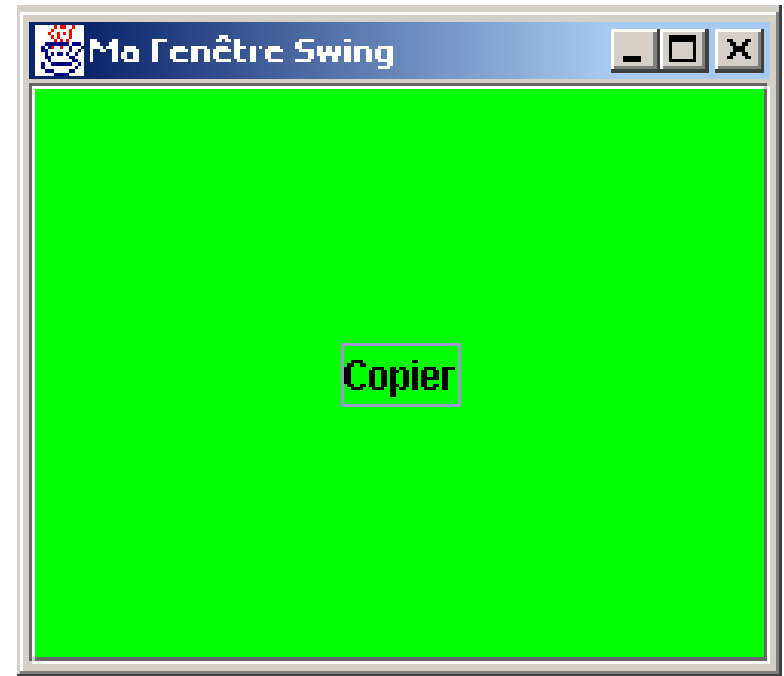
**javax.swing.\***

**javax.swing.event.\***

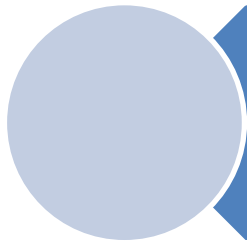
# Ajout d'un composant léger: un JButton

```
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 intitule)** .

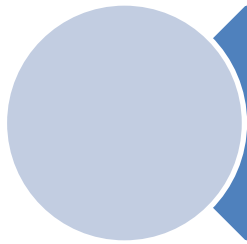
```
JButton bouton = new JButton ("Copier");
```

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 ContentPane) en utilisant la méthode **add (Component comp)** :

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



## Remarques sur l'ajout du JButton

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 ContentPane. Ceci s'explique du 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 ContentPane.

Pour **JFrame**, le gestionnaire est la classe **BorderLayout**.

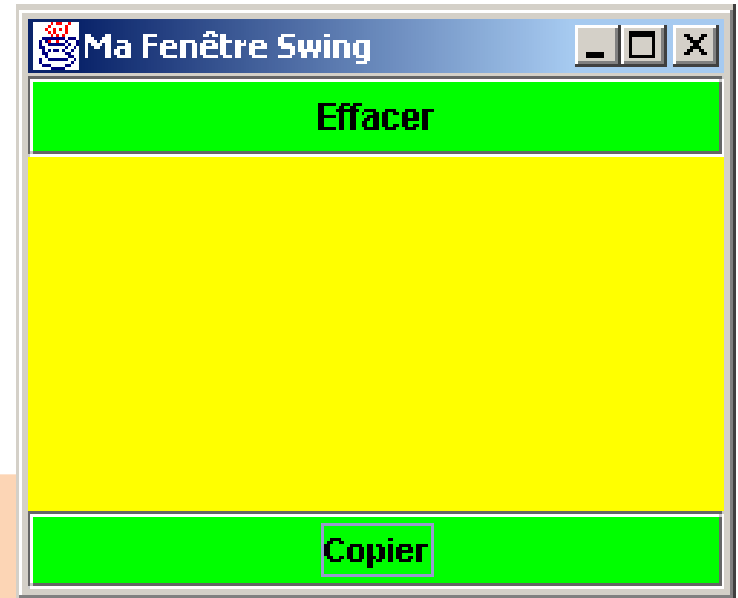
Avec ce gestionnaire, le composant occupe toute la fenêtre. Donc même si vous rajouter un deuxième bouton à la fenêtre, **il va se substituer au premier** et vous ne verrez donc que le dernier composant ajouté. Pour visualiser les deux composants, il faut indiquer leur position car BorderLayout place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

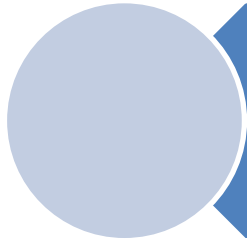
# Le gestionnaire de JFrame: BorderLayout

```
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 bouton1 = new JButton ("Copier");  
        bouton1.setBackground (Color.green);  
        contenu.add (bouton1, BorderLayout.SOUTH);  
        JButton bouton2 = new JButton (" Effacer");  
        bouton2.setBackground (Color.green);  
        contenu.add (bouton2, BorderLayout.NORTH);
```

```
    }  
}
```



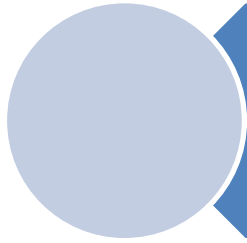


## Gestion de l'interface MouseListener

Nous allons implémenter l'interface **MouseListener**. Donc voir comment traiter un clic de 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

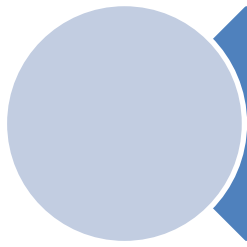
Il existe une catégorie ***d'évènement souris*** qu'on peut traiter avec un ***écouteur de souris***, c'est-à-dire un objet d'une classe implémentant l'interface **MouseListener**.

Cette interface possède cinq méthodes:

**mouseClicked**, **mouseEntered**, **mouseReleased**, **mouseExited** et **mousePressed**.

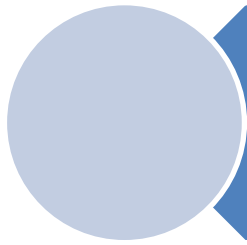
Pour prendre en compte la gestion du clic, seul l'évènement clic nous intéresse et ce dernier correspond à la méthode **mouseClicked**. Mais comme on implémente une interface, on est obligé de redéfinir toutes les méthodes de cette dernière.

Voici comment il faut procéder:



# 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);
        /*la fenêtre est son propre écouteur d'événement souris*/
        this.addMouseListener ( this );
    }
    /*redéfinition obligatoire de toutes les méthodes de l'interface*/
    public void mouseClicked( MouseEvent e)
    {System.out.println ("vous avez cliqué au 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) { }
}
```



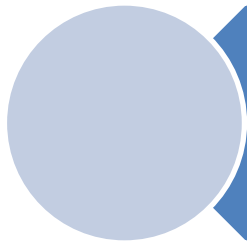
# Commentaires

Dans l'exemple précédent, nous avons choisi la fenêtre comme son propre écouteur d'évènement souris. C'est pourquoi, il est obligatoire de mentionner **implements MouseListener** dans l'entête de la classe. Si vous l'omettez, il y a erreur de compilation.

La mention **this.addMouseListener ( this )** associe un écouteur à la fenêtre principale. Si vous l'omettez, il n'y a pas erreur, seulement le clic sera sans effet.

Supposons maintenant, au lieu de considérer que la fenêtre soit son propre écouteur d'évènement souris, que son écouteur soit un objet quelconque. Tout ce qu'il faut vraiment savoir ici est que la classe de cet objet doit implémenter l'interface **MouseListener**.

Voyons comment traiter l'exemple précédent :

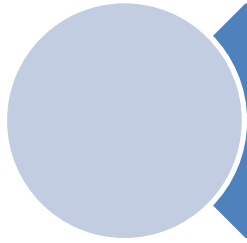


# Personnalisation de l'objet écouteur (1/2)

```
import java.awt.*; import java.awt.event.*; import javax.swing.*;
public class Swing05 extends JFrame {
public Swing05 (String titre) {
this.setTitle(titre);
this.setSize(250,200);
Container contenu = this.getContentPane( );
contenu.setBackground(Color.yellow);
/* on crée un objet écouteur de la fenêtre*/
    EcouteurFenetre ecout = new EcouteurFenetre ( );

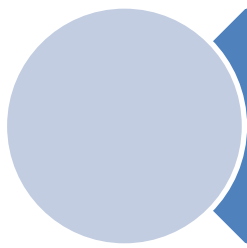
/*l'objet ecout devient maintenant l'écouteur d'évènement souris de la fenêtre*/
    this.addMouseListener ( ecout );
}
} // fin de la classe Swing05
```





## Personnalisation de l'objet écouteur (2/2)

```
class EcouteurFenetre implements MouseListener {  
    /*redéfinition obligatoire de toutes les méthodes de l'interface*/  
  
    public void mouseClicked( MouseEvent e)  
        {System.out.println ("vous avez clique au point de coordonnes : "+e.getX()+"  
            "+e.getY());  
        }  
  
        public void mouseReleased ( MouseEvent e) { }  
        public void mouseExited ( MouseEvent e) { }  
        public void mousePressed ( MouseEvent e) { }  
        public void mouseEntered ( MouseEvent e) { }  
}
```



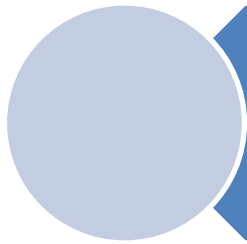
# Les classes Adaptateur (1/4)

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

**Comment faire donc pour n'utiliser que la méthode qui nous intéresse ici ?**

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

```
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) { }
}
```

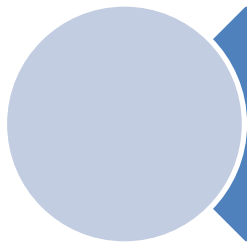


## Les classes Adaptateur (2/4)

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 Adaptateur (3/4)

```
import java.awt.*; import java.awt.event.*; import javax.swing.*;
public class Swing05 extends JFrame {
public Swing05 (String titre) {
this.setTitle(titre);
this.setSize(250,200);
Container contenu = this.getContentPane( );
contenu.setBackground(Color.yellow);
/* on crée un objet écouteur de la fenêtre*/
    EcouteurFenetre ecout = new EcouteurFenetre ( );
/*l'objet ecout devient maintenant l'écouteur d' événement souris de la fenêtre*/
    this.addMouseListener ( ecout );
}
} // fin de la classe Swing05
```

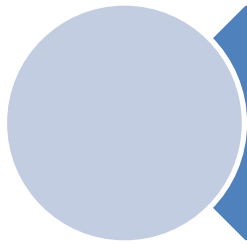
# Les classes Adaptateur (4/4)

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



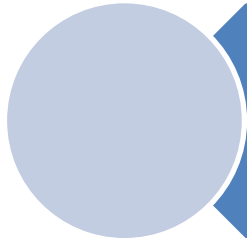
ATTENTION

Si vous utilisez ici la classe MouseAdapter au lieu de l'interface il ne sera plus possible de considérer *que la fenêtre est son propre écouteur*. Ceci impliquerait de dériver la classe Swing05 en même temps de JFrame et de MouseAdapter, ce qui est interdit.



## Gestion de l'écouteur Avec une classe Anonyme

```
public class Swing04 extends JFrame {  
    public Swing04(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 cliqué au point de coordonnées :  
                "+e.getX()+" "+ e.getY());  
            }  
        }  
    }  
}
```



## Mettre fin à l'application (1/2)

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

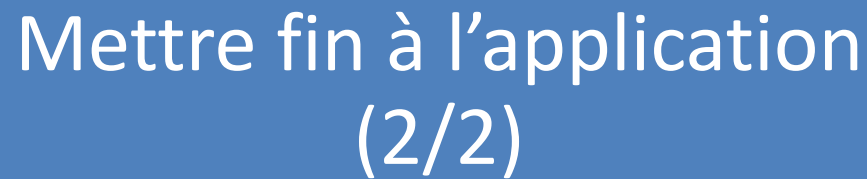
Le clic de fermeture est équivalent à faire:

```
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.

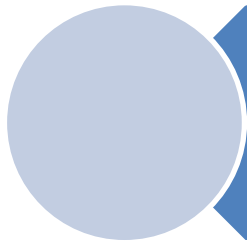


## Mettre fin à l'application (2/2)

```
import java.awt.*;
import javax.swing.*;
public class Swing03 {
    public Swing03 (String titre) {
        this.setTitle (titre); this.setSize (250,200);
        Container contenu = this.getContentPane( );
        contenu.setBackground (Color.yellow);

        /* pour mettre fin a l'application des 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 gère 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).**

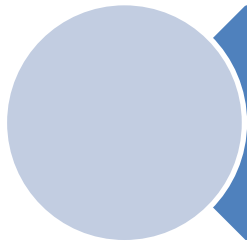
Comme illustration, nous allons considérer un bouton et deux zones de texte, l'une contenant un texte et l'autre étant vide;

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.

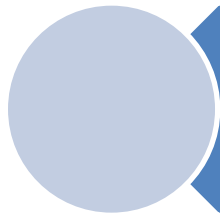
# Action sur un bouton

```
import java.awt.event.*;import java.awt.*;import javax.swing.*;
public class Swing06 extends JFrame implements ActionListener {
    JTextField texteinitial, textefinal;
    JButton bouton;
    public Swing06 (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);
        bouton.addActionListener (this);
        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);
    }
```



# 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) {  
    Swing06 fen = new Swing06("Ma Fenêtre Swing");  
    fen.setVisible(true);  
}  
}
```

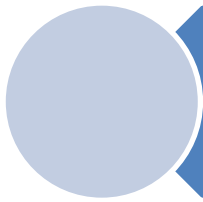


# Action sur un bouton

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.

Au lieu d'utiliser la référence du bouton pour tester dans l'instruction **if** la source de l'évènement, on peut utiliser l'étiquette du bouton, appelée une **chaîne de command**. Dans ce cas, on n'utilise pas la méthode `getSource( )` mais la méthode `getActionCommand`, de la façon suivante:

```
public void actionPerformed(ActionEvent e)
{ /*on récupère l'étiquette du bouton sur lequel on clique dans une chaine*/
    String etiquette = e.getActionCommand ( );
    // on utilise equals pour comparer le contenu de deux chaines
    if ( etiquette.equals (" Copier" ) )
    { textefinal.setText( texteinitial.getText ( ) );
      texteinitial.setText(" ");
    }
}
```



## Les gestionnaires 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. Nous avons déjà vu le gestionnaire **BorderLayout** pour la fenêtre principale. Nous allons à présent explorer les autres types de gestionnaires.

**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.

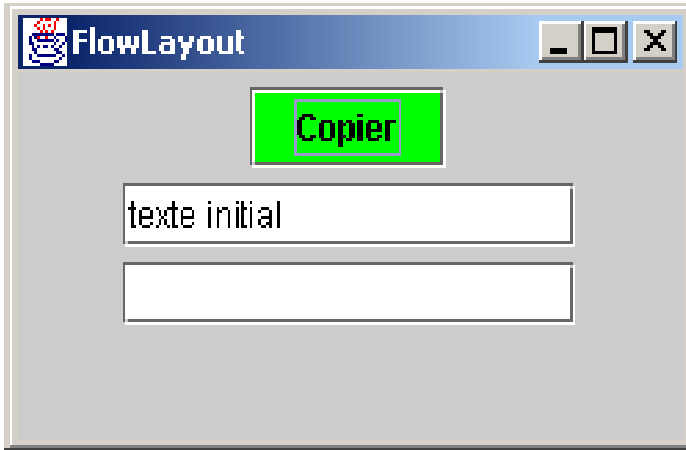
**CardLayout** : permet de disposer des composants suivant une pile, à la manière d'un paquet de cartes, un seul composant étant visible à la fois,

**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.

# Exemples de mise en œuvre de FlowLayout

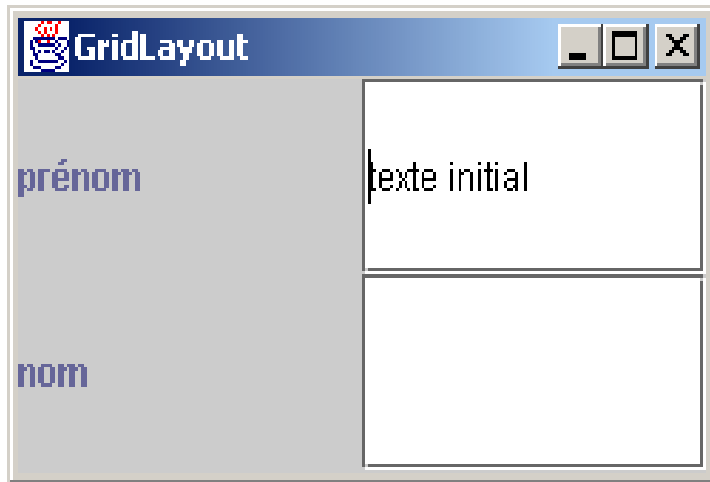


On associe un gestionnaire à un conteneur de haut niveau avec la méthode **setLayout (LayoutManager)**

```
public class Swing07 extends JFrame {
    JTextField texteinitial;
    JButton bouton;
    JTextField textefinal;
    public Swing07(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);
    }
}
```

# Exemples de mise en œuvre de GridLayout



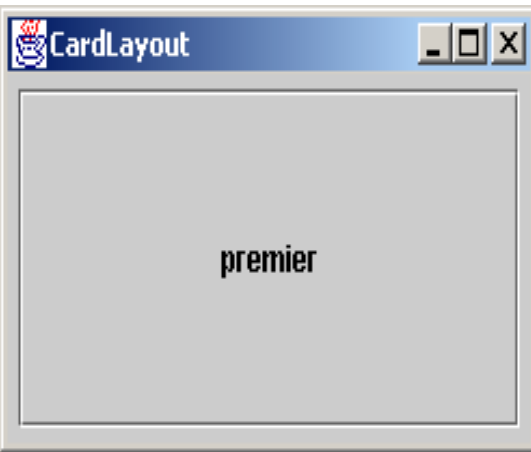
Les zones de texte sont trop spacieuses; on verra qu'on peut avoir des zones plus rétrécies avec l'utilisation d'objets panneaux (**JPanel**)

```
public class Swing08 extends JFrame{
    JTextField texteprenom;
    JLabel prenom ,nom;
    JTextField textenom;
    public Swing08(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 );
```

# Exemples de mise en œuvre de CardLayout



Si on clique sur **premier** il affiche **deux**, si on clique sur **deux**, il affiche **quat** ...

```
public class Swing09 extends JFrame implements ActionListener{
    CardLayout pile; JButton prem,deux,trois,quat;
    public Swing09(String titre) {
        this.setTitle (titre); this.setSize (250,150);
        pile = new CardLayout (5,3); // hgap = 5 vgap = 3
        this.getContentPane ( ).setLayout (pile) ;
        prem = new JButton("premier"); prem.addActionListener (this);
        deux = new JButton("deuxieme"); deux.addActionListener (this);
        trois = new JButton("troisieme"); trois.addActionListener (this);
        quat = new JButton("quatrieme"); quat.addActionListener (this);
        this.getContentPane( ).add (prem,"Bouton"); //obligatoire
        this.getContentPane( ).add (deux,"Bouton");
        this.getContentPane( ).add (trois,"Bouton");
        this.getContentPane( ).add (quat,"Bouton");
    }
    public void actionPerformed (ActionEvent e)
    { if (e.getSource ( ) == prem) pile.next (this.getContentPane());
      if (e.getSource ( ) == deux) pile. last (this.getContentPane());
      if (e.getSource ( ) == trois) pile.first (this.getContentPane());
      if (e.getSource ( ) == quat) pile.previous (this.getContentPane ());
    }
}
```



## Tour d'horizon de GridBagLayout

Des différents Layout Manager, le gestionnaire **GridBagLayout** est le plus difficile à manier.

Il permet de disposer les composants selon une grille, ceux-ci pouvant occuper plusieurs cases.

Cette classe ne dispose que d'un seul constructeur sans paramètre:

**GridBagLayout** ( ). Pour associer ce gestionnaire à un **Container** conteneur : **conteneur.setLayout (new GridBagLayout ( ) );**

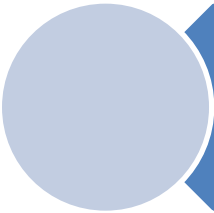
Mais cette instruction ne suffit pas pour placer les composants au conteneur.

Tout composant à ajouter doit disposer d'un objet **GridBagConstraints** lequel spécifie comment faire le placement des différents composants:

*/\*The GridBagConstraints class provides the means to control the layout of components within a Container whose LayoutManager is GridBagLayout.\*/*

**GridBagConstraints** **objetPlaceur** = **new GridBagConstraints ( ) ;**

Cet objet **objetPlaceur** dispose alors de variables et de méthodes permettant de réaliser le placement des composants.



## GridBagConstraints: variables et méthodes (1/2)

**/\*Specifies the alignment of the component in the event that it is smaller than the space allotted\*/**

**public int anchor**

**/\*The component's resize policy if additional space available. \*/**

**public int fill**

**/\*Number of columns (gridheight), of rows (gridwidth) a component occupies.\*/**

**public int gridheight,** **public int gridwidth**

**/\*Horizontal (gridx), vertical (gridy) grid position at which to add component. \*/**

**public int gridx,** **public int gridy**

**/\*Specifies the outer padding around the component.\*/**

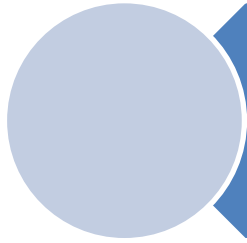
**public Insets insets**

**/\*Serves as the internal padding within the component in both the right and left direction\*/**

**public int ipadx**

**/\*Serves as the internal padding within the component in both the top and bottom directions\*/**

**public int ipady**



## GridBagConstraints: variables et méthodes (2/2)

**/\*Represents the percentage of extra horizontal space that will be given to this component if there is additional space available within the container\*/.**

**public double weightx**

**/\*Represents the percentage of extra vertical space that will be given to this component if there is additional space available within the container. \*/**

**public double weighty**

# Exemple de mise en œuvre de GridBagLayout

**1**

Création d'un nouveau client

*Identité client*

Prénom :

Nom :

Adresse :

Code Postal :

☐ NouveauBanquier

Valider

Retour

**2**

Création d'un nouveau client

*Identité client*

Prénom :

Nom :

Adresse :

Code Postal :

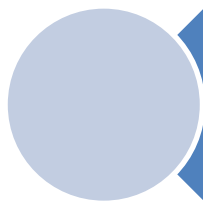
mot de passe :

☒ NouveauBanquier

Valider

Retour

Le clic sur la case **NouveauBanquier** dans **1** donne la même fenêtre **2**



# Code Example (1/7)

```
public class ExGridBagConstraints extends JFrame implements ActionListener{
    JLabel motpass;
    JCheckBox nbq;
    JPasswordField txtpass;
    Container c;
    GridBagConstraints gr;
    Container c;
    public ExGridBagConstraints( ) {
        this.setResizable ( false ) ;
        this.setTitle ("Création d'un nouveau client") ;
        Dimension screensize = Toolkit.getDefaultToolkit() .getScreenSize() ;
        Dimension framesize = this.getSize() ;
        if (framesize.width > screensize. width ) framesize. width =screensize.width ;
        if (framesize.height > screensize.height ) framesize.height =screensize.height ;
        this.setLocation ((screensize.width - framesize. width )/3,
            (screensize.height - framesize. height )/3) ;
    }
}
```

## Code Example (2/7)

```
c = this.getContentPane();  
JLabel acc = new JLabel("Identité client");  
acc.setFont (new Font("Helvetica", Font.BOLD + Font.ITALIC ,18));  
JLabel pr = new JLabel("Prénom :");      JTextField txtpr = new JTextField();  
JLabel nm = new JLabel("Nom :");          JTextField txtnm = new JTextField();  
JLabel cp = new JLabel("Code Postal :");  JTextField txtcp = new JTextField();  
JLabel adr = new JLabel("Adresse :");     JTextField txtadr =new JTextField();  
motpass = new JLabel("mot de passe :");  
txtpass = new JPasswordField();  
nbq = new JCheckBox("NouveauBanquier"); nbq.addActionListener(this) ;  
JButton valider = new JButton("Valider");  
JButton retour = new JButton("Retour");
```

# Code Example (3/7)

**/\*ajouter centrer le label Bienvenue\*/**

```
gr = new GridBagConstraints( );
```

```
gr.gridx =1; gr.gridy =4;  
gr.insets = new Insets(10,30,10,50);  
c.add (acc,gr) ;
```

**/\*ajouter le label prénom\*/**

```
gr = new GridBagConstraints();  
gr.gridx =1; gr.gridy =8;  
gr.anchor =GridBagConstraints.WEST ;  
gr.ipadx =0;  
gr.insets = new Insets(40,20,2,0);  
c.add (pr,gr) ;
```

**/\*ajouter le label nom\*/**

```
gr = new GridBagConstraints();  
gr.gridx = 1; gr.gridy =12;  
gr.anchor =GridBagConstraints.WEST ;  
gr.fill =  
GridBagConstraints.HORIZONTAL ;  
gr.ipadx =2;  
gr.insets = new Insets (0,20,2,0);  
c.add (nm,gr) ;
```

## Code Exemple (4/7)

**/\*ajouter le label adresse\*/**

```
gr = new GridBagConstraints();  
gr.gridx =1;gr.gridy =16;  
gr.anchor =GridBagConstraints.WEST ;  
gr.fill =GridBagConstraints.HORIZONTAL ;  
gr.ipadx =2;  
gr.insets =new Insets(0,20,2,0);  
c.add (adr, gr) ;
```

**/\*ajouter le label code postal\*/**

```
gr= new GridBagConstraints();  
gr.gridx =1;gr.gridy =20;  
gr.anchor =GridBagConstraints.WEST ;  
gr.fill =GridBagConstraints.HORIZONTAL ;  
gr.ipadx =2;  
gr.insets = new Insets(0,20,2,10);  
c.add (cp, gr) ;
```

**/\*ajouter la zone pour le prénom\*/**

```
gr= new GridBagConstraints();  
gr.gridx =2;gr.gridy =8;  
gr.fill  
=GridBagConstraints.HORIZONTAL ;  
gr.ipadx =100;  
gr.insets =new Insets(40,0,2,0);  
c.add(txtpr,gr) ;
```



# Code Exemple (5/7)

```
/*ajouter la zone pour le nom*/  
gr = new GridBagConstraints();  
gr.gridx =2;gr.gridy =12;  
gr.fill =GridBagConstraints.HORIZONTAL ;  
gr.ipadx =2;  
c.add(txtnm,gr) ;
```

```
/*ajouter la zone adresse*/  
gr = new GridBagConstraints();  
gr.gridx =2;gr.gridy =16;  
gr.fill =GridBagConstraints.HORIZONTAL ;  
gr.ipadx =2;  
c.add(txtadr,gr) ;
```

```
/*ajouter la zone code postal*/  
gr = new GridBagConstraints();  
gr.gridx =2;gr.gridy =20;  
gr.fill =GridBagConstraints.HORIZONTAL  
gr.ipadx =2;  
c.add(txtcp,gr) ;  
/*ajouter la case pour le nouveau  
banquier*/  
gr = new GridBagConstraints();  
gr.gridx =1;gr.gridy =28;  
gr.anchor =GridBagConstraints.WEST ;  
gr.fill =GridBagConstraints.HORIZONTAL  
gr.ipadx =2;  
gr.insets = new Insets(0,20,2,0);  
c.add(nbq,gr) ;
```

# Code Example (6/7)

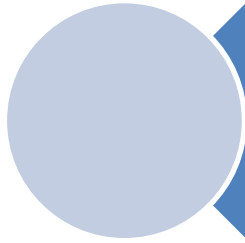
```
/*ajouter du label nouveau banquier*/  
gr = new GridBagConstraints();  
gr.gridx =1;gr.gridy =24;  
gr.anchor =GridBagConstraints.WEST ;  
gr.fill =GridBagConstraints.HORIZONTAL ;  
gr.ipadx =2;  
gr.insets =new Insets(0,20,2,0);  
motpass.setVisible ( false ) ;  
c.add(motpass,gr) ;  
/*ajouter de la zone mot de passe pour  
le nouveau banquier*/  
gr = new GridBagConstraints();  
gr.gridx =2;gr.gridy =24;  
gr.fill =GridBagConstraints.HORIZONTAL ;  
gr.ipadx =2;  
txtpass.setVisible ( false ) ;  
c.add(txtpass,gr) ;
```

```
/*ajouter du bouton valider*/  
gr = new GridBagConstraints();  
gr.gridx =1;gr.gridy =30;  
gr.ipadx =1;  
gr.insets =new Insets(0,0,35,0);  
c.add(valider,gr) ;
```

```
/*ajouter du bouton retour*/  
gr = new GridBagConstraints();  
gr.gridx =1;gr.gridy =40;  
gr.ipadx =1;  
gr.insets =new Insets(0,0,35,0);  
c.add(retour,gr) ;
```

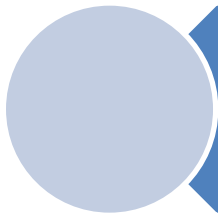
```
pack( );
```

```
}
```



## Code Example (7/7)

```
public void actionPerformed (ActionEvent es)
{
    if ( es.getSource() == nbq)
    {
        motpass.setVisible( true) ;
        txtpass.setVisible( true) ;
    }
    if ( nbq.isSelected ( ) == false)
    {
        motpass.setVisible( false) ;
        txtpass.setVisible( false) ;
    }
}
} // fin de la classe
```



## Aucun Gestionnaire de disposition

Il se peut que, lors de la construction d'une interface graphique que le programmeur ne veuille utiliser aucun des gestionnaires prédéfinis. 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:

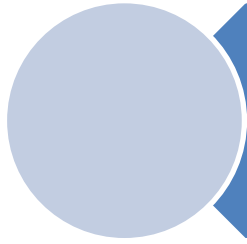
**a = abscisse du point d'insertion du composant,**

**b = ordonnée du point d'insertion du composant,**

**c = largeur du composant,**

**d = hauteur du composant.**

***NB: cette technique demande beaucoup d'attention surtout avec l'usage de setBounds.***



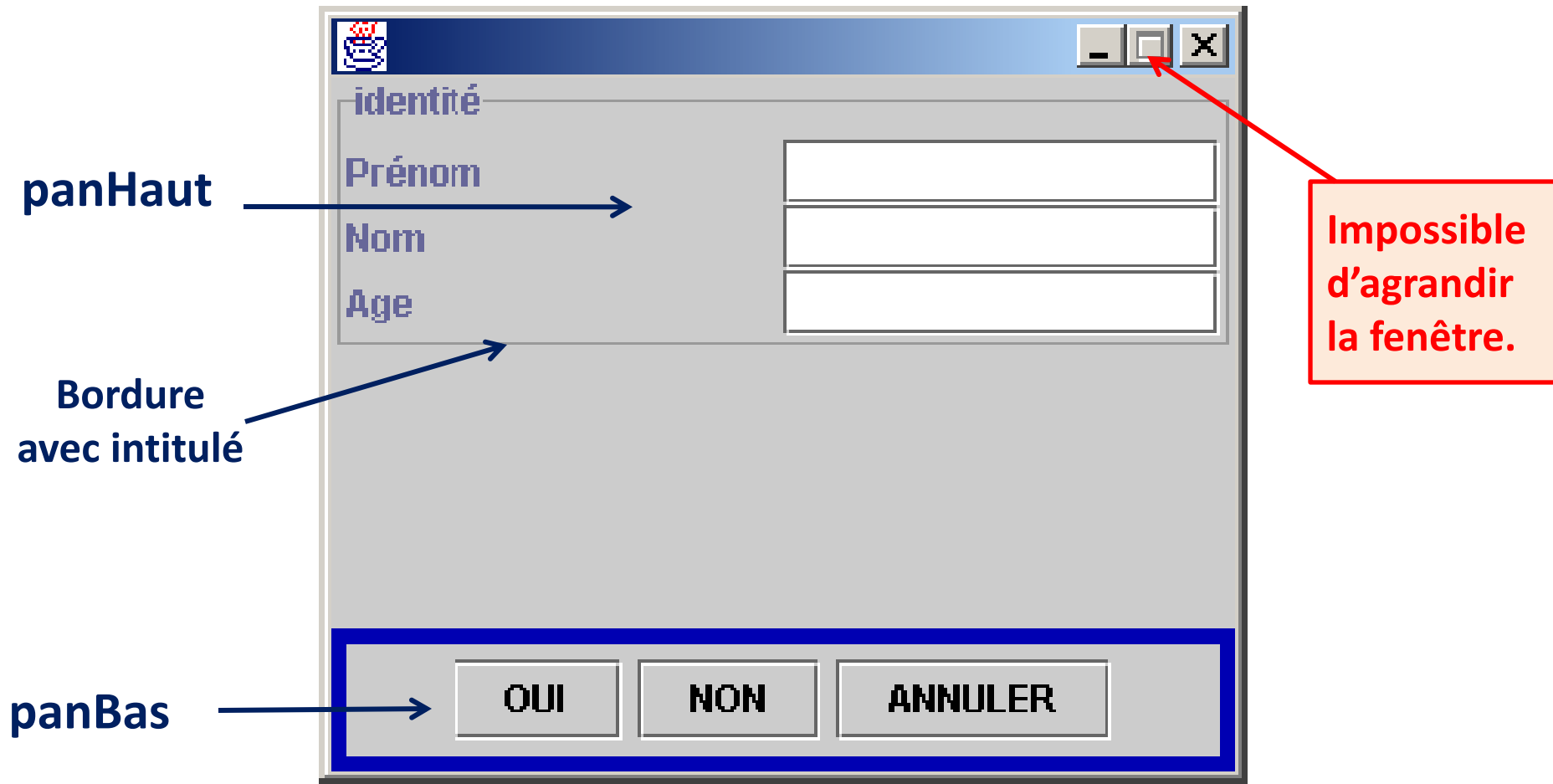
# Objet JPanel

Si vous voulez rangés 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 puisque vous ne pouvez 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 conteneurs puisqu'ils servent à contenir des composants. Un panneau est une sorte de sous fenêtre sans titre, ni bordure.

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

# Exemple de JPanel



# Code Exemple de JPanel

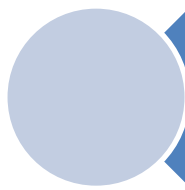
```
import java.awt.*; import javax.swing.*; import javax.swing.border.*;
public class SwingJPanel01 extends JFrame {
    JPanel panHaut, panBas;
    public SwingJPanel01 ( ) {
        /*initialisation du JFrame*/
        super ( );
        this.setSize (new Dimension (300,250));
        this.setResizable ( false ); //on ne pourra pas agrandir la fenêtre
        /*récupération du ContentPane*/
        Container contenu = this.getContentPane ();
        /*création des JPanel avec leur Layout Manager*/
        panHaut = new JPanel(new GridLayout (3,2));
        panBas = new JPanel ( );
    }
}
```

nécessaire  
pour la  
bordure  
avec  
intitulé.

## Code Exemple de JPanel (suite)

```
/*ajout des panneaux au ContentPane, l'un au nord, l'autre au sud*/
contenu.add (panHaut, BorderLayout.NORTH);
contenu.add(panBas, BorderLayout.SOUTH);
/*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 intitulé à panHaut*/
panHaut.setBorder ( new TitledBorder("identité"));
/*ajout d'une bordure épaisse à panBas*/
Border b = BorderFactory.createLineBorder (Color.blue .darker ( ) ,5) ;
panBas.setBorder (b);
}}
```





# Dessiner dans un JPanel

Pour dessiner dans un panneau, il faut *redéfinir* la méthode **paintComponent** (appartenant à la classe **JPanel** ). Il faut alors créer un panneau *personnalisé* c'est à dire une classe dérivée de JPanel puisqu'il y a nécessité de redéfinition.

**NB:** lorsque vous redéfinissez *paintComponent*, prenez la peine d'appeler la méthode de la super classe par **super.paintComponent ( g )** puisque celle-ci appelle la méthode *ComponentUI.update ( )* qui redessine le fond du composant s'il est opaque (JPanel).

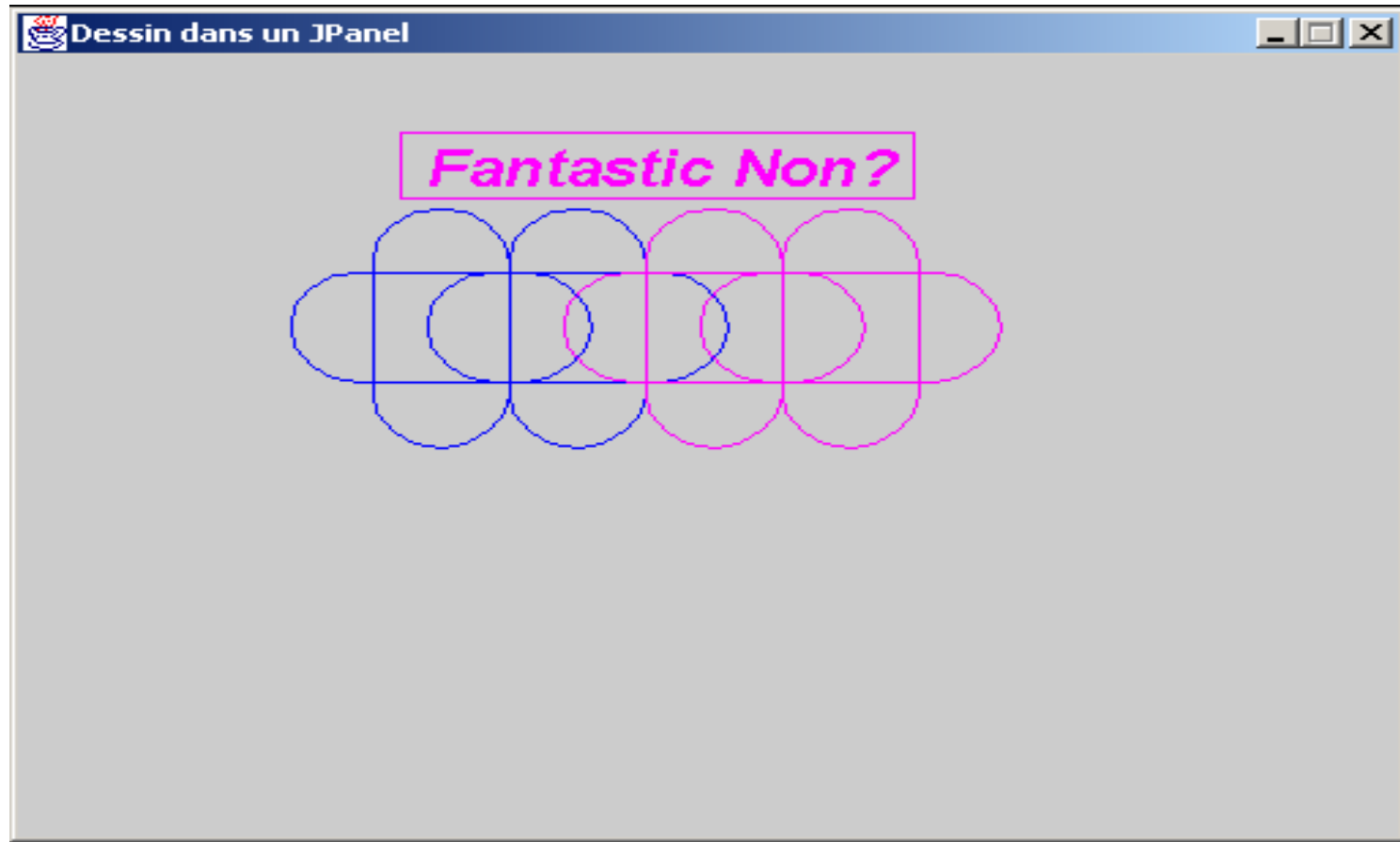
L'entête de la méthode à redéfinir est :

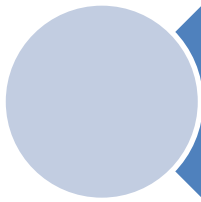
**public void paintComponent (Graphics g)**

L'argument *g* est ce que l'on nomme un **contexte graphique**, c'est un intermédiaire entre les commandes de dessin et leur réalisation effective.

**Voici un exemple d'implémentation d'une classe qui redéfinit **paintComponent**.**

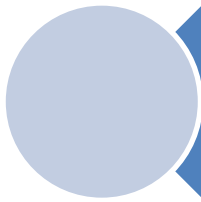
# Exemple de dessin dans un JPanel





## Exemple de dessin dans un JPanel

```
import java.awt.*; import javax.swing.*;
public class SwingDraw01 extends JFrame{
    JPanel pan;
    public SwingDraw01 (String titre) {
        super (titre);
        /*dimension de la fenêtre en fonction de celle de l'écran*/
        Toolkit tk =Toolkit.getDefaultToolkit ( ) ;
        Dimension dim = tk.getScreenSize ( ) ; // on récupère les dimensions de l'écran
        int larg = dim.width /2;
        int haut = dim.height /2;
        this.setSize ( larg, haut);
        this.setResizable ( false ) ;
        /*récupération du ContentPane */
        Container c = this.getContentPane();
        /*création et ajout du panneau au conteneur*/
        pan = new Panneau ( ); pan.setBackground ( new Color (200,150,200,150));
        c.add (pan); } }
```



## Exemple de dessin dans un JPanel (suite)

**/\*création personnalisée d'un panneau\*/**

**class** **Paneau** **extends** JPanel

**{ public void** paintComponent( Graphics g) **//on redéfinit paintComponent**

**{ super.paintComponents (g); // pour redessiner le fond**

**g.setColor (Color.blue.brighter ( ).brighter());**

**g.drawRoundRect ( 100,100,110,50,50,50 );**

**g.drawRoundRect ( 130,70,50,110,50,50 );**

**g.drawRoundRect ( 150,100,110,50,50,50 );**

**g.drawRoundRect ( 180,70,50,110,50,50 ); g.setColor (Color.magenta ) ;**

**g.drawRoundRect ( 200,100,110,50,50,50 );**

**g.drawRoundRect ( 230,70,50,110,50,50 );**

**g.drawRoundRect ( 250,100,110,50,50,50 );**

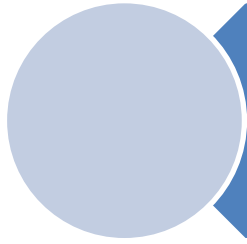
**g.drawRoundRect ( 280,70,50,110,50,50 );**

**g.setFont ( new Font ("Arial",Font.ITALIC + Font.BOLD , 24));**

**g.drawRect ( 140,35,180,30 );**

**g.drawString ("Fantastic Non?",150,60);**

**}}**

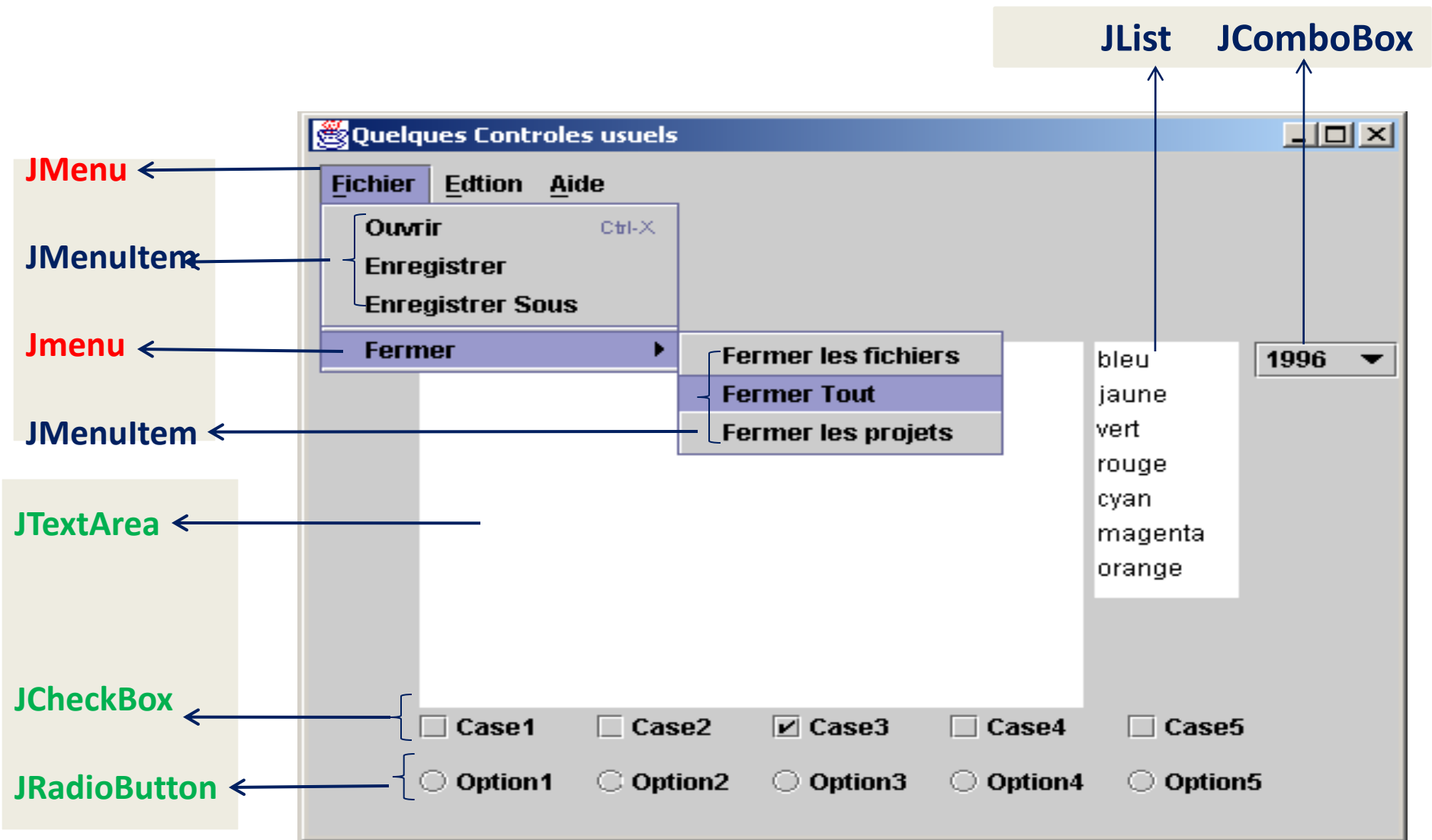


## Quelques contrôles et les menus

Nous allons dans cette partie voir comment créer des contrôles comme des zones de texte sur plusieurs lignes ( **JTextArea** ), des cases à cocher (**JCheckBox**), des boutons d'options (**JRadioButton**), des boîtes de listes (**JList**) et listes combinées (**JComboBox**).

La création de menus est aussi mise en exergue.

# Exemple d'application





## Code Exemple d'application (1/5)

```
public class SwingControls extends JFrame{
protected JCheckBox case1,case2,case3,case4,case5;
protected JRadioButton opbout1, opbout2, opbout3, opbout4, opbout5;;
private JTextArea aire;
protected JList listesimple;
JComboBox listecomplex;
static String annees [ ] = new String[10];
static String couleurs [ ] = {"bleu","jaune","vert","rouge","cyan","magenta","orange"};
static { for ( int i = 0;i < annees.length;i++)
    annees [i] = 1996+i+"";
}
```

## Code Exemple d'application (2/5)

```
public SwingControls(String titre) {
    this.setSize (500,400); this.setTitle (titre);
    Container c = this.getContentPane();
    /*création de panneaux avec leur gestionnaire*/
    JPanel panHaut = new JPanel();panHaut.setLayout (new FlowLayout (0,5,5));
    JPanel panCentre = new JPanel();panCentre.setLayout ( null );
    JPanel panBas = new JPanel();
    /*ajout des panneaux au ContentPane*/
    c.add (panHaut, BorderLayout.NORTH);
    c.add(panCentre, BorderLayout.CENTER);
    c.add(panBas, BorderLayout.SOUTH);
    /*création d'une barre de menus et des menus*/
    JMenuBar barMenu = new JMenuBar();
    JMenu fichier = new JMenu ("Fichier"); fichier.setMnemonic ('F');
    JMenu edition = new JMenu ("Edition"); edition.setMnemonic ('E');
    JMenu aide = new JMenu ("Aide"); aide.setMnemonic ('A');
    JMenuItem ouvrir = new JMenuItem ("Ouvrir");
```

Pour mettre la barre de menu complètement à gauche.

hgap :interstice horizontal

vgap :interstice vertical



## Code Exemple d'application (3/5)

```
/*un accélérateur CTRL X pour le menu ouvrir*/  
ouvrir.setAccelerator (KeyStroke.getKeyStroke (KeyEvent.VK_X,  
InputEvent.CTRL_MASK) ); JMenuItem enregistrer = new  
JMenuItem("Enregistrer");  
JMenuItem enregistrerSous = new JMenuItem("Enregistrer Sous");  
JMenu fermer = new JMenu("Fermer");  
JMenuItem fermerfics = new JMenuItem("Fermer les fichiers");  
JMenuItem fermertout = new JMenuItem("Fermer Tout");  
JMenuItem fermerproj = new JMenuItem("Fermer les projets");  
/*ajout de la barre de menus au panneau panHaut*/  
panHaut.add(barMenu);  
/*ajout des menus a la barre de menus*/  
barMenu.add(fichier,0); barMenu.add (edition); barMenu.add(aide);  
fichier.add(ouvrir); fichier.add (enregistrer); fichier.add(enregistrerSous);  
fichier.addSeparator( ); fichier.add(fermer);  
fermer.add (fermerfics); fermer.add (fermertout); fermer.add (fermerproj);
```

## Code Exemple d'application (4/5)

**/\*zone de texte sur plusieurs lignes\*/**

```
aire = new JTextArea("Ça c'est une zone de texte sur plusieurs lignes.");  
aire.setBounds(50,70,300,200);  
panCentre.add(aire);
```

**/\*les autres contrôles\*/**

```
case1 = new JCheckBox("Case1"); case2 = new JCheckBox("Case2");  
case3 = new JCheckBox("Case3",true);  
case4 = new JCheckBox("Case4"); case5 = new JCheckBox("Case5");  
case1.setBounds (new Rectangle(50,270,80,20)); panCentre.add (case1);  
case2.setBounds ( new Rectangle(130,270,80,20)); panCentre.add (case2);  
case3.setBounds (new Rectangle(210,270,80,20)); panCentre.add (case3);  
ButtonGroup groupe1 = new ButtonGroup();  
case4.setBounds (new Rectangle(290,270,80,20)); panCentre.add(case4);  
case5.setBounds (new Rectangle(370,270,80,20)); panCentre.add(case5);  
groupe1.add(case4); groupe1.add(case5);
```

## Code Exemple d'application (5/5)

```
opbout1 = new JRadioButton("Option1");opbout2 = new JRadioButton("Option2");
opbout3 = new JRadioButton("Option3");
opbout1.setBounds(50,300,80,20); opbout2.setBounds(130,300,80,20);
opbout3.setBounds(210,300,80,20);
opbout4 = new JRadioButton("Option4");opbout5 = new JRadioButton("Option5");
opbout4.setBounds(290,300,80,20);opbout5.setBounds(370,300,80,20);
panCentre.add(opbout1); panCentre.add(opbout2); panCentre.add(opbout3);
panCentre.add(opbout4); panCentre.add(opbout5);
ButtonGroup groupe2 = new ButtonGroup();
ButtonGroup groupe3 = new ButtonGroup();
groupe3.add (opbout1); groupe3.add (opbout2); groupe3.add (opbout3);
groupe2.add (opbout4); groupe2.add (opbout5);
/*création des listes*/
listesimple = new JList(couleurs); listesimple.setBounds (355,70,65,140);
panCentre.add(listesimple);
listecomplex = new JComboBox(annees); listecomplex.setBounds (427,70,65,20);
panCentre.add(listecomplex);} }
```

# Commentaires sur le code

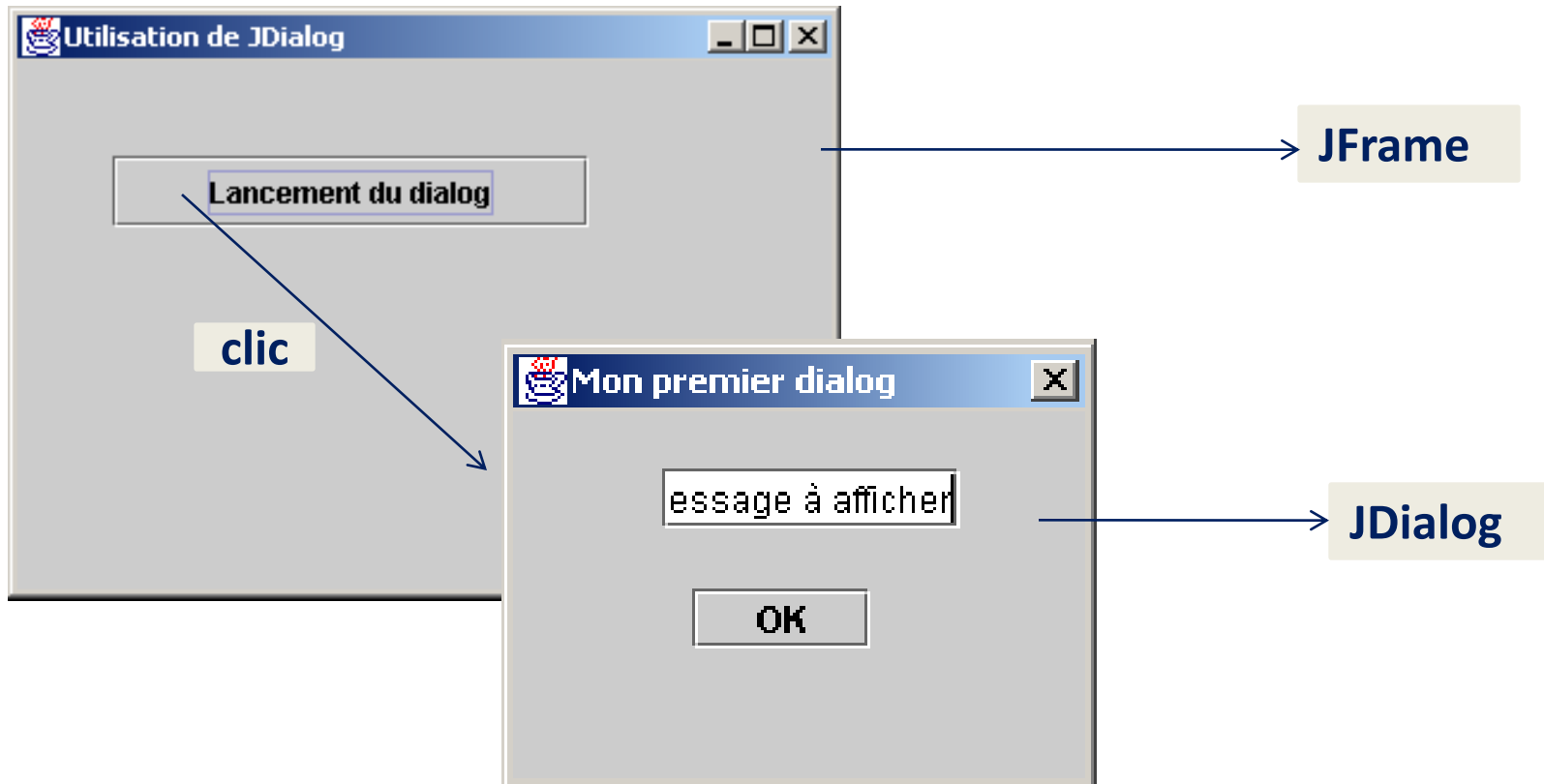
Dans l'utilisation des **JCheckBox**, il est possible de sélectionner *plusieurs cases*, si vous désirez interdire à l'utilisateur de cocher plus de deux cases à la fois (donc une seule case peut être sélectionnée) il faut utiliser un **ButtonGroup** et ajouter les composants à ce dernier. Ainsi une seule case pourra être sélectionnée à la fois.

**ATTENTION:** le **ButtonGroup** n'est pas un composant, donc il ne peut pas être ajouté à un conteneur. Donc même si vous ajoutez des composants à un **ButtonGroup**, il faudra également ajouter ces mêmes composants au conteneur en question (un **Jpanel** par exemple).

Les événements liés aux **JCheckBox** et aux **JRadioButton** sont soit, l'action de l'utilisateur sur le composant soit connaître l'état du composant (sélectionné ou non). Les interfaces qu'ils utilisent sont respectivement **ActionListener** contenant une seule méthode **void actionPerformed (ActionEvent e)** et **ItemListener** contenant aussi une seule méthode **void itemStateChange (ItemEvent e)**.

# Création de boîtes de dialogue

## Utilisation de la classe `JDialog`





## Code de l'interface (1/2)

```
public class SwingDialog extends JFrame implements ActionListener{
JDialog dialog;
JButton lancer, ok;
public SwingDialog (String title) {
    this.setTitle( title);
    this.setSize(350,250);
    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);
}
```

## Code de l'interface (2/2)

```
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 sur le JDialog

Dans l'instruction :

`dialog = new JDialog( this, "Mon premier dialog", true);` on a trois arguments:  
**this** désigne la fenêtre propriétaire (parent) c'est à dire celle contenant le *JDialog*  
**"Mon premier dialog "** désigne le titre de la boîte de dialogue  
**true** la boîte de dialogue est *modale* c'est à dire une fois lancée, l'utilisateur ne peut pas agir sur d'autres que ceux intégrés dans la boîte de dialogue.

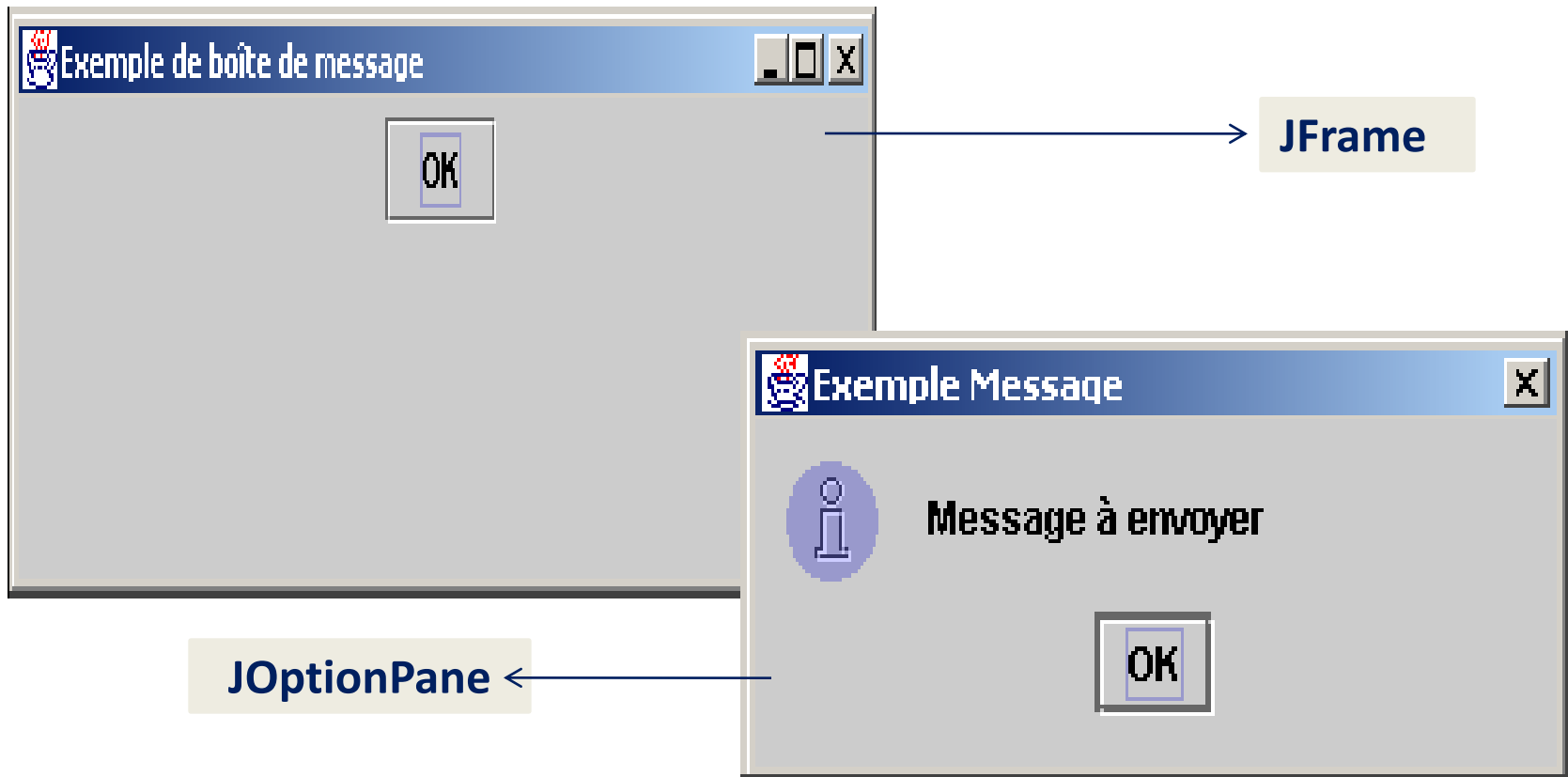
**Remarque** : il est possible (de la même façon qu'on utilise la classe **JFrame**) de créer une classe qui dérive de **JDialog** et d'y ajouter toutes les fonctionnalités dont on souhaite disposer.

Il est aussi possible de créer des boîtes de dialogue sans faire usage de la classe **JDialog**. C'est que nous allons voir dans le paragraphe suivant avec la classe **JOptionPane**.



# La classe: javax.swing.JOptionPane

Les boîtes de Message: **JOptionPane.showMessageDialog**



## Exemple message: JOptionPane.showMessageDialog

```
public class SwingMessage extends JFrame implements ActionListener{
    JButton ouvre;
    public SwingMessage (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","Exemple Message",
                                           JOptionPane.INFORMATION_MESSAGE, null);
        }
```

Fenêtre parent

Objet message

Titre boîte

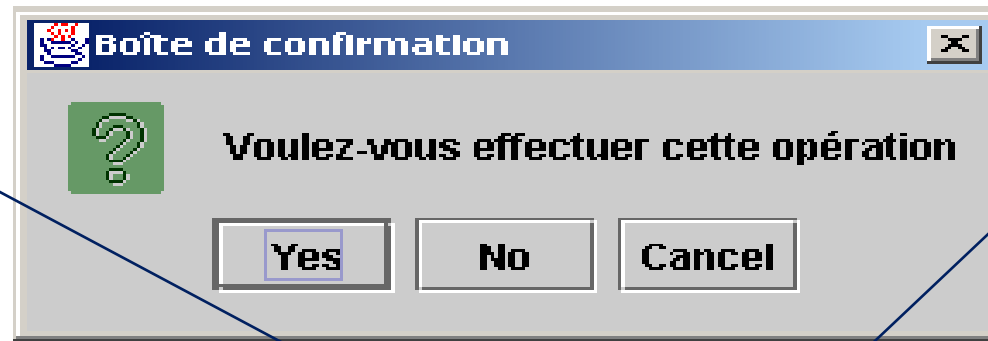
Type du message

Icon de la boîte

# La classe: javax.swing.JOptionPane

## Les boîtes de Confirmation: **JOptionPane.showConfirmDialog**

Une vraie application nécessite toujours le stockage de données sur disque ou sur tout autre média. Avant de faire des sauvegardes permanentes, il est aussi bon de demander une confirmation de la part de l'utilisateur. Pour ce faire, en Java, on peut utiliser les boîtes de confirmation .



*Fenêtre parent*

*message*

*type*

*Type du message*

```
JOptionPane.showConfirmDialog( this,"Voulez-vous effectuer cette opération",  
                                "Boîte de confirmation",  
                                JOptionPane.QUESTION_MESSAGE );
```

# Remarques (1/2)

Les boîtes de confirmation apparaissent par défaut avec des boutons **Yes**, **No** et **Cancel**.

On peut souhaiter n'afficher que les boutons Yes et Cancel; dans ce cas utilisez la méthode `showConfirmDialog (...)` où le quatrième attribut permet de déterminer les boutons qui apparaîtront. Pour le cas évoqué, on fera

```
JOptionPane.showConfirmDialog (this,"Voulez vous effectuer cette  
opération", "Boîte de confirmation",  
JOptionPane.OK_CANCEL_OPTION ,JOptionPane.QUESTION_MESSAGE);
```

Il se peut aussi qu'on veuille effectuer un certain traitement si l'on clique sur l'un des boutons **Yes**, **No** ou **Cancel**. Dans ce cas, récupérer la valeur renvoyée par la méthode **showConfirmDialog** sous une valeur de type entière (int ).

La valeur **0** correspond au clic sur **Yes**, **1** au clic sur **No** et **2** au clic sur **Cancel**.

# Remarques (2/2)

Il peut arriver qu'on veuille personnaliser le nom des boutons **Yes**, **No** et **Cancel** selon la langue utilisée.

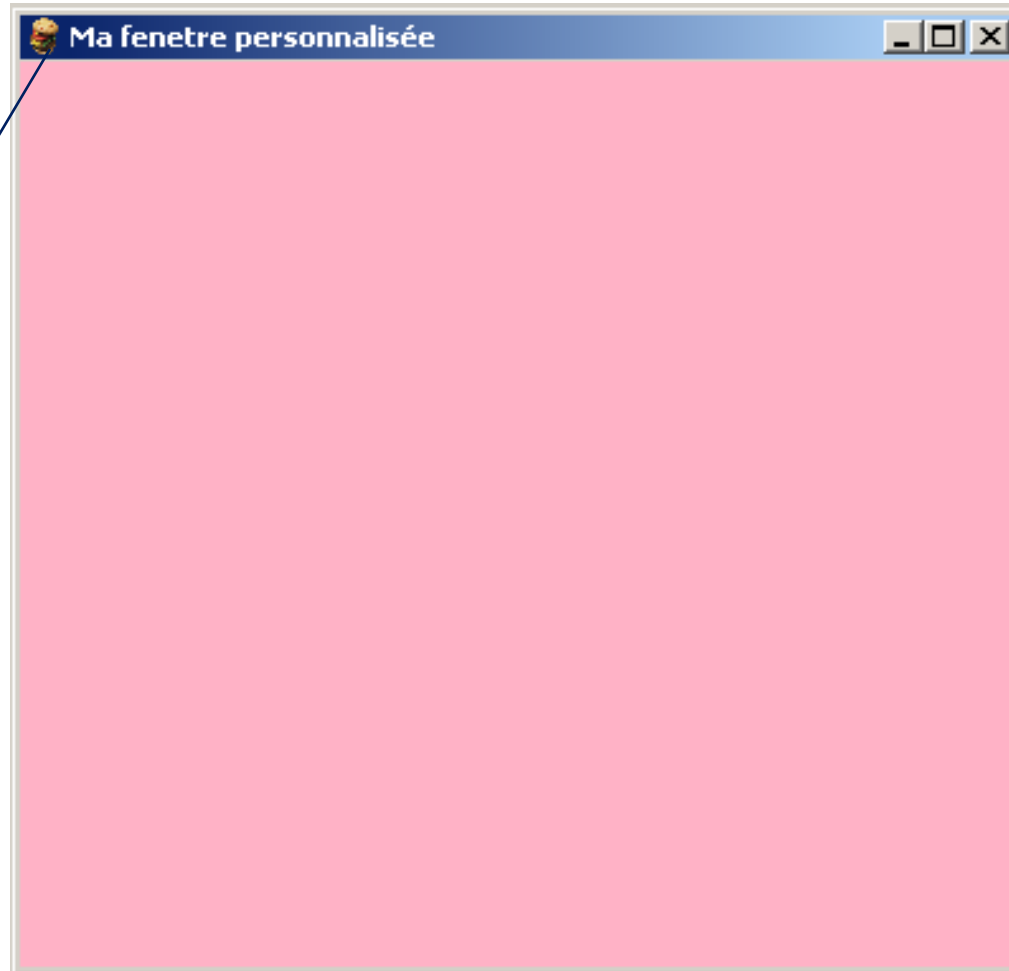
Comment ferait-on par exemple pour remplacer ces boutons par **Oui**, **Non** et **Annuler**?

Utilisez, pour ce faire la méthode `showOptionDialog (...)`.

*/\* la méthode `showConfirmDialog` utilise par défaut des boutons YES, NO et CANCEL; pour les remplacer par OUI, NON et ANNULER, on fait une personnalisation en utilisant `showOptionDialog`\*/*

```
static int openJOptionPaneConfirmDialog (Component comp, String question,String titre)
{ Object options[ ] = {"OUI","NON"}; // on ne tient compte que de ces boutons
return JOptionPane. showOptionDialog (comp,question,titre,
                                       JOptionPane.DEFAULT_OPTION,
                                       JOptionPane.QUESTION_MESSAGE,
                                       null, options, options[0]);
}
```

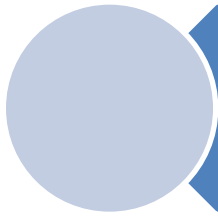
# Personnalisation de l'icône d'un JFrame



***Changer cette icône***

# Exemple de code

```
public class TestJFrame4 {  
    public static void main(String argv[]) {  
        JFrame f = new JFrame("Ma fenêtre personnalisée");  
        f.setSize (400,400);  
        JPanel b = new JPanel();  
        b.setBackground (new Color (255,25,0,125,85)) ;  
        f.getContentPane().add (b);  
        /*rend le bouton de fermeture inactif*/  
        f.setDefaultCloseOperation (WindowConstants.DO_NOTHING_ON_CLOSE);  
        /*change l'icone de la barre de titre du JFrame*/  
        ImageIcon image = new ImageIcon("d:/image000/ burger.gif");  
        f.setIconImage (image.getImage());  
        f.setVisible (true);  
    }  
}
```



# Remarque

Pour modifier l'icône de la fenêtre, vous pouvez utiliser un objet de la classe **Imagelcon** .

**/\*change l'icone de la barre de titre du JFrame\*/**

```
Imagelcon image = new Imagelcon( "d:/image000/ burger.gif" );  
f.setIconImage (image.getImage());  
f.setVisible (true);
```

Vous pouvez aussi passer par la classe **Toolkit**:

**// utilisation d'un toolkit pour l'affichage**

**// d'une icone associée à la fenêtre**

```
Toolkit tk = Toolkit.getDefaultToolkit();  
Image JFramelcon = tk.getImage( "d:/image000/ burger.gif" );  
setIconImage( JFramelcon);
```



