

Les Interfaces Graphiques

Avec l'API SWING

Pr. Omar El Midaoui



**ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR**
Membre de
HONORIS UNITED UNIVERSITIES



Objectif

- Les applications en mode ligne de commande manquent de souplesse et de convivialité.
- Pour cela créer des interfaces graphiques (GUI), ou encore des interfaces homme machine (IHM) augmente l'interactivité et rend l'application plus ergonomique et plus souple.
- Nous allons découvrir dans ce chapitre les bases de la programmation des interfaces graphiques et leurs fonctionnalités capitales, notamment la **gestion des événements** et les **classes internes**.

Tout commence par une fenêtre

Un **JFrame** est l'objet qui représente une fenêtre à l'écran.

Dans laquelle vous placez tous les composants de l'interface :

- boutons,
- cases à cocher,
- champs de texte, etc.
- Elle peut posséder une barre de menus.
- Et elle dispose de toutes les icônes (les widgets) correspondant à votre plate-forme pour agrandir, réduire et fermer la fenêtre.

Tout commence par une fenêtre

L'aspect d'un **JFrame** varie en fonction de votre plate-forme.

Voici un **JFrame** sous Windows :



Un JFrame avec une barre de menus et deux « widgets » (un bouton et un bouton radio).

Construction d'une Fenêtre

Une interface graphique est facile à construire :

- ① Créer un cadre (un JFrame)

```
JFrame cadre = new JFrame();
```

- ② Créer un widget (bouton, champ de texte, etc.)

```
JButton bouton = new JButton("cliquez-moi");
```

- ③ Ajouter le widget au cadre

```
cadre.getContentPane().add(bouton);
```

On n'ajoute rien au cadre directement. Imaginez que le cadre est une bordure qui entoure la fenêtre et que vous ajoutez les widgets au panneau de cette fenêtre.

- ④ L'afficher (fixer sa taille et le rendre visible)

```
cadre.setSize(300,300);  
cadre.setVisible(true);
```

Construction d'une Fenêtre

Une fois que vous avez un **JFrame**, vous pouvez y placer des widgets.

Il existe beaucoup de composants (dans le package **javax.swing**) que vous pouvez ajouter :

Les plus courants sont :

- **JButton**,
- **JRadioButton**,
- **JCheckBox**,
- **JLabel**,
- **JList**,
- **JScrollPane**,
- **JSlider**,
- **JTextArea**,
- **JTextField**
- **JTabbedPane**
- **Jtable**

La plupart sont très simples à utiliser, mais certains (comme **JTable**) peuvent être un peu plus complexes.

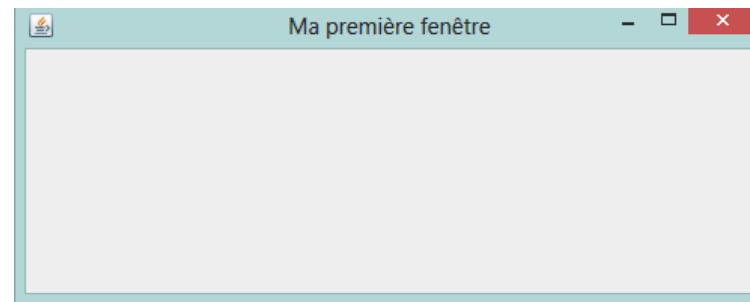
Votre première interface graphique : Une fenêtre vide

```
import javax.swing.JFrame; ← Ne pas oublier d'importer le
                           package swing.

public class SimpleGUI {

    public static void main(String[] args) {
        JFrame fenetre = new JFrame();
        //Définit un titre pour notre fenêtre
        fenetre.setTitle("Ma première fenêtre");
        //Définit sa taille : 500 pixels de large et 200 pixels de haut
        fenetre.setSize(500, 200);
        //Positionner la fenêtre au centre
        fenetre.setLocationRelativeTo(null);
        //Termine le processus lorsqu'on clique sur la croix rouge
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Et enfin, la rendre visible
        fenetre.setVisible(true);
    }
}
```

Enfin, le rendre visible!! (Si vous oubliez cette étape, vous ne verrez rien quand vous exécuterez ce code.)



Cette ligne termine le programme dès que vous fermez la fenêtre. Si vous l'oubliez, elle restera indéfiniment affichée à l'écran.)

Votre première interface graphique : Une Façon plus dynamique

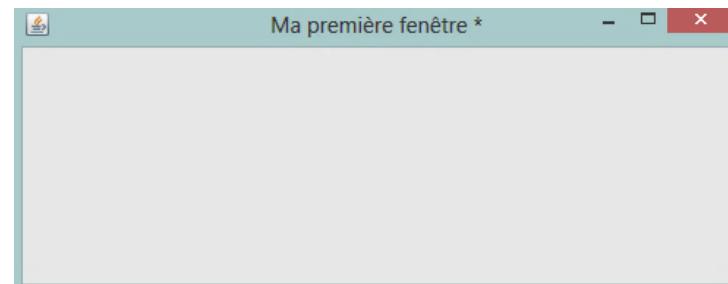
Afin de profiter de la POO, et de pouvoir réutiliser notre objet fenêtre sans répéter à chaque fois le même code, il est préférable de créer une classe pour notre fenêtre !

```
import javax.swing.JFrame;
public class SimpleIHM extends JFrame {

    public SimpleIHM(String Titre){

        this.setTitle(Titre);
        this.setSize(500, 200);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    public static void main(String[] args){

        new SimpleIHM("Ma première fenêtre *");
    }
}
```



Les quatre option de fermeture possible :
DO NOTHING ON CLOSE
HIDE ON CLOSE
DISPOSE ON CLOSE
EXIT ON CLOSE

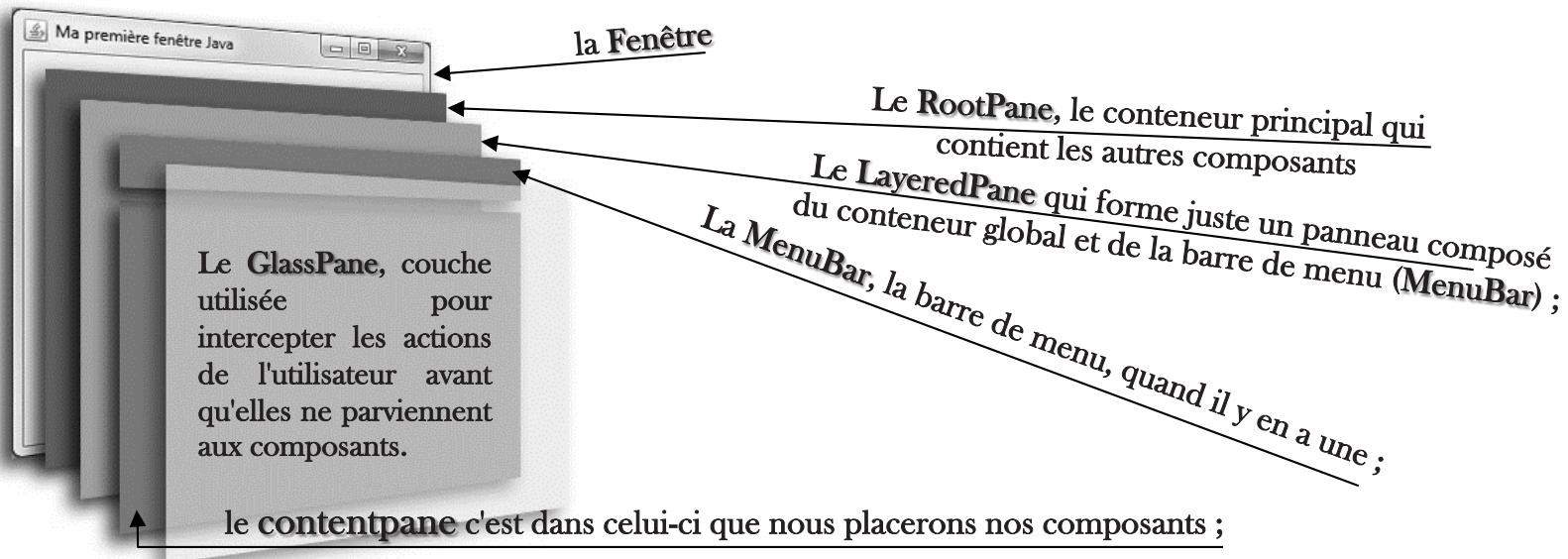
D'autres Fonctions utiles :

- ✓ Positionner la fenêtre à l'écran : **setLocation** (**int** x, **int** y)
- ✓ Empêcher le redimensionnement de la fenêtre : **setResizable** (**boolean** b)
- ✓ Garder la fenêtre au premier plan : **setAlwaysOnTop** (**boolean** b)
- ✓ Retirer les contours et les boutons de contrôle : **setUndecorated** (**boolean** b) → utile pour créer un Splash

```
...  
this.setResizable(true);  
this.setAlwaysOnTop(true); —————→  
this.setUndecorated(true);  
...
```

Peupler Votre Fenêtre

- ✓ Jusqu'à maintenant notre fenêtre est vide, et ne contient aucun contrôle, pour ajouter des éléments à cette fenêtre, il faut encore apprendre une bricole. En effet, notre fenêtre, telle qu'elle apparaît, cache plusieurs couches :



Un Panneau Principale :

Si on veut changer la couleur du fond de notre fenêtre, sachant l'existence de la méthode **setBackground(Color c)**, on va penser directement à ajouter dans notre constructeur de fenêtre l'instruction :

```
this.setBackground(Color.ORANGE);
```

Cependant, en exécutant ce code, la couleur du fond reste inchangée, et cela est du au fait que notre fenêtre a plusieurs couches comme on a vu.

Donc pour remplir notre fenêtre ou changer sa couleur nous allons nous servir uniquement du **Content pane**.

Un Panneau Principale :

Pour le récupérer, il nous suffit d'utiliser la méthode :

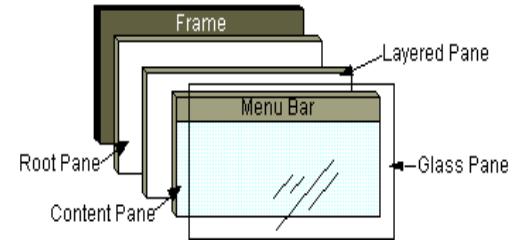
getContentPane() de la classe **JFrame**.

Exemple

```
this.getContentPane().setBackground(Color.ORANGE);
```

ou

```
this.getContentPane().setBackground(new Color(255, 255, 200));
```



La notation RGB

Fenêtre avec couleur de fond

```
import javax.swing.JFrame;
import java.awt.Color;

public class SimpleIHM extends JFrame {

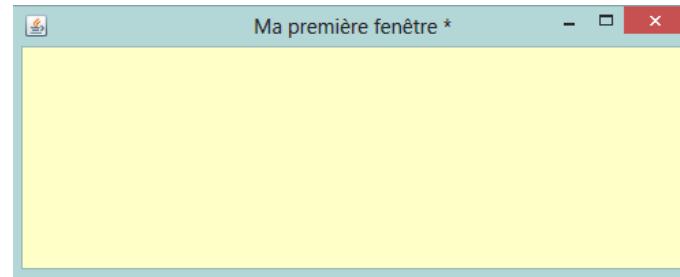
    public SimpleIHM(String Titre){

        this.setTitle(Titre);
        this.setSize(500, 200);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);

        this.getContentPane().setBackground(new Color(255, 255, 200));
    }

    public static void main(String[] args){

        new SimpleIHM("Ma première fenêtre *");
    }
}
```



Remplir Notre Fenêtre

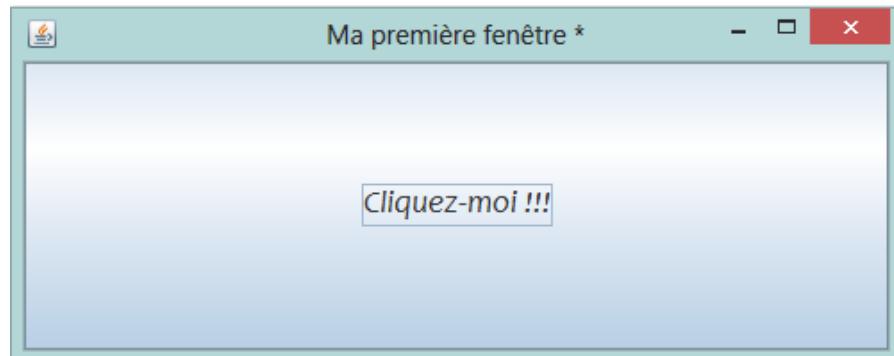
Ne pas oublier d'importer le bouton

```
import javax.swing.JButton;  
  
public class SimpleIHM extends JFrame {  
  
    JButton Click = new JButton("Cliquez-moi !!!");  
  
    public SimpleIHM(String Titre){  
  
        this.setTitle(Titre);  
        this.setSize(500, 200);  
        this.setLocationRelativeTo(null);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setVisible(true);  
        this.getContentPane().setBackground(new Color(255, 255, 200));  
  
        this.getContentPane().add(Click);  
    }  
}
```

*Créer le bouton, avec
un texte à afficher*

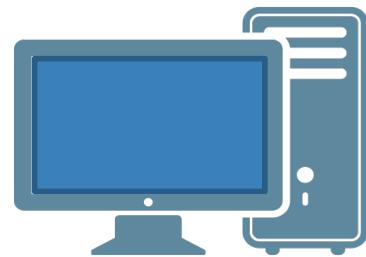
Ajouter le bouton au panneau principale

Exécutons le programme



Le bouton remplit tout l'espace disponible dans le cadre.
Plus tard, nous apprendrons à contrôler son emplacement, sa taille...

Sans layout manager





Le layout manager : null

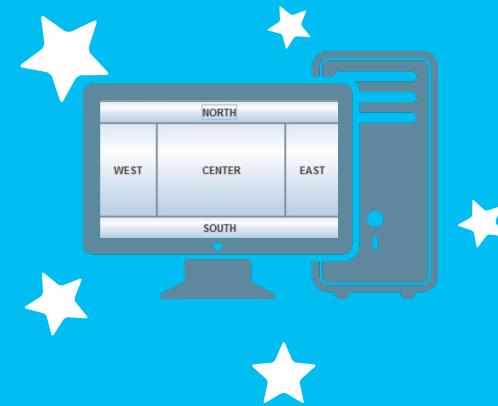
```
Container cp = this.getContentPane();  
cp.setLayout(null);  
cp.setBackground(new Color(255, 255, 200));
```

Dans ce cas les composants sont positionnés manuellement.

```
JPanel p1=new JPanel();  
cp.add(p1);  
p1.setBounds(50,100,400,150);  
p1.setBackground(new Color(205, 205, 100));  
p1.setLayout(null);  
  
JLabel L1=new JLabel("Label : ");  
p1.add(L1);  
L1.setBounds(20,20,300,50);  
L1.setForeground(Color.BLUE);
```



Les composants : **Jlabel, JTextField**





Les étiquettes : JLabel

Pour afficher un texte d'une seule ligne, qui ne peut pas être modifié par l'utilisateur.

Déclaration :

```
JLabel L=new JLabel("Label exemple : ");
```

Définir le positionnement :

```
L.setBounds(20,20,300,50);
```

Définir le contenu du label :

```
L.setText("Nouveau label ");
```

Définir la police, taille et forme de l'écriture :

```
Font f = new Font("Candara", Font.BOLD, 20);
L.setFont(f);
```



Les étiquettes : JLabel

Couleur de l'écriture :

```
L.setForeground(Color.blue);
```

Définir la couleur du fond :

```
L.setOpaque(true);  
L.setBackground(Color.blue);
```

Lire le contenu d'une étiquette :

```
L.getText();
```

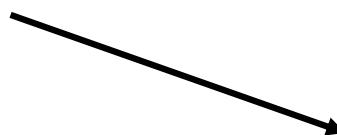


Les étiquettes : JLabel

D'autres méthodes :

```
"public void setHorizontalTextPosition(int textposition)  
"Public void setIcon(Icon image)  
  
"public void setVisible(boolean b)  
"public void setEnabled( boolean b )  
"public void setSize(int w,int h)  
  
"public void setToolTipText(String s)
```

```
Icon monIcone = new ImageIcon("Image.gif");  
  
// Un JLabel  
JLabel monLabel = new JLabel("Mon Label");  
monLabel.setIcon(monIcone);  
monLabel.setHorizontalAlignment(JLabel.RIGHT);  
  
// Un JButton  
JButton monBouton = new JButton("Mon bouton", monIcone);
```



Définit le texte qui sera affiché lorsque la souris passera sur ce composant



Les zones de texte : JTextField

Champs de texte d'une seule ligne modifiable par l'utilisateur,

Exemple avec **JTextField** :

```
public class MyWinTextField extends JFrame{  
    private JTextField T1;  
  
    public MyWinTextField(String T){  
        super(T);  
        T1=new JTextField("Text exemple...");  
        setSize(300,200); setLayout(null);  
        add(T1);  
        T1.setBounds(20,30,200,30);  
        T1.setBackground(Color.CYAN);  
        T1.setForeground(Color.gray);  
        setVisible(true);  
    }  
}
```

Ajout d'article

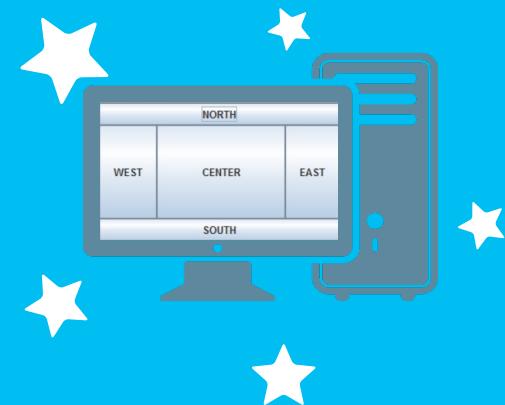


Cas Pratique :
Gestion de Stock

Article
Id
Libelle
Description
PrixA
PrixV
Info()

Fenêtre
Les composants List<Article> L
InitComponents() CreateEvents()

Positionner son composant : les layout managers





les layout managers : objets d'agencement

Dans l'exemple précédent, le bouton était énorme ! il occupait toute la place disponible, parce que le content pane de votre JFrame ne possède pas d'objet d'agencement :

il existe plusieurs sortes de layout managers, dont le rôle est de gérer la position des éléments sur la fenêtre.

Tous ces layout managers se trouvent dans le package **java.awt**.

L'objet **BorderLayout**

L'objet **GridLayout**

L'objet **CardLayout**

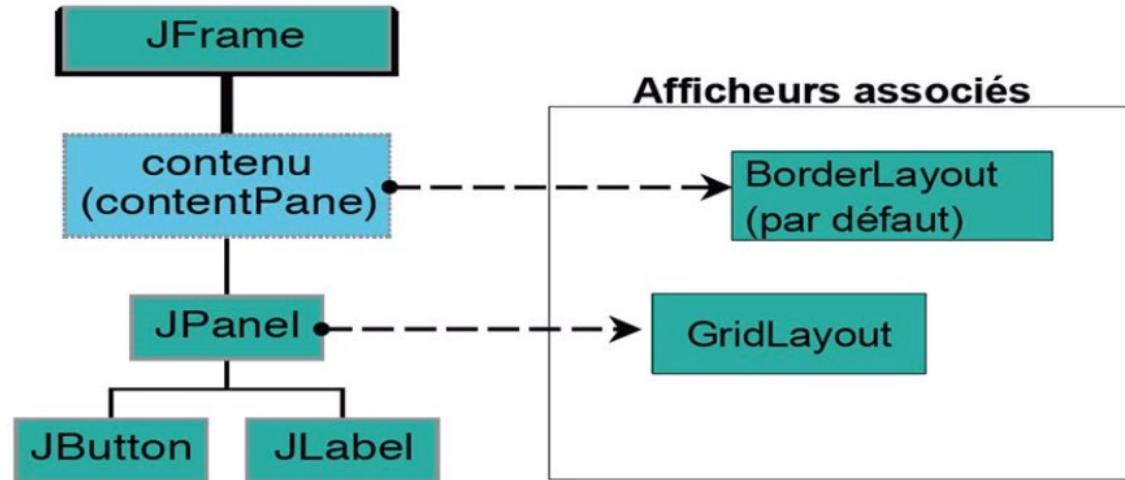
L'objet **FlowLayout**

L'objet **BoxLayout**

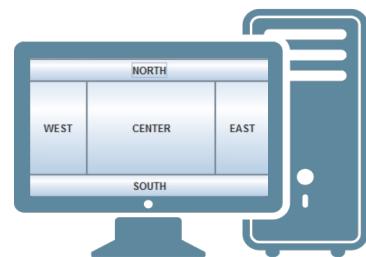


La hiérarchie des composants

Les composants d'une GUI peuvent être représenté sous forme d'arborescente, comme sur l'exemple suivant :



L'objet BorderLayout



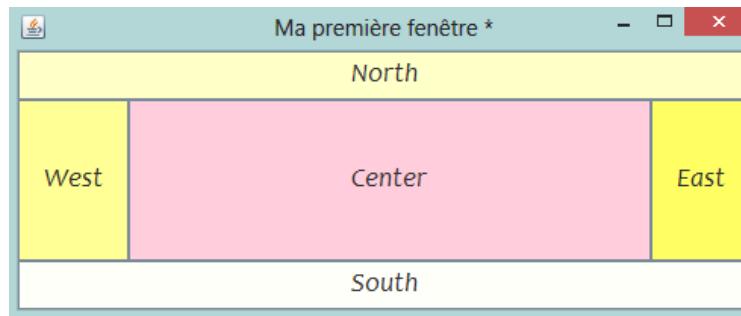


le layout manager : BorderLayout

Le BorderLayout est très pratique si vous voulez placer vos composants de façon simple par rapport à une position cardinale de votre conteneur.

Vous pouvez utiliser les valeurs **NORTH**, **SOUTH**, **EAST**, **WEST** ou encore **CENTER**.

Exemple : fenêtre composée de cinq **JButton** positionnés aux cinq endroits différents que propose un **BorderLayout**.





le layout manager : BorderLayout

Voici le code de cette fenêtre :

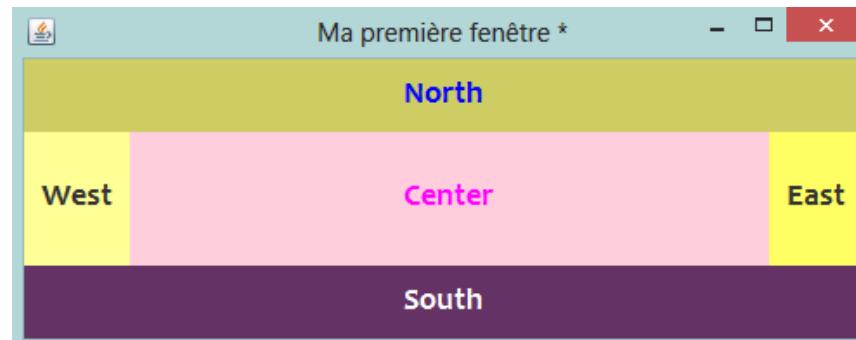
```
...
JButton North = new JButton("North"),
South = new JButton("South"),
East = new JButton("East"),
West = new JButton("West"),
Center = new JButton("Center");
Création des boutons
this.getContentPane().setLayout(new BorderLayout());
Définition de l'objet d'agencement  
Et l'associer au panneau principale  
de la fenêtre
this.getContentPane().add(Center, BorderLayout.CENTER);
this.getContentPane().add(North, BorderLayout.NORTH);
this.getContentPane().add(South, BorderLayout.SOUTH);
this.getContentPane().add(East, BorderLayout.EAST);
this.getContentPane().add(West, BorderLayout.WEST);
Ajouter les boutons aux  
différentes positions
```



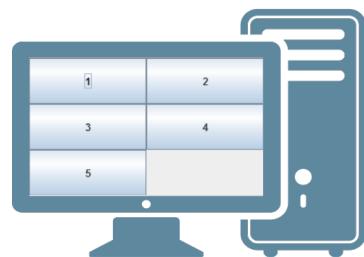
le layout manager : BorderLayout

Pour le formatage des boutons :

```
Font f = new Font("Candara", Font.ITALIC, 18);
...
North.setFont(f);
North.setForeground(Color.BLUE);
North.setBackground(new Color(205, 205, 100));
North.setBorderPainted(false);
...
Center.setFont(f);
Center.setForeground(Color.MAGENTA);
Center.setBackground(new Color(255, 205, 220));
Center.setBorderPainted(false);
```



L'objet GridLayout



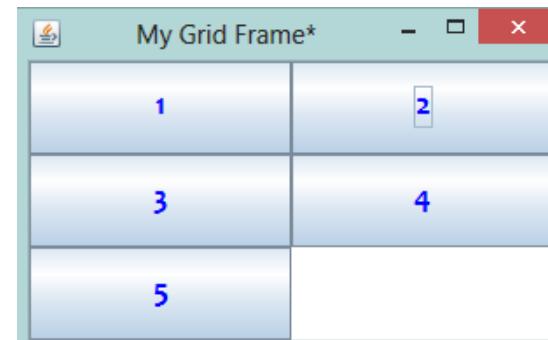


le layout manager : GridLayout

Le **GridLayout** permet d'ajouter des composants suivant une grille définie par **un nombre de lignes** et de **colonnes**. Les éléments sont disposés à partir de la case située en haut à gauche. Dès qu'une ligne est remplie, on passe à la suivante.

Exemple :

fenêtre composée d'une grille de 3 lignes et de 2 colonnes :





le layout manager : GridLayout

```
import java.awt.GridLayout; N'oubliez pas d'importer la classe GridLayout
```

```
JButton one      = new JButton("1"),
two      = new JButton("2"),
three    = new JButton("3"),
four    = new JButton("4"),
five    = new JButton("5");
```

Création des boutons

```
this.getContentPane().setLayout(new GridLayout(3,2));
```

Définition de l'objet d'agencement
Et l'associer au panneau principale de
la fenêtre

```
this.getContentPane().add(one);
this.getContentPane().add(two);
this.getContentPane().add(three);
this.getContentPane().add(four);
this.getContentPane().add(five);
```

Ajouter les boutons
au contentPane



le layout manager : GridLayout

Pour le formatage des boutons :

```
Font f = new Font("Candara", Font.ITALIC, 18);
this.getContentPane().setLayout(new GridLayout(3,2,10,10));
```

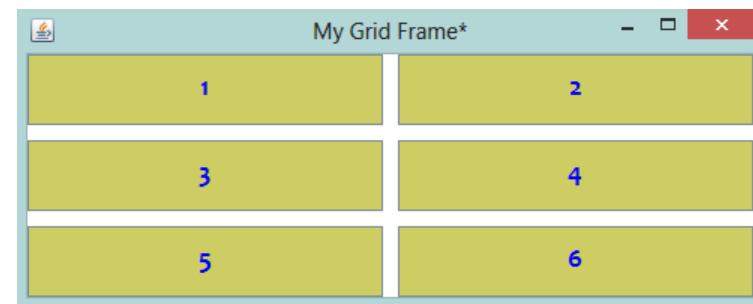
ou

```
GridLayout gl = new GridLayout(3, 2);
gl.setHgap(10);
gl.setVgap(10);
```

```
this.getContentPane().setLayout(gl);
```

```
one.setFont(f);
one.setForeground(Color.BLUE);
one.setBackground(new Color(205, 205, 100));
one.setBorderPainted(true);
```

10 pixels d'espace entre les colonnes et le lignes
(H comme Horizontal) et (V comme Vertical)





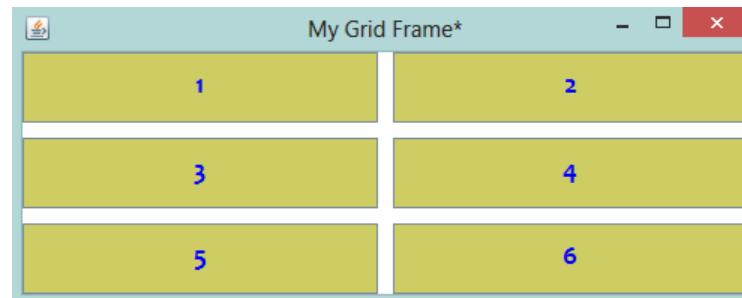
le layout manager : GridLayout

Pour le formatage des boutons :

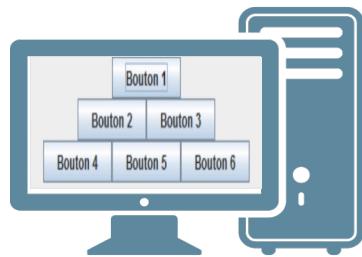
```
Font f = new Font("Candara", Font.ITALIC, 18);
one.setFont(f);
one.setForeground(Color.BLUE);
one.setBackground(new Color(205, 205, 100));
one.setBorderPainted(true);

...
this.getContentPane().setLayout(new GridLayout(3,2,10,10));

...
this.getContentPane().add(one);
this.getContentPane().add(two);
this.getContentPane().add(three);
this.getContentPane().add(foor);
this.getContentPane().add(five);
this.getContentPane().add(six);
```



L'objet BoxLayout





le layout manager : BoxLayout

Le **BoxLayout** permet de ranger vos composants à la suite soit sur une ligne, soit sur une colonne.

BoxLayout.LINE_AXIS Pour les ranger horizontalement et **BoxLayout.PAGE_AXIS** pour les ranger verticalement.

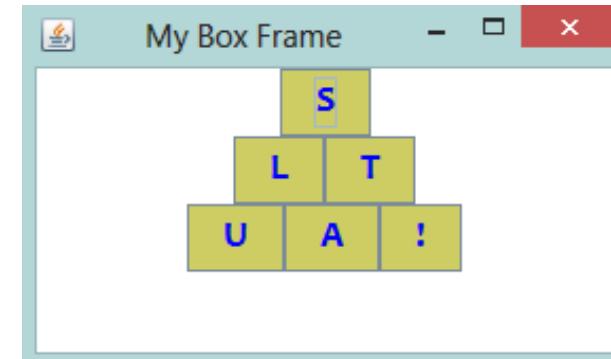
Exemple :

fenêtre composée de 3 Boxes horizontales :

- Box 1 : Bouton S
- Box 2 : Bouton L et T
- Box 3 : Boutons U, A et !

Et un Boxe Verticale :

- Box 4 : Box1, Box2 et Box 3





le layout manager : BoxLayout

```
import javax.swing.BoxLayout; N'oubliez pas d'importer la classe BoxLayout
```

```
JPanel panneau1 = new JPanel(),  
panneau2 = new JPanel(),  
panneau3 = new JPanel(),  
panneau4 = new JPanel();
```

```
JButton Lettre1 = new JButton("S"),  
Lettre2 = new JButton("L"),  
Lettre3 = new JButton("T"),  
Lettre4 = new JButton("U"),  
Lettre5 = new JButton("A"),  
Lettre6 = new JButton("!");
```

```
// Box 1 - Horizontal  
panneau1.setLayout(new BoxLayout(panneau1, BoxLayout.LINE_AXIS));  
panneau1.add(Lettre1);
```

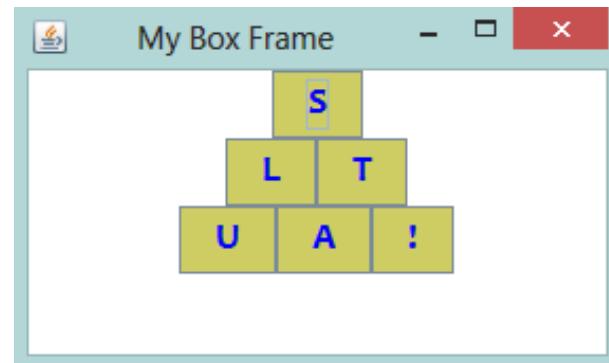
Création des Box et des boutons

Définition de l'objet arrangement
Et l'associer au panneau de chaque Box
Avec la contrainte Line Axis ou Page Axis

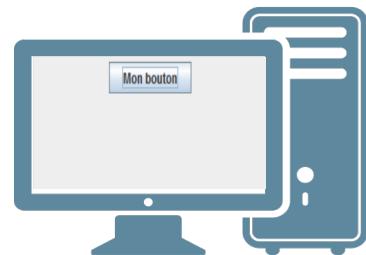


le layout manager : BoxLayout

```
// Box 1 - Horizontal  
panneau1.setLayout(new BoxLayout(panneau1, BoxLayout.LINE_AXIS));  
panneau1.add(Lettre1);  
  
// Box 2 - Horizontal  
panneau2.setLayout(new BoxLayout(panneau2, BoxLayout.LINE_AXIS));  
panneau2.add(Lettre2);  
panneau2.add(Lettre3);  
  
// Box 3 - Horizontal  
panneau3.setLayout(new BoxLayout(panneau3, BoxLayout.LINE_AXIS));  
panneau3.add(Lettre4);  
panneau3.add(Lettre5);  
panneau3.add(Lettre6);  
  
// Box 4 - Vertical  
panneau4.setLayout(new BoxLayout(panneau4, BoxLayout.PAGE_AXIS));  
panneau4.add(panneau1);  
panneau4.add(panneau2);  
panneau4.add(panneau3);  
  
this.getContentPane().add(panneau4);
```



L'objet FlowLayout





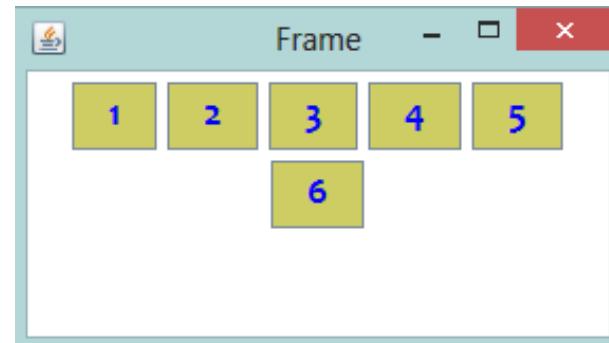
le layout manager : FlowLayout

Le **FlowLayout** est le plus facile à utiliser ! Il se contente de centrer les composants dans le conteneur. C'est Le layout manager défini par défaut dans les objets **JPanel**

Exemple :

fenêtre composée de 6 boutons

```
this.getContentPane().setLayout(new FlowLayout());
```



L'objet CardLayout





le layout manager : CardLayout

Le **CardLayout** permet de gérer vos conteneurs comme un tas de cartes (les uns sur les autres), et basculer d'un contenu à l'autre.

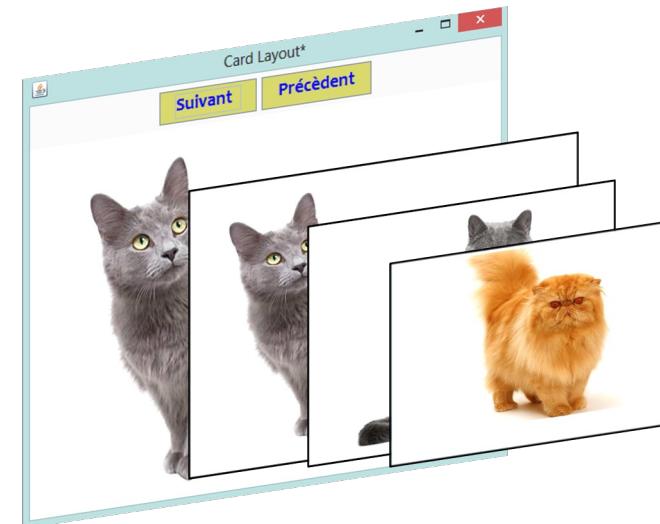
Le principe est d'assigner des conteneurs au layout en leur donnant un nom afin de les retrouver plus facilement

Exemple :

fenêtre composée de 3 Cartes:

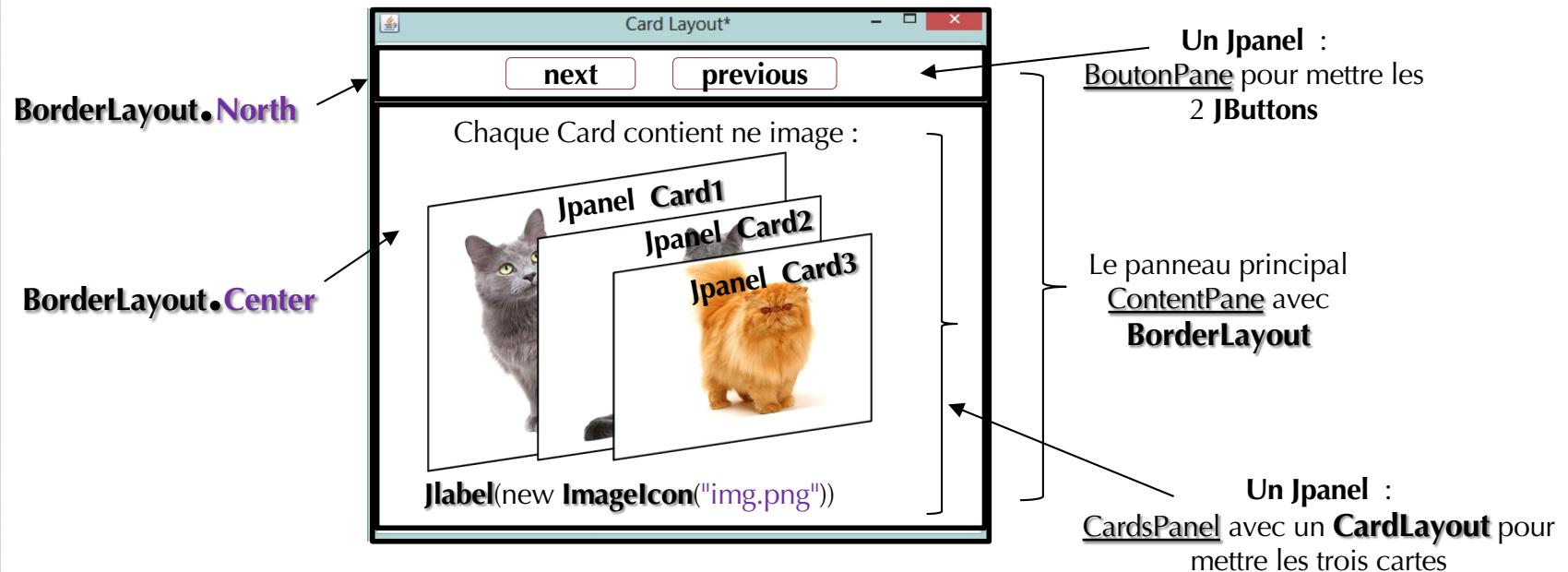
- Carte 1 : image1
- Carte 2 : image2
- Carte 3 : image2

Et deux boutons pour basculer entre ces cartes





le layout manager : CardLayout





le layout manager : CardLayout

```
import javax.swing.BoxLayout; N'oubliez pas d'importer la classe BoxLayout et CardLayout
import java.awt.CardLayout;

JPanel CardsPanel = new JPanel();
// We Create 3 Cards as Panels
JPanel card1 = new JPanel();
JPanel card2 = new JPanel();
JPanel card3 = new JPanel();

JPanel boutonPane = new JPanel();
JButton Suivant      = new JButton("Suivant ");
JButton Precedent    = new JButton("Précédent");

JLabel img1 = new JLabel(new ImageIcon("img1.png"));
JLabel img2 = new JLabel(new ImageIcon("img2.png"));
JLabel img3 = new JLabel(new ImageIcon("img3.jpg"));

Création des 3 Cartes
et du panel englobant les 3 cartes
Création des 2 boutons
et du panel englobant les 2 boutons
Création de 3 images comme Labels
```



le layout manager : CardLayout

```
JLabel img1 = new JLabel(new ImageIcon("img1.png"));
JLabel img2 = new JLabel(new ImageIcon("img2.png"));
JLabel img3 = new JLabel(new ImageIcon("img3.jpg"));

card1.add(img1);
card2.add(img2);
card3.add(img3);

CardLayout cl = new CardLayout();
CardsPanel.setLayout(cl);

CardsPanel.add(card1);
CardsPanel.add(card2);
CardsPanel.add(card3);

boutonPane.add(Suivant);
boutonPane.add(Precedent);

this.getContentPane().setLayout(new BorderLayout());
this.getContentPane().add(boutonPane, BorderLayout.NORTH);
this.getContentPane().add(CardsPanel, BorderLayout.CENTER);
```

Création de 3 images comme Labels

Ajouter à chaque carte l'image correspondante

Définir le CardLayout pour le CardsPanel

Et Ajouter les 3 Cartes (Panneaux)

Ajouter les 2 boutons au boutonPane

Définir le BorderLayout pour le ContentPane

Et Ajouter les 2 Panneaux au nord et au centre

Gestion des événements :
ActionListener





ActionListener

Java **ActionListener** est un objet de la bibliothèque **java.awt.event**, qui est notifié à chaque fois que vous cliquez sur un bouton. **ActionListener** est une interface fonctionnelle ayant une seule méthode **actionPerformed()**, invoquée automatiquement chaque fois que vous cliquez sur un widget (bouton). La notification est faite par un objet de type **ActionEvent**.

Pour Lire **L'ActionEvent** d'un widget (Bouton par exemple) il faut :

1. Implémenter l'interface **ActionListener**.
2. Lier et enregistrer cet objet **ActionListener** auprès du bouton (lui associer cet écouteur d'événements).
3. Retourner vers la méthode de gestion de l'événement et définir l'action à faire
→ Dans l'implémentation la méthode **actionPerformed()** de l'interface **ActionListener**.





ActionListener – Implémentation de l'interface

```
import java.awt.event.ActionEvent;  importer le package awt.event  
import java.awt.event.ActionListener;  
public class MyGUI extends JFrame implements ActionListener{  
    ...
```

```
    Button_Next.addActionListener(this); ② Ajouter les deux boutons à la liste des auditeurs,  
    Button_Previous.addActionListener(this); à gérer par MyGUI dans la méthode actionPerformed
```

```
    @Override  
    public void actionPerformed(ActionEvent e) {  
        if (e.getSource() == Button_Next) {  
            cardLayout.next(CardsPanel);  
        }  
        if (e.getSource() == Button_Previous) {  
            cardLayout.previous(CardsPanel);  
        }  
    }
```

Implémenter l'interface ActionListener.
Notre Classe MyGUI doit implémenter
la méthode actionPerformed

①

③ Implémentation de la méthode actionPerformed
C'est elle qui gère réellement l'évènement



ActionListener – Classe anonyme

On peut gérer les évènements des boutons en implémentant l'interface **ActionListener**, au moment de l'enregistrement de l'écouteur d'évènement auprès du bouton et cela en définissant l'action directement dans le bouton :

```
Button_Next.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        cardLayout.next(CardsPanel);
    }
});

Button_Previous.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        cardLayout.previous(CardsPanel);
    }
});
```



MouseListener –

Java **MouseListener** est un objet de la bibliothèque **java.awt.event**, qui est notifié à chaque fois que vous modifiez l'état de la souris sur un widget.
MouseListener est une interface ayant 5 méthodes, invoquées automatiquement chaque fois qu'une notification est faite par un objet de type **MouseEvent**.

Les cinq méthodes de l'interface **MouseListener** sont données ci-dessous :

- ✓ **mouseClicked(MouseEvent e)**
- ✓ **mouseEntered(MouseEvent e)**
- ✓ **mouseExited(MouseEvent e)**
- ✓ **mousePressed(MouseEvent e)**
- ✓ **mouseReleased(MouseEvent e)**





MouseListener – Exemple

De la même façon :

Pour Lire un **MouseEvent** d'un widget (Notre JFrame par exemple)

Il faut :

1. Implémenter l'interface **MouseListener**.
2. Lier et enregistrer cet objet **MouseListener** auprès de notre JFrame
3. Retourner vers la méthode de gestion de l'événement et définir l'action à faire.

```
public void mouseClicked (MouseEvent e) { lbl_action.setText("Souris Cliqué"); }
public void mouseEntered (MouseEvent e) { lbl_action.setText("Souris Entrée"); }
public void mouseExited (MouseEvent e) { lbl_action.setText("Souris Quittée"); }
public void mousePressed (MouseEvent e) { lbl_action.setText("Appuyée ^_^"); }
public void mouseReleased(MouseEvent e) { lbl_action.setText("Libéré ^_^"); }

public static void main(String[] args) {
    SwingUtilities.invokeLater(()-> new MouseListenerFrame());
}
```

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

public class MouseListenerFrame extends JFrame implements MouseListener
{
    10 usages
    private JLabel lbl_action;
    3 usages
    private Container frameContainer;
    1 usage
    private void initComponents(){
        lbl_action = new JLabel();
        lbl_action.setBounds( x: 90, y: 80, width: 130, height: 20);
        lbl_action.setFont(new Font( name: "Optima", Font.BOLD, size: 28));
        lbl_action.setHorizontalTextPosition(JLabel.CENTER);

        frameContainer = getContentPane();
        frameContainer.setBackground(new Color( r: 203, g: 221, b: 57, a: 255));
        frameContainer.add(lbl_action);
    }

    1 usage
    public MouseListenerFrame(){
        initComponents();
        addMouseListener( l: this);
        setTitle("MouseListener Example");
        setSize( width: 250, height: 250);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
}
```



MouseListener – Exemple

On peut gérer les évènements de souris de notre fenêtre en implémentant l'interface `MouseListener`, au moment de l'enregistrement de l'écouteur d'événement de souris auprès de la fenêtre :

```
public class MouseListenerFrame extends JFrame
{
    10 usages
    private JLabel lbl_action;
    3 usages
    private Container frameContainer;
    1 usage
    private void initComponents(){
        lbl_action = new JLabel();
        lbl_action.setBounds( x: 90, y: 80, width: 130, height: 20);
        lbl_action.setFont(new Font(name: "Optima", Font.BOLD, size: 28));
        lbl_action.setHorizontalAlignment(JLabel.CENTER);

        frameContainer = getContentPane();
        frameContainer.setBackground(new Color( r: 203, g: 221, b: 57, a: 255));
        frameContainer.add(lbl_action);
    }

    public MouseListenerFrame(){
        initComponents();
        addMouseListener(new MouseListener() {
            public void mouseClicked (MouseEvent e) { lbl_action.setText("Souris Cliqué"); }
            public void mouseEntered (MouseEvent e) { lbl_action.setText("Souris Entrée"); }
            public void mouseExited (MouseEvent e) { lbl_action.setText("Souris Quittée"); }
            public void mousePressed (MouseEvent e) { lbl_action.setText("Appuyée ^_^"); }
            public void mouseReleased(MouseEvent e) { lbl_action.setText("Libéré ^_^"); }
        });
        setTitle("MouseListener Example");
        setSize( width: 250, height: 250);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
}
```



MouseListener –

On peut choisir de gérer que deux types des évènements de souris de notre fenêtre (pour définir une action de HOVER par exemple) en implémentant l'interface **MouseAdapter**, au moment de l'enregistrement de l'écouteur d'événement de souris auprès de la fenêtre :

```
public MouseListenerFrame(){
    initComponents();
    addMouseListener(new MouseAdapter() {
        public void mouseEntered (MouseEvent e) { lbl_action.setText("Souris Entrée"); }
        public void mouseExited (MouseEvent e) { lbl_action.setText("Souris Quittée"); }
    });
    setTitle("MouseListener Example");
    setSize( width: 250, height: 250);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
}
```

Boîte à outil Swing





Les composants Swing

JLabel	JTextField
JComboBox	JButton
JCheckBox	JRadioButton
JToggleButton	JTextArea
JFormattedT...	JPasswordField...
JTextPane	JEditorPane
JSpinner	JList
JTable	JTree
JProgressBar	JScrollBar
JSeparator	JSlider

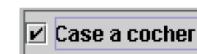
Elle permet de définir des boîtes permettant de choisir une valeur parmi celles proposées. Ses principales méthodes sont :

JComboBox(Object[]) construction avec définition de la liste. On peut utiliser un tableau de chaînes de caractères (`String`) ou de toute autre classe d'objets.
void addItem(Object) qui ajoute une valeur possible de choix
int getSelectedIndex() qui retourne le numéro du choix actuel
Object getSelectedItem() qui retourne l'objet associé au choix actuel. Attention l'objet retourné est de classe `Object`, il faudra utiliser l'opérateur de coercition pour le transformer en sa classe d'origine (`String` par exemple).
void setSelectedIndex(int) qui sélectionne un élément défini par son numéro
void addActionListener(ActionListener) pour associer l'objet qui traitera les choix faits



Elle permet de définir des cases à cocher. Ses principales méthodes sont :

JCheckBox(String) construction avec définition du texte contenu dans la case à cocher
JCheckBox(String,boolean) construction avec en plus définition de l'état initial de la case à cocher
boolean isSelected() qui retourne l'état de la case à cocher (cochée ou non).
void setSelected(boolean) qui définit l'état de la case à cocher (cochée ou non).
String getText() qui retourne le texte contenu dans la case à cocher
void setText(String) qui définit le texte contenu dans la case à cocher
void addActionListener(ActionListener) pour associer l'objet qui traitera les actions sur la case à cocher





Les composants Swing

JLabel	JTextField
JComboBox	JButton
JCheckBox	JRadioButton
JToggleButton	JTextArea
JFormattedT...	JPasswordField...
JTextPane	JEditorPane
JSpinner	JList
JTable	JTree
JProgressBar	JScrollBar
JSeparator	JSlider

Elle permet de définir des listes non déroulantes (pour disposer de listes avec ascenseur il faut faire appel à un contenant possédant des ascenseurs comme **JScrollPane**). La sélection dans ces listes peut porter sur 1 ou plusieurs objets. Ses principales méthodes sont :

JList(Object[]) construction avec définition de la liste. On peut utiliser un tableau de chaînes de caractères (`String`) ou de toute autre classe d'objets.

setListData(Object[]) définition de la liste. On peut utiliser un tableau de chaînes de caractères (`String`) ou de toute autre classe d'objets.

void setVisibleRowCount(int) qui définit le nombre d'éléments visibles sans ascenseur

int getSelectedIndex() qui retourne le numéro du premier élément sélectionné

int[] getSelectedIndices() qui retourne les numéros des éléments sélectionnés

Object getSelectedValue() qui retourne le premier objet sélectionné. Attention l'objet retourné est de classe `Object`, il faudra utiliser l'opérateur de coercition pour le transformer en sa classe d'origine (`String` par exemple).

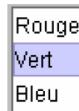
Object[] getSelectedValues() qui retourne les objets actuellement sélectionnés.

void setSelectedIndex(int) qui sélectionne l'élément désigné par son numéro

void setSelectedIndices(int[]) qui sélectionne les éléments désignés par leurs numéros

void clearSelection() qui annule toutes les sélections

void addListSelectionListener(ListSelectionListener) pour associer l'objet qui traitera les sélections dans la liste





Les composants Swing

JLabel	JTextField
JComboBox	JButton
JCheckBox	JRadioButton
JToggleButton	JTextArea
JFormattedT...	JPasswordField...
JTextPane	JEditorPane
JSpinner	JList
JTable	JTree
JProgressBar	JScrollBar
JSeparator	JSlider

Elle permet de définir des zones de texte sur plusieurs lignes modifiables sans ascenseurs. Ses principales méthodes sont :

`JTextArea(String)` qui crée une zone de texte avec un contenu initial
`JTextArea(String,int,int)` qui crée une zone de texte avec un contenu initial et précise le nombre de lignes et de colonnes de la zone de texte
`void append(String)` qui ajoute la chaîne à la fin du texte affiché
`void insert(String,int)` qui insère la chaîne au texte affiché à partir du rang donné
`void setTabSize(int)` qui définit la distance entre tabulations.
`void setLineWrap(boolean)` qui détermine si les lignes longues doivent ou non être repliées.

`void setWrapStyleWord(boolean)` qui détermine si les lignes sont repliées en fin de mot (true) ou pas.
`String getText()` qui retourne le texte contenu dans la zone de texte
`void setText(String)` qui définit le texte contenu dans la zone de texte
`setEditable(boolean)` qui rend la zone de texte modifiable ou pas
`void setSelectedTextColor(Color c)` qui définit la couleur utilisée pour le texte sélectionné
`String getSelectedText()` qui retourne le texte sélectionné
`void select(int,int)` qui sélectionne le texte compris entre les deux positions données en paramètre
`void selectAll()` qui sélectionne tout le texte

Un texte multi-ligne pour taper ce q

Remarque : Lorsque l'on saisit du texte dans une zone de texte celle-ci adapte sa taille au texte saisi au fur et à mesure. Ce comportement n'est pas toujours souhaitable, on peut l'éviter en mettant ce composant dans un `JScrollPane` pour disposer d'ascenseurs.



Les panneaux- JScrollPane

Pour avoir une barre de défilement pour une liste ou une zone de texte, on utilise le type de panneaux **JScrollPane**, comme conteneur de l'objet.

C'est une version améliorée de **JPanel** qui possède des ascenseurs de défilement vertical et horizontal. Ses principales méthodes sont :

JScrollPane() qui crée un **JScrollPane** vide

JScrollPane(Component) qui crée un **JScrollPane** contenant un seul composant (celui passé en paramètre).

JScrollPane(int,int) qui crée un **JScrollPane** vide en précisant le comportement des ascenseurs (voir ci-dessous).

JScrollPane(Component,int,int) qui crée un **JScrollPane** contenant un seul composant (celui passé en paramètre) en précisant le comportement des ascenseurs.

Le premier entier définit le comportement de l'ascenseur vertical, il peut prendre les valeurs :

`JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED` (l'ascenseur n'apparaît que s'il est nécessaire),

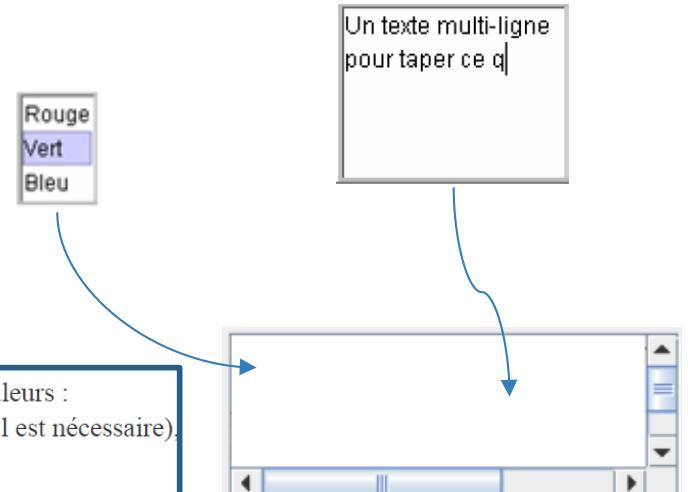
`JScrollPane.VERTICAL_SCROLLBAR_NEVER` (pas d'ascenseur) ou

`JScrollPane.VERTICAL_SCROLLBAR_ALWAYS` (l'ascenseur est toujours présent).

Le dernier entier définit le comportement de l'ascenseur horizontal, il peut prendre les valeurs :

`JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED` (l'ascenseur n'apparaît que s'il est nécessaire), `JScrollPane.HORIZONTAL_SCROLLBAR_NEVER` (pas d'ascenseur) ou

`JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS` (l'ascenseur est toujours présent).





Les composants Swing - JScrollPane

Un exemple avec une JTextArea

```
JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(22, 41, 480, 192);

JTextArea TA = new JTextArea();
TA.setText("JTextArea exemple");
scrollPane.setViewportView(TA);
```

ou

```
JTextArea TA = new JTextArea();
TA.setText("JTextArea exemple");

JScrollPane scrollPane = new JScrollPane(TA);
scrollPane.setBounds(22, 41, 480, 192);
```



Les panneaux- JTabbedPane

Pour une présentation en onglets, on utilise le conteneur de type **JTabbedPane**.

Elle permet de placer les éléments selon des onglets. Ses principales méthodes sont :

JTabbedPane(int) crée l'objet en précisant où se placent les onglets. Le paramètre peut prendre les valeurs :
JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT ou JTabbedPane.RIGHT.

void addTab(String,Component) ajoute un onglet dont le libellé est donné par le premier paramètre et y place le composant donné en second paramètre

void addTab(String,ImageIcon,Component) ajoute un onglet dont le libellé est donné par le premier paramètre et auquel est associée une icône (second paramètre) et y place le composant donné en dernier paramètre

void addTab(String,ImageIcon,Component,String) ajoute un onglet dont le libellé est donné par le premier paramètre et auquel est associée une icône (second paramètre) et y place le composant donné en troisième paramètre. Le dernier est une bulle d'aide qui apparaîtra lorsque la souris passera sur l'onglet.

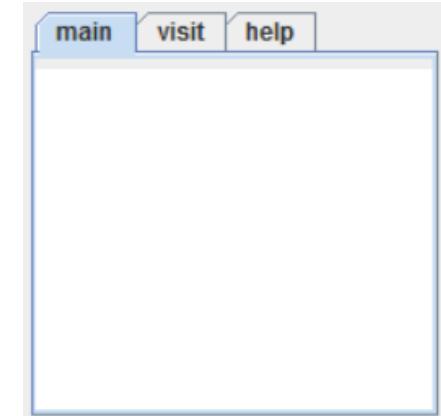
void insertTab(String,ImageIcon,Component,String,int) insère un onglet dont le libellé est donné par le premier paramètre et auquel est associée une icône (second paramètre) et y place le composant donné en troisième paramètre. Le quatrième paramètre est une bulle d'aide qui apparaîtra lorsque la souris passera sur l'onglet. Le dernier indique la position à laquelle doit être inséré l'onglet.

void removeTabAt(int) qui supprime l'onglet désigné.

void setIconAt(int , ImageIcon) associe une icône à l'onglet désigné

int getSelectedIndex() qui retourne le numéro de l'onglet sélectionné

void setSelectedIndex(int) qui sélectionne l'onglet dont le numéro est passé en paramètre





Les panneaux- JTabbedPane

String getTitleAt(int) qui retourne le libellé de l'onglet désigné
void setTitleAt(int,String) qui définit le libellé de l'onglet désigné.
boolean isEnabledAt(int) qui indique si l'onglet désigné est accessible ou pas.
boolean setEnabledAt(int,boolean) qui rend accessible ou pas l'onglet désigné.
int getTabCount() qui retourne le nombre d'onglets.
int indexOfTab(String) qui retourne le numéro correspondant à l'onglet désigné par son libellé.
void setBackgroundAt(int,Color) qui définit la couleur de fond pour la page correspondant à l'onglet désigné par le premier paramètre.
void setForegroundAt(int,Color) qui définit la couleur de tracé pour la page correspondant à l'onglet désigné par le premier paramètre.

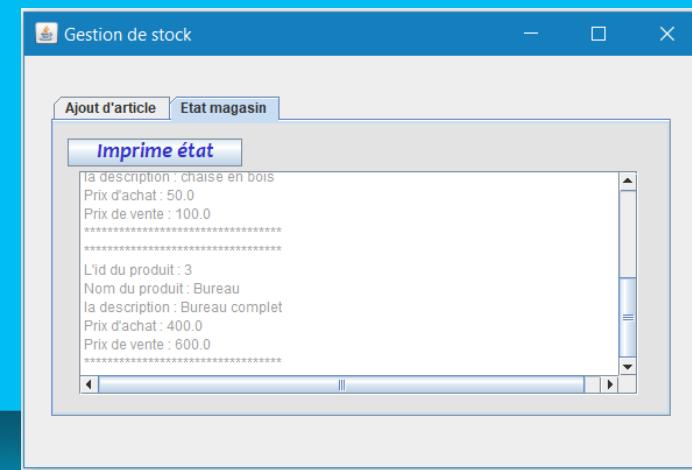
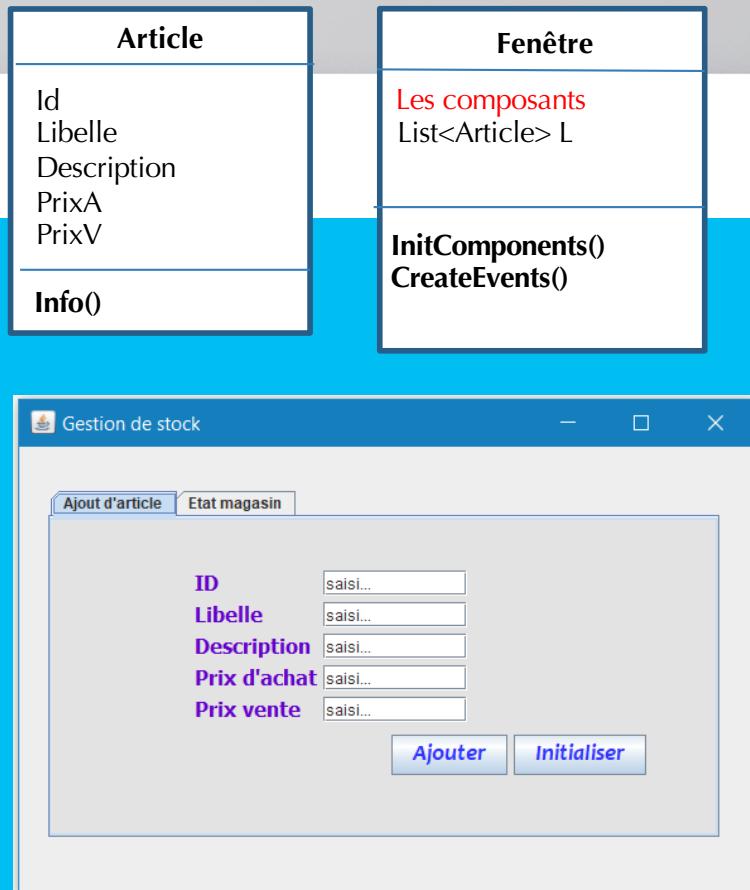
```
JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);
tabbedPane.setBounds(23, 33, 532, 276);
contentPane.add(tabbedPane);

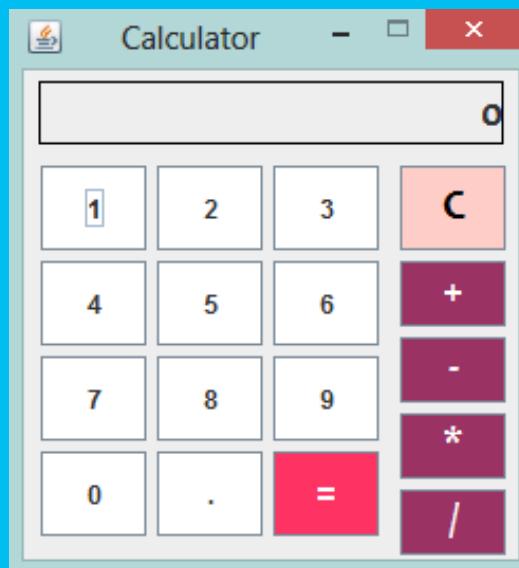
JPanel panel = new JPanel();

tabbedPane.addTab("main", null, panel, "cet onglet représente la vue principale")
```

63

Cas Pratique : ★ Gestion de Stock ★ ★ ★





Cas Pratique : Calculatrice

