

Chapitre :

Les interfaces graphiques en JAVA

A series of horizontal lines in teal and light blue colors, with varying lengths and offsets, creating a modern, layered effect across the middle of the slide.

Généralités sur les interfaces graphiques

Interface avec l'utilisateur

- La quasi-totalité des programmes informatiques nécessitent:
 - l'affichage de questions posées à l'utilisateur
 - l'entrée de données par l'utilisateur
 - l'affichage des résultats obtenus par le traitement informatique
- Cet échange d'informations peut s'effectuer avec une interface utilisateur (UI en anglais) en mode texte (ou console) ou en mode graphique

Généralités sur les interfaces graphiques

Interface Graphique

- Une interface graphique est formée d'une ou plusieurs fenêtres qui contiennent divers composants graphiques (*widgets*) tels que
 - boutons
 - listes déroulantes
 - menus
 - champ texte
 - etc.
- Les interfaces graphiques sont souvent appelés GUI d'après l'anglais *Graphical User Interface*.



API utilisées pour les interfaces graphiques en Java

Les API

- bibliothèques :
 - AWT (*Abstract Window Toolkit, JDK 1.1*)
 - Swing (JDK 1.2)
- Swing et AWT font partie de JFC (*Java Foundation Classes*) qui offre des facilités pour construire des interfaces graphiques
- Swing est construit au-dessus de AWT
 - même gestion des événements
 - les classes de *Swing* héritent des classes de *AWT*

API utilisées pour les interfaces graphiques en Java

Swing ou AWT ?

- AWT (Abstract Window Toolkit)
 - Composants graphiques « lourds »
 - Chaque composant est relié à son équivalent dans l'OS par un « peer »
 - Look & Feel dépendant de l'OS
- SWING
 - Nouveau package
 - Composants graphiques « légers », en pur Java
 - Tous les composants sont détachés de l'OS
 - Look & Feel indépendant de l'OS

API utilisées pour les interfaces graphiques en Java

Swing ou AWT ?

- Tous les composants de AWT ont leur équivalent dans Swing
 - En plus joli
 - avec plus de fonctionnalités
- Swing offre de nombreux composants qui n'existent pas dans AWT
 - Il est fortement conseillé d'utiliser les composants Swing

Swing est plus lourd et plus lent que AWT

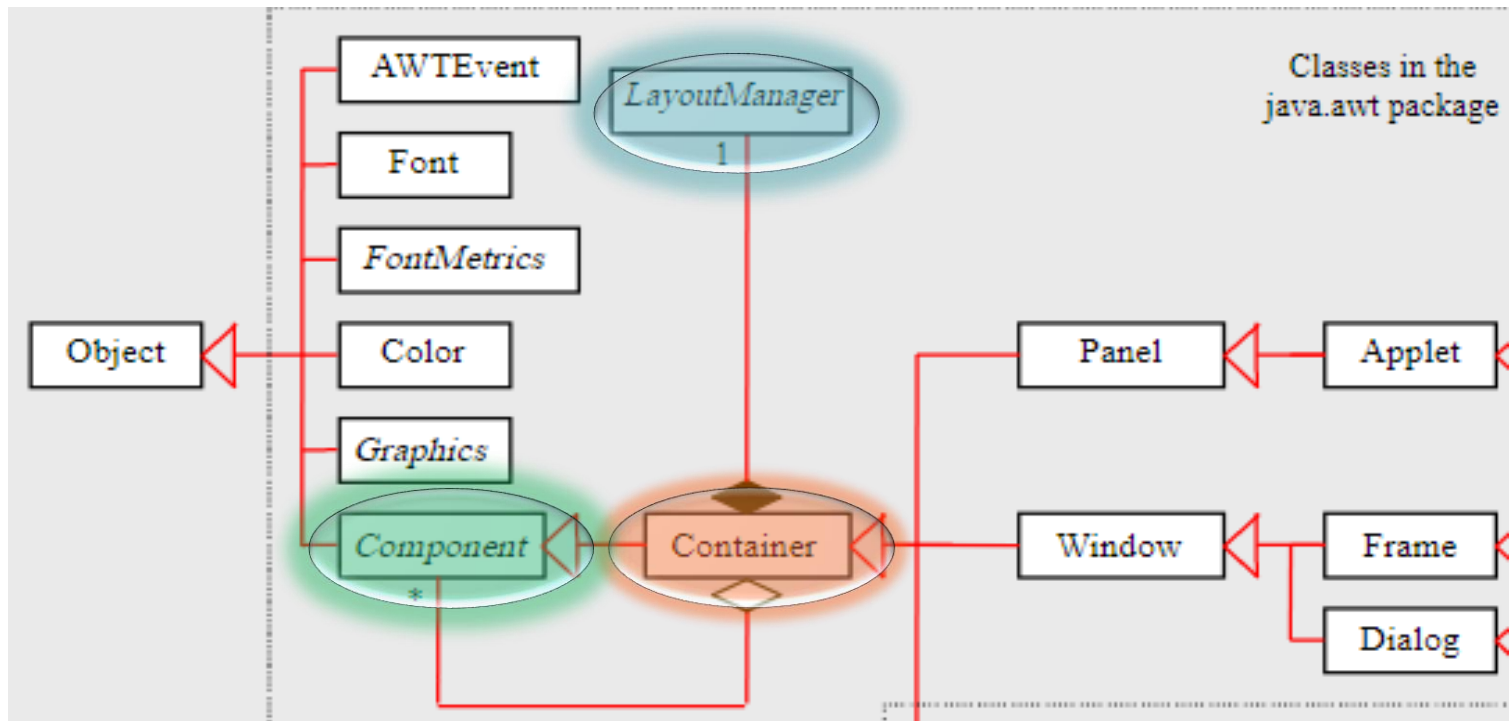
API utilisées pour les interfaces graphiques en Java

- AWT et Swing utilise des classes différentes pour représenter chaque composant graphique.
- Tous les composants ont une classe racine commune (cette classe n'est pas abstraite !!)
- La classe représentant les composants est :
 - pour l'AWT : **Component**
 - pour Swing : **JComponent**
- Notons que tous les noms des composants graphiques de Swing commence par **J** . . .

Composant et hiérarchie de composants

Structure de l'AWT

- L'AWT offre trois types d'éléments graphiques
 - Les « **Containers** » (contenants)
 - Les « **Components** » (composants ou contenus)
 - Les « **LayoutManagers** » (disposition des objets d'un contenant)



Composant et hiérarchie de composants

Structure de l'AWT

Les « Containers »

Sont destinés à accueillir des composants

Gèrent l'affichage des composants

Ex: Frame, Panel, Window

Les « Components »

Constituent différents éléments de l'affichage (boutons, barres de menus, etc.)

Ex: Button, Canvas, Label, Scrollbar, Checkbox

Les « LayoutManagers »

Gèrent la disposition des composants au sein d'un conteneur.



Composant et hiérarchie de composants: AWT

Les « Components »

- Héritage de méthodes:

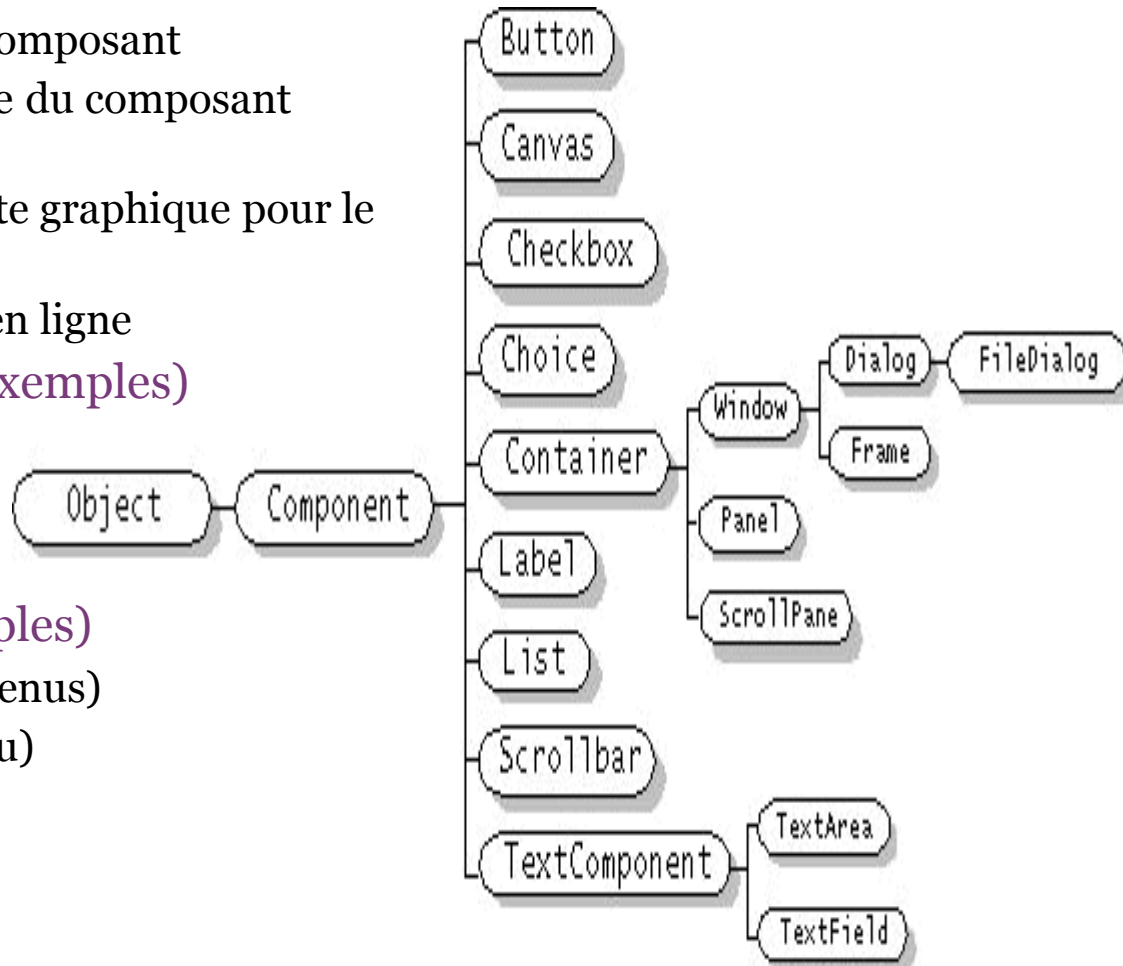
- `paint(Graphics g)` : Affiche le composant
- `repaint()` : Rafraîchit l'affichage du composant (rappelle la méthode `paint`)
- `getGraphics()` : Crée un contexte graphique pour le composant
- Etc. voir documentation Java en ligne

- Composants de formulaires (exemples)

- `Button` (bouton)
- `CheckBox` (case à cocher)
- `Label` (case de texte)

- Composants de fenêtre (exemples)

- `Menu` (Menu d'une barre de menus)
- `MenuItem` (Elément d'un menu)



Composant et hiérarchie de composants: AWT

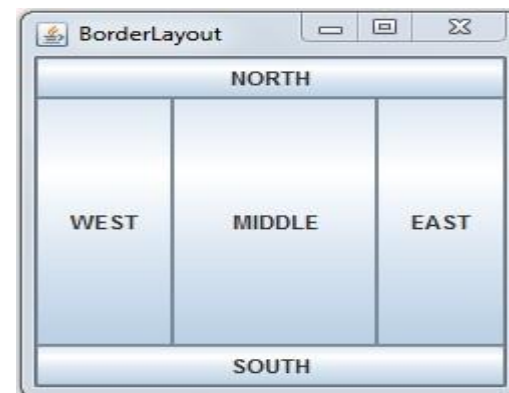
Les « Containers »

- Héritage de méthodes:
 - `add(Component c)` : Intègre le composant spécifié à la fin du container
 - `setLayout(LayoutManager l)` : Configure le `LayoutManager` du container
 - Etc. voir documentation Java en ligne
- La classe « `Frame` »
 - Composant du plus haut niveau
 - La fenêtre d'une application est une instance de cette classe
 - Le `Frame` contient les différents composants graphiques de l'application
 - Ne peut être intégré dans un autre conteneur
- Les classes « `Panel` », « `Window` », « `ScrollPane` », etc.
 - Contenants essentiels
 - Peuvent être intégrés au sein d'un `Frame`

Composant et hiérarchie de composants: AWT

Les « LayoutManagers »

- Rôle
 - Gérer la disposition des composants au sein d'un conteneur
- Types principaux:
 - **BorderLayout**: divise le conteneur en 5 zones
 - **FlowLayout**: rajoute les composants au fur et à mesure
 - **GridLayout**: applique une grille au conteneur pour aligner les composants
 - **CardLayout**: pour un conteneur qui contient plusieurs cartes
 - **GridBagLayout**: grille de cellules élémentaires



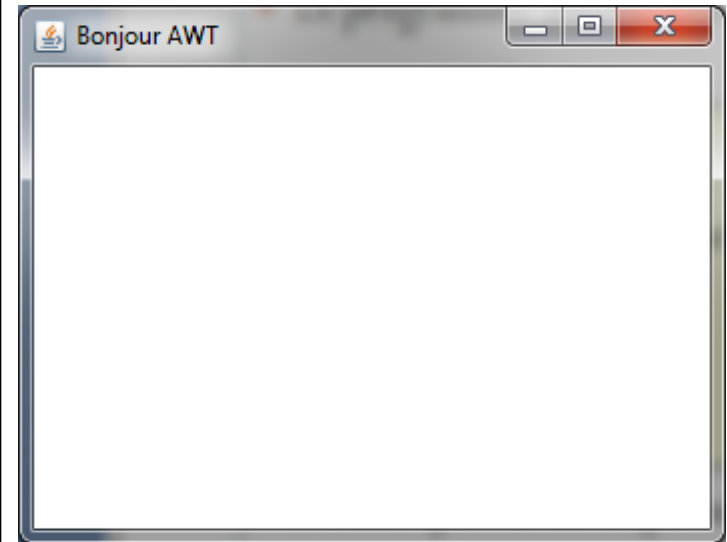
Composant et hiérarchie de composants : AWT

HelloWorld avec AWT

- Le programme affiche une fenêtre ayant pour titre « **Bonjour AWT** ».

```
import java.awt.Frame;

public class HelloAWT {
    public static void main(String[] args) {
        Frame frame = new Frame();
        frame.setTitle("Bonjour AWT");
        frame.setSize(400,300);
        frame.setVisible(true);
    }
}
```

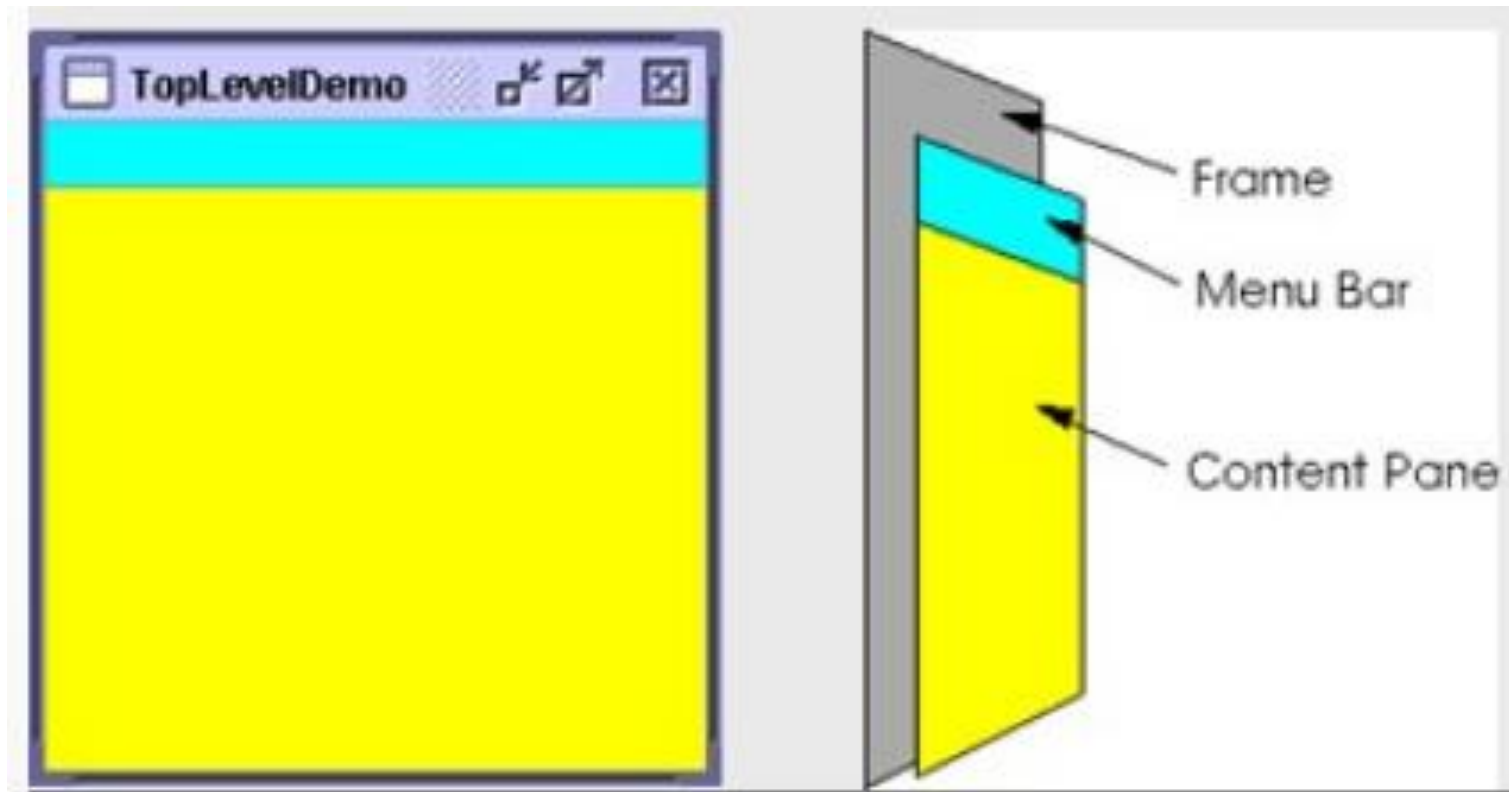


Notons que s'il on clique sur la croix, la fenêtre devient non visible mais l'application continue.

Composant et hiérarchie de composants : AWT

Hiérarchie de composants

- Les composants sont organisés sous la forme d'un arbre

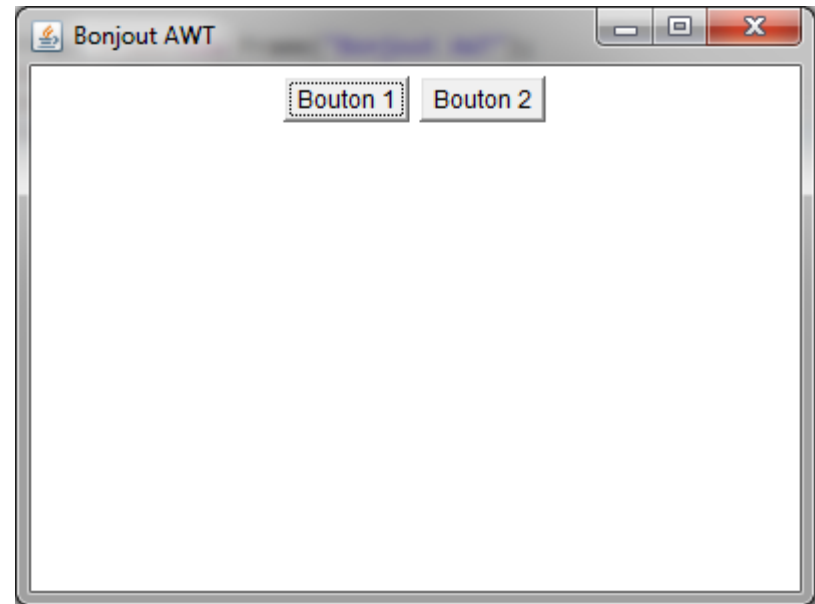


Composant et hiérarchie de composants : AWT

Hiérarchie de composants

- La méthode **add()** permet d'ajouter un composant à un container
- **Panel** est un sous-type de **Container**

```
import java.awt.*;  
public class Hierarchy {  
    public static void main(String[] args) {  
        Button bouton1=new Button("bouton 1");  
        Button bouton2=new Button("bouton 2");  
  
        Panel panel=new Panel();  
        panel.add(bouton1);  
        panel.add(bouton2);  
  
        Frame frame=new Frame("Bonjour AWT");  
        frame.add(panel);  
        frame.setSize(400,300);  
        frame.setVisible(true);  
    }  
}
```



Composant et hiérarchie de composants : AWT

Hiérarchie de composants

Chaque composant possède un état indiquant s'il est visible ou non (**is/setVisible()**)

Un changement d'état de la visibilité se propage sur l'ensemble des composants fils du composant.

Les composants **Frame**, **Dialog**, **Window**, **Applet** possèdent aussi la méthode **setVisible()**

```
import java.awt.*;
public class Hierarchy {
    public static void main(String[] args) {
        Button bouton1=new Button("bouton 1");
        Button bouton2=new Button("bouton 2");

        Panel panel=new Panel();
        panel.add(bouton1);
        panel.add(bouton2);

        Frame frame=new Frame("Bonjour AWT");
        frame.add(panel);
        frame.setSize(400,300);
        frame.setVisible(true);
    }
}
```

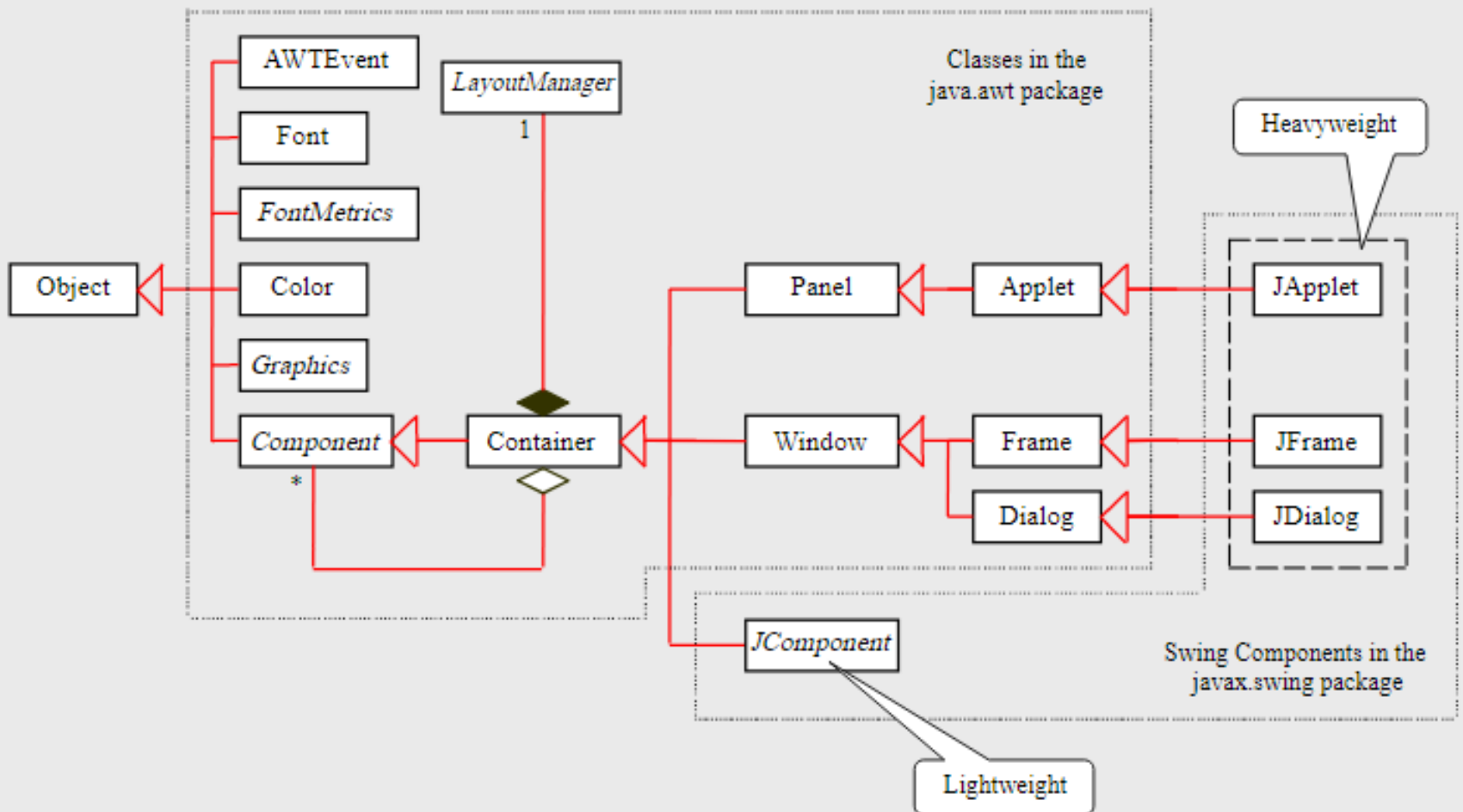

Composant et hiérarchie de composants

Relation entre Swing et AWT

- Swing utilise l'AWT pour ouvrir une fenêtre
- Il y a deux types de composants Swing
 - les **heavyweight** : gérés par l'AWT, correspondent à des fenêtres de la plateforme : **JFrame**, **JDialog**, **JWindow**, **JApplet**
 - les **lightweight** : composants Java qui effectuent le dessin
- Tous les composants AWT sont heavyweights

Composant et hiérarchie de composants

Relation entre Swing et AWT

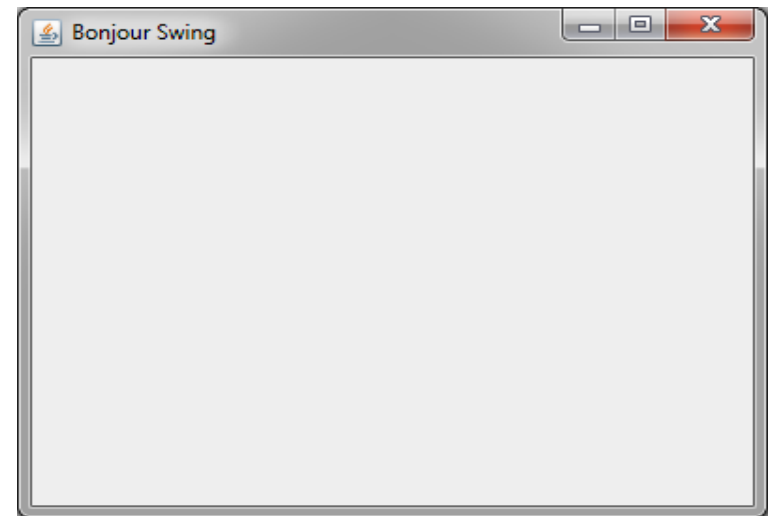


Composant et hiérarchie de composants : SWING

HelloWorld avec Swing

- Le programme affiche une fenêtre ayant pour titre « Bonjour Swing ».

```
import javax.swing.JFrame;  
public class HelloSwing {  
    public static void main(String[] args)  
    {  
        JFrame frame = new JFrame();  
        frame.setTitle("Bonjour Swing");  
        frame.setSize(400, 300);  
        frame.setVisible(true);  
    }  
}
```

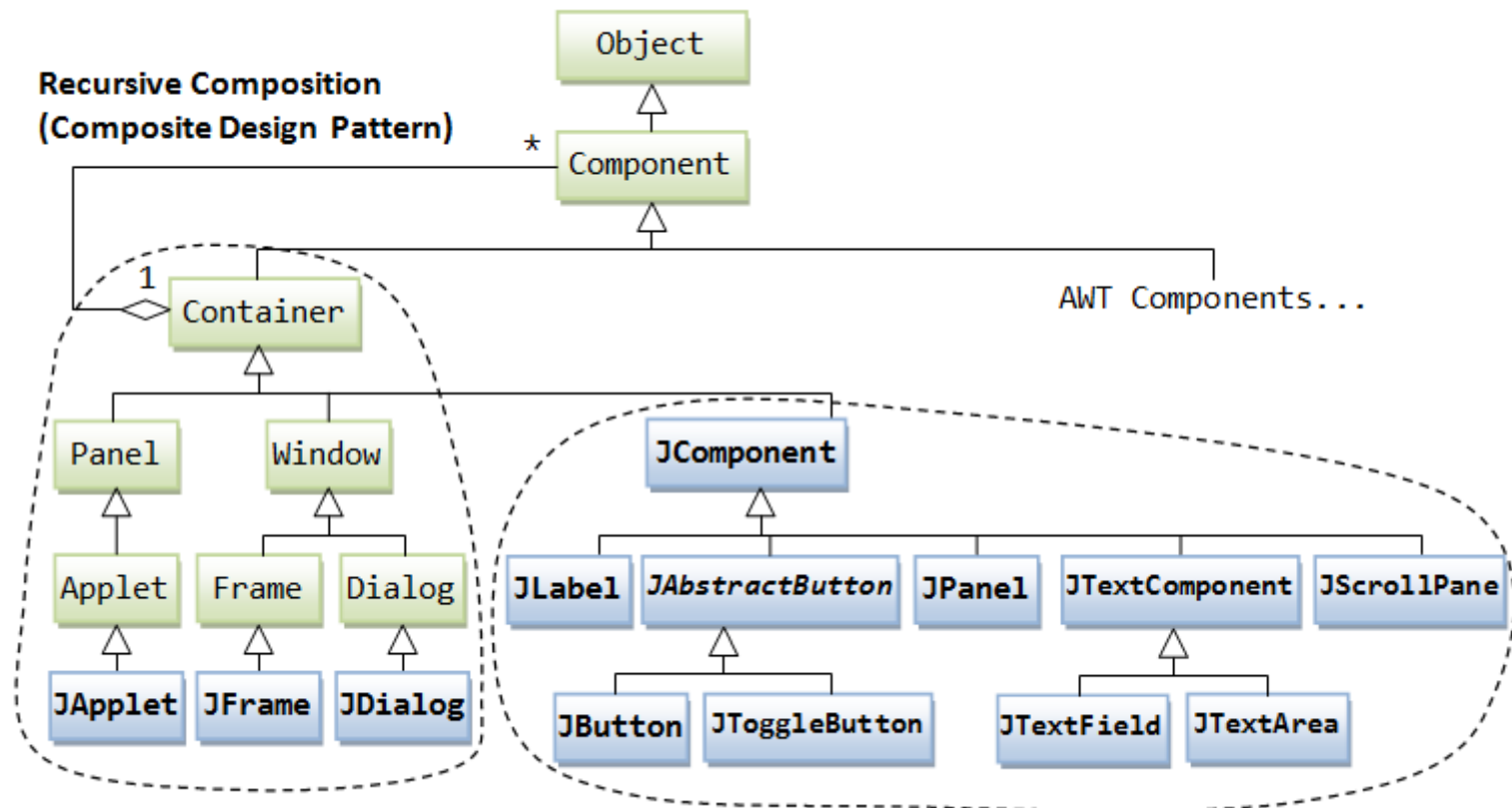


Notons que s'il on clique sur la croix, l'application ne se ferme toujours pas.

Composant et hiérarchie de composants : SWING

Composants de Swing

- Les composants de Swing partagent une partie de leur implantation avec ceux de l'AWT
- Il y a beaucoup plus de composants Swing que de composants AWT



Composant et hiérarchie de composants : SWING

Composants de Swing

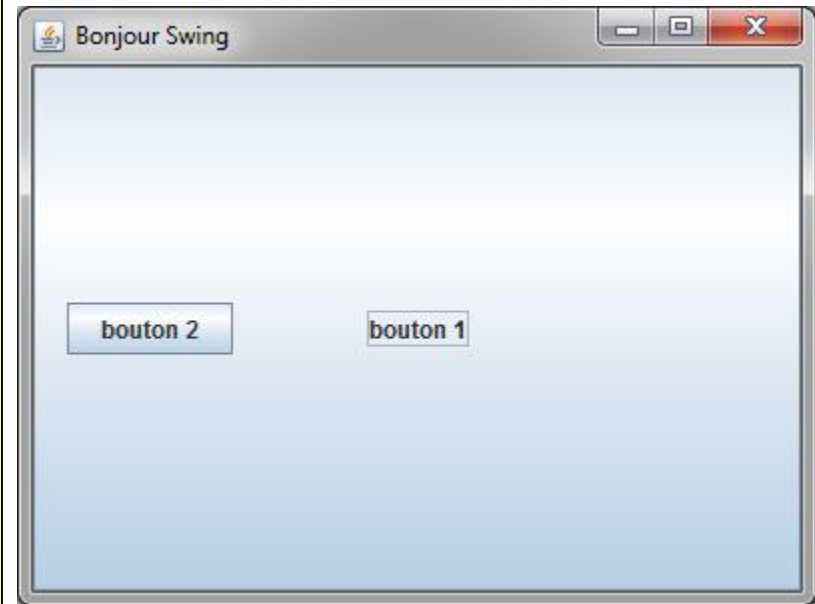
- Problème de design n'importe quel **JComponent** est un **Container**
- Le code suivant est donc possible :

```
import javax.swing.*;

public class PbDesign {
    public static void main(String[] args) {
        JFrame frame=new JFrame();

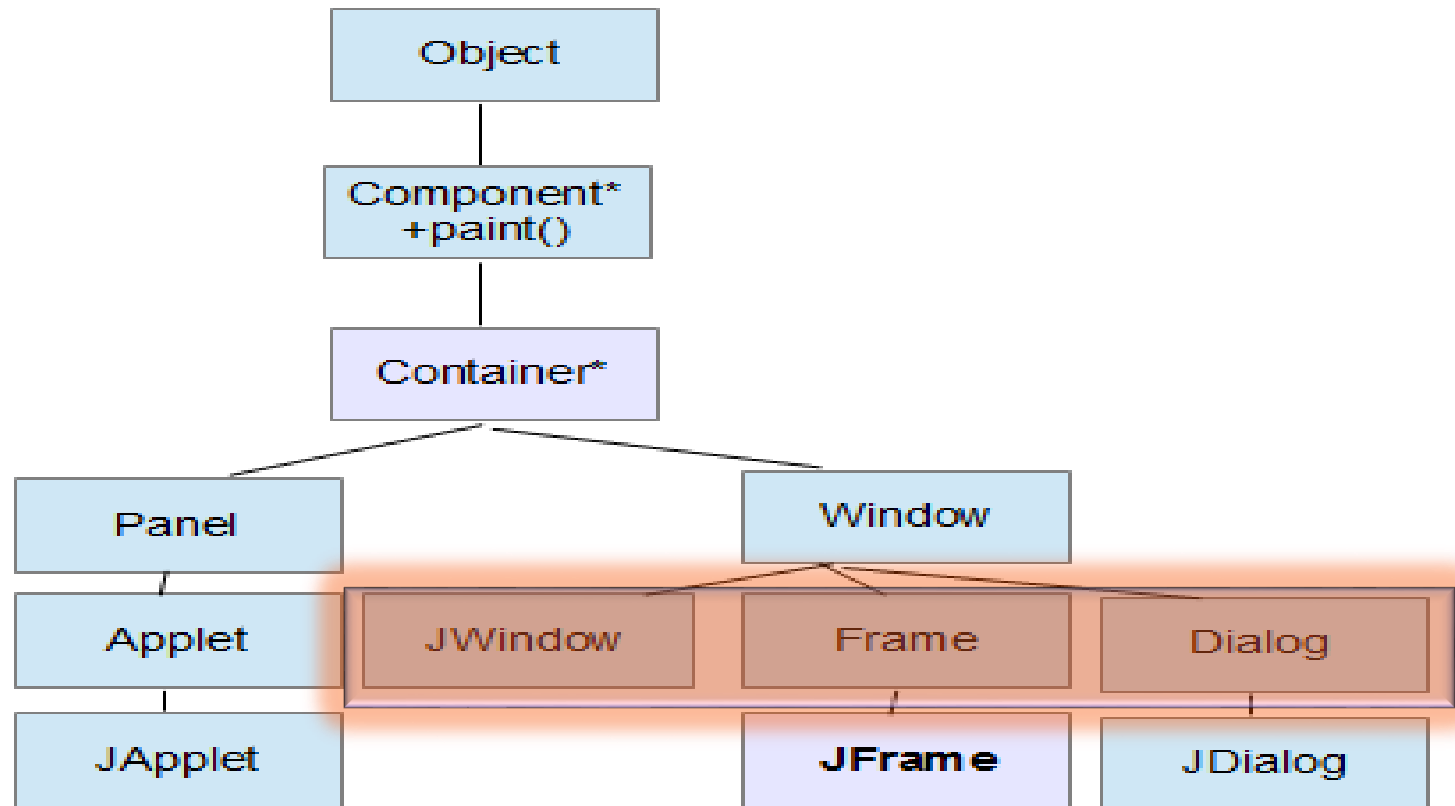
        JButton bouton1=new JButton("bouton 1");
        JButton bouton2=new JButton("bouton 2");
        bouton1.add(bouton2);

        frame.setContentPane(bouton1);
        frame.setTitle("Bonjour Swing");
        frame.setSize(400,300);
        frame.setVisible(true);
    }
}
```



Les fenêtres

- Il existe plusieurs types de fenêtres dans Swing :



Les fenêtres

- Swing propose deux types de fenêtres : **JWindow** et **JFrame**.
 - Composants proches et descendant **Window** (AWT).
- Constructeur est appelé pour créer une fenêtre
- Par défaut, une fenêtre créée n'est pas affichée
 - La méthode **setVisible** permet d'afficher une fenêtre.
- La taille d'une fenêtre dépend des éléments qu'elle contient.
- Pour éviter d'estimer ou de calculer la taille, Swing propose une méthode (**pack**) qui calcule la taille de la fenêtre en fonction de la taille préférée de ses composants internes.
- Lors de l'appel de la méthode **pack**, la méthode **getPreferredSize** est appelée sur tous les composants pour connaître leurs dimensions.
 - Ces informations sont utilisées pour calculer la dimension de la fenêtre.
- La méthode **setLocation** permet de positionner le composant à l'intérieur du conteneur.
- L'écran est le conteneur des **JFrame** et des **JWindow**

Les fenêtres

JWindow

- La fenêtre la plus basique.
- C'est juste un conteneur que vous pouvez afficher sur votre écran.
- Il n'a pas de :
 - barre de titre, boutons de fermeture / redimensionnement, et n'est pas redimensionnable par défaut.
- Vous pouvez bien sûr lui ajouter toutes ces fonctionnalités.
- On utilise surtout les JWindow pour faire des SplashScreen, c'est-à-dire des interfaces d'attente qui se ferment automatiquement.

```
import javax.swing.JWindow;

public class TestJWindow {
    public static void main(String[] args) {
        JWindow fenetre = new JWindow();
        fenetre.setSize(300, 300);
        fenetre.setLocation(300, 300);
        fenetre.setVisible(true);
    }
}
```



Les fenêtres

JFrame

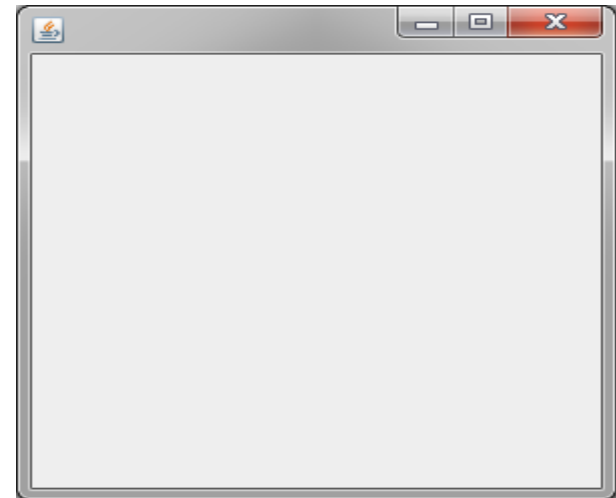
- C'est une fenêtre destinée à être la fenêtre principale de votre application.
- Elle n'est dépendante d'aucune autre fenêtre et ne peut pas être modale.
- La plupart des applications sont construites à partir d'une (ou plusieurs) **JFrame**.
 - ➔ En effet, **JFrame** construit des fenêtres qui comportent une bordure, un titre, des icônes et éventuellement un menu.
- La position des icônes et la police du titre dépend donc du système.
- Par contre, l'icône représentant la fenêtre lorsqu'elle est réduite peut être modifié en utilisant la méthode **setIconImage**.

Les fenêtres

JFrame

```
import javax.swing.JFrame;

public class TestJFrame {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame();
        fenetre.setSize(300, 300);
        fenetre.setVisible(true);
        fenetre.setLocation(500, 500);
    }
}
```

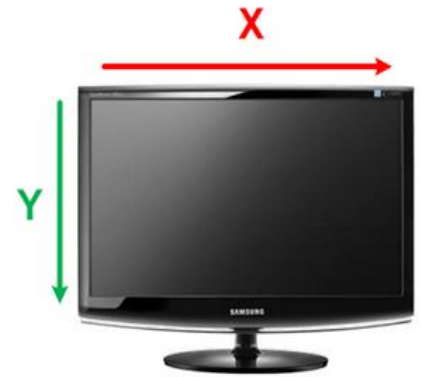


Exemple 1(Jframe) document Exemples-GUI

Les fenêtres

JFrame

- la méthode `setLocation(int x, int y)` : spécifier où doit se situer votre fenêtre sur l'écran. Les coordonnées, exprimées en pixels, sont basées sur un repère dont l'origine est représentée par le coin supérieur gauche.

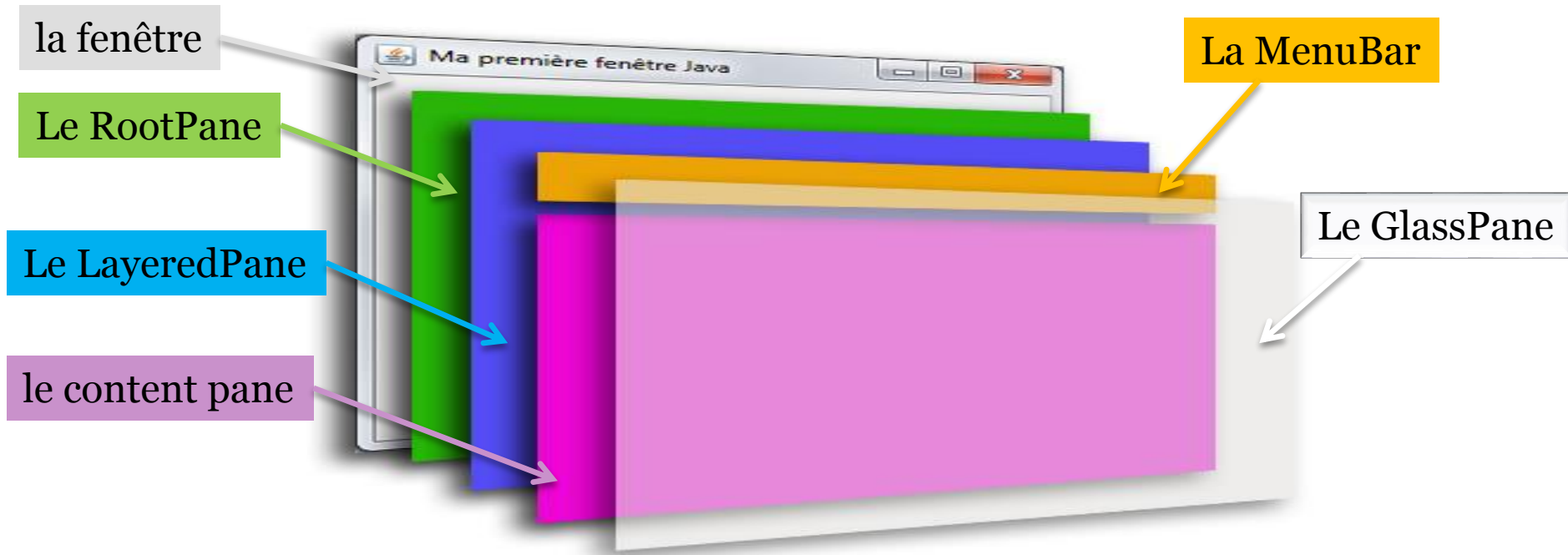


- La méthode `setResizable(boolean b) : false` empêche le redimensionnement tandis que `true` l'autorise.
- `setAlwaysOnTop(boolean b) : true` laissera la fenêtre au premier plan.

Les fenêtres

JFrame

Une JFrame est découpée en plusieurs parties superposées:



Conteneur principal qui contient les autres composants.

Forme juste un panneau composé du conteneur global et de la barre de menu .

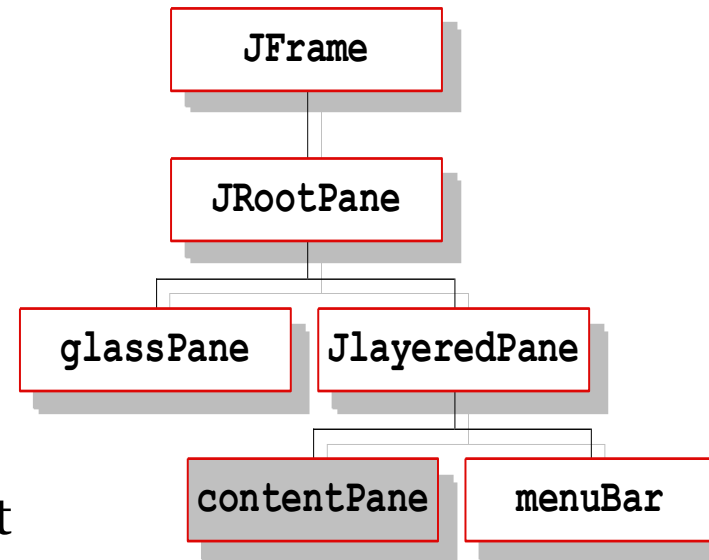
C'est dans celui-ci que nous placerons nos composants ;

Couche utilisée pour intercepter les actions de l'utilisateur avant qu'elles ne parviennent aux composants.

Les fenêtres

JFrame

- Une **JFrame** contient une fille unique, de la classe **JRootPane**
- Cette fille contient deux fils, **glassPane** (**JPanel**) et **layeredPane** (**JLayeredPane**)
- La layeredPane a deux fils, **contentPane** (un **Container**) et **menuBar** (un **JMenuBar**)
- On travaille dans **contentPane**.
- **JApplet**, **JWindow** et **JDialog** utilisent aussi un **JRootPane**.

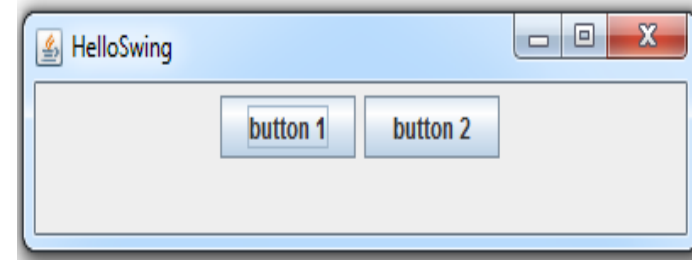


Les fenêtres

Utilisations du contentPane

JFrame

- Il est possible :
 - de demander le `contentPane` (`getContentPane()`)
 - de changer de `contentPane` (`setContentPane()`)



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

public class HierarchySwing {

    public static void main(String[] args) {

        JFrame frame=new JFrame("HelloSwing");
        Container c=frame.getContentPane();
        c.setLayout(new FlowLayout());

        JButton button1=new JButton("button 1");
        c.add(button1);
        JButton button2=new JButton("button 2");
        c.add(button2);

        frame.setSize(400,100);
        frame.setVisible(true);
    }
}
```

```
import javax.swing.*;

public class HierarchySwing2 {

    public static void main(String[] args) {

        JButton button1 = new JButton("button 1");
        JButton button2 = new JButton("button 2");

        JPanel panel = new JPanel();
        panel.add(button1);
        panel.add(button2);

        JFrame frame = new JFrame("HelloSwing");
        frame.setContentPane(panel);

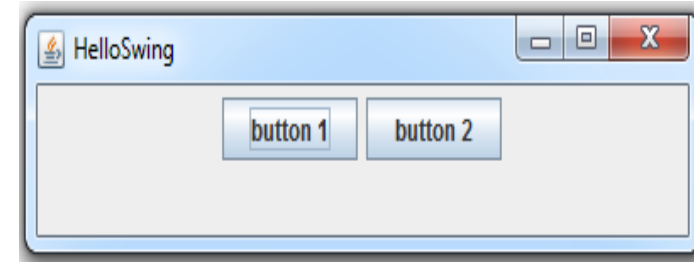
        frame.setSize(400, 100);
        frame.setVisible(true);
    }
}
```

Les fenêtres

Utilisations du `ContentPane`

`JFrame`

- Normalement, l'ajout de composants se fait sur le `ContentPane` et non sur la `JFrame`
- Mais, si l'on effectue un `add()` sur une `JFrame` :
 - En 1.4 et avant, il y a une erreur à l'exécution
 - En 1.5, est équivalent à `getContentPane().add()`
- Ce mécanisme marche pour les méthodes :
`add/remove/setLayout`



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

public class HierarchySwing3 {
    public static void main(String[] args) {

        JFrame frame=new JFrame("HelloSwing");
        frame.setLayout(new FlowLayout());

        JButton button1=new JButton("button 1");
        frame.add(button1);
        JButton button2=new JButton("button 2");
        frame.add(button2);

        frame.setSize(400,100);
        frame.setVisible(true);
    }
}
```

Les fenêtres

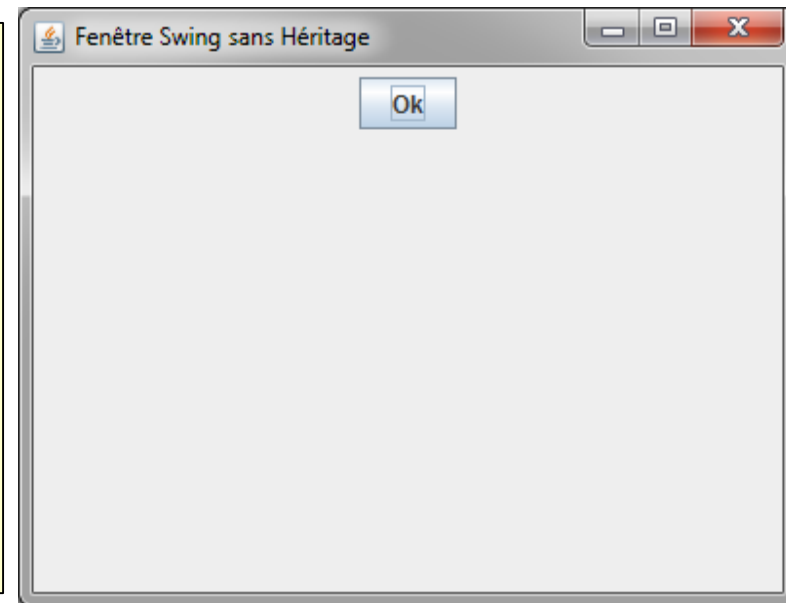
Interface graphique et programmation objet

JFrame

- Java est un langage Objet, il est donc possible d'utiliser l'héritage.
- Doit-on hériter par exemple de la classe JFrame ?

```
import javax.swing.*;

public class MorphSwing1 {
    public static void main(String[] args) {
        JButton button=new JButton("Ok");
        JPanel panel=new JPanel();
        panel.add(button);
        JFrame frame=new JFrame("Fenêtre Swing sans Héritage");
        frame.setContentPane(panel);
        frame.setSize(400,300);
        frame.setVisible(true);
    }
}
```



Les fenêtres

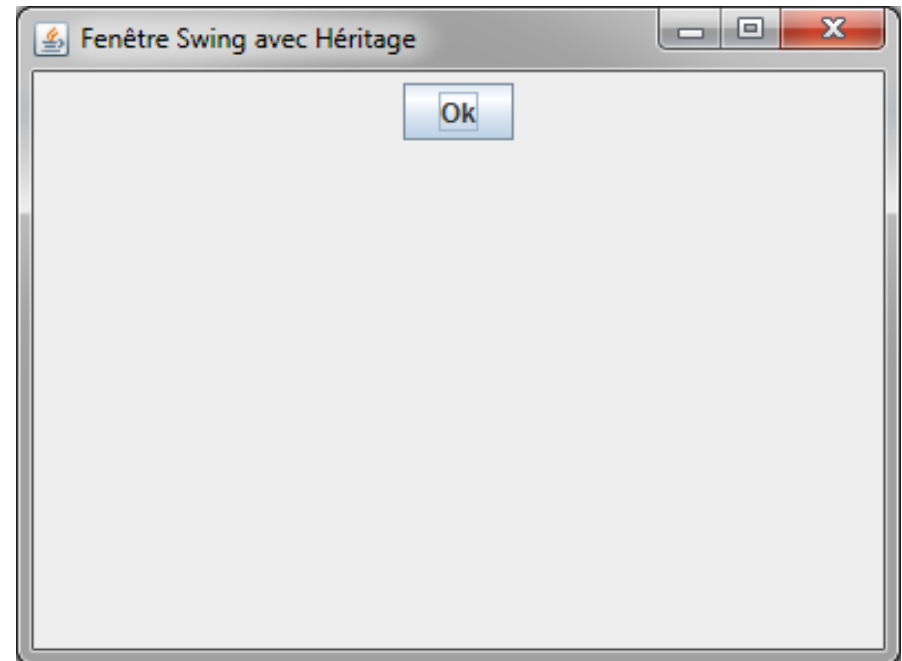
Interface graphique et programmation objet

JFrame

- Java est un langage Objet, il est donc possible d'utiliser l'héritage.
- Doit-on hériter par exemple de la classe JFrame ?
- **OUI**

```
import javax.swing.*;

public class MorphSwing2 extends JFrame {
    public MorphSwing2() {
        super("Fenêtre Swing avec Héritage");
        JButton button=new JButton("Ok");
        JPanel panel=new JPanel();
        panel.add(button);
        setContentPane(panel);
        setSize(400,300);
        setVisible(true);
    }
    public static void main(String[] args) {
        new MorphSwing2();
    }
}
```



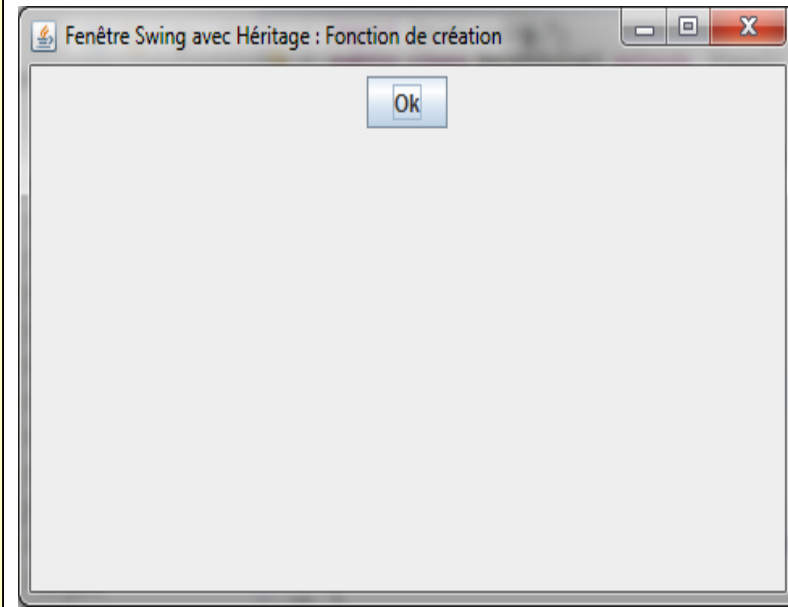
Les fenêtres

Interface graphique et programmation objet

JFrame

- On hérite d'une classe si on veut en changer les fonctionnalités (i.e. redéfinir une méthode)
 - Il est possible de faire des fonctions pour rendre le code plus clair.

```
import javax.swing.*;
public class MorphSwing3 extends JPanel{
    public MorphSwing3() {
        super();
        JButton button=new JButton("Ok");
        this.add(button);
    }
    JFrame créer(String title) {
        JFrame frame = new JFrame(title);
        frame.setContentPane(this);
        frame.setSize(400, 300);
        frame.setVisible(true);
        return frame; }
    public static void main(String[] args) {
        new MorphSwing3().créer("Fenêtre Swing avec
        Héritage : Fonction de création ");
    }
}
```

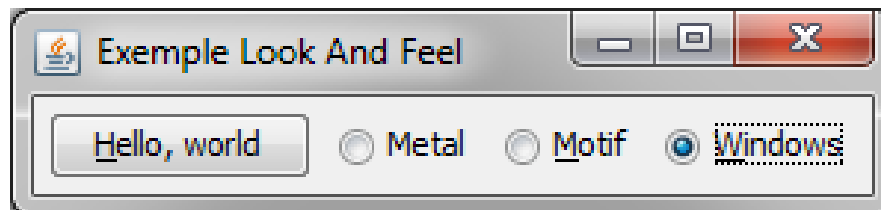
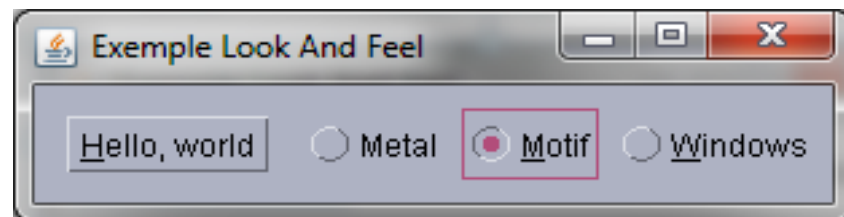
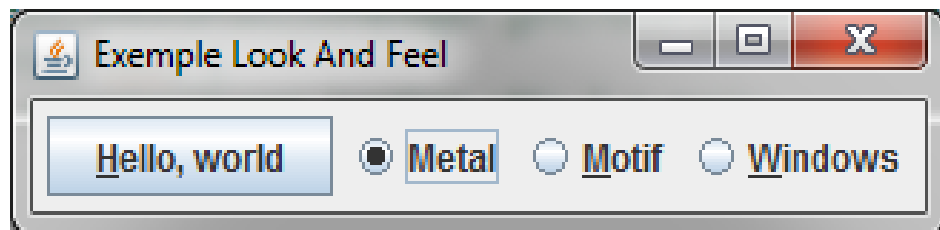


Les fenêtres

Pluggable Look & Feel

JFrame

- Swing sépare les composants de leur rendu.
- Il est ainsi possible de changer le look d'un ensemble de composants.
 - Windows : `com.sun.java.swing.plaf.windows.WindowsLookAndFeel`
 - Motif : `com.sun.java.swing.plaf.motif.MotifLookAndFeel`
 - Metal : `javax.swing.plaf.metal.MetalLookAndFeel`
 - GTK : `com.sun.java.swing.plaf.gtk.GTKLookAndFeel`



Pluggable Look & Feel

JFrame

- Swing sépare les composants de leur rendu.
- Il est ainsi possible de changer le look d'un ensemble de composants.
 - Windows : `com.sun.java.swing.plaf.windows.WindowsLookAndFeel`
 - Motif : `com.sun.java.swing.plaf.motif.MotifLookAndFeel`
 - Metal : `javax.swing.plaf.metal.MetalLookAndFeel`
 - GTK : `com.sun.java.swing.plaf.gtk.GTKLookAndFeel`

```
try {  
    UIManager.setLookAndFeel(className);  
} catch (UnsupportedLookAndFeelException e) {  
    ...  
} catch (Exception e) {  
    ...  
}  
SwingUtilities.updateComponentTreeUI(frame);  
frame.pack();
```

```
try {  
    UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");  
} catch (Exception e) { e.printStackTrace(); }
```

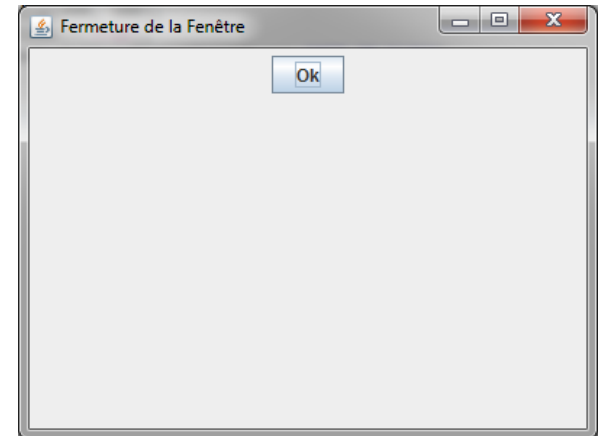
Les fenêtres

Fermeture de fenêtre

JFrame

- Par défaut, en cliquant sur la croix d'une fenêtre, l'application n'est pas arrêtée.
- `setDefaultCloseOperation()` permet de spécifier un comportement
- Comportements :
 - `DO_NOTHING_ON_CLOSE`
 - `HIDE_ON_CLOSE` (défaut)
 - `DISPOSE_ON_CLOSE`
 - `EXIT_ON_CLOSE`

```
import javax.swing.*;  
public class FermerFrame extends JFrame {  
    public static void main(String[] args) {  
        JButton button = new JButton("Ok");  
        JPanel panel = new JPanel();  
        panel.add(button);  
        JFrame frame = new JFrame("Fermeture de la Fenêtre");  
        frame.setContentPane(panel);  
        frame.setSize(400, 300);  
        frame.setVisible(true);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

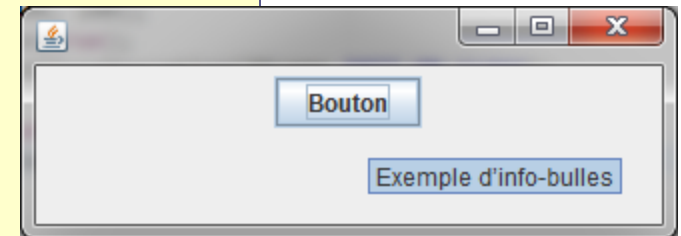


Les info bulles

JToolTip

- Tous les composants dérivant de **JComponent** peuvent être agrémentés d'info-bulles :
 - ➔ petites boîtes contenant un texte qui présente le rôle du contrôle.
- En pratique, il n'est pas nécessaire d'instancier un objet.
 - ➔ La classe **JComponent** propose une méthode qui permet de créer directement le composant **JToolTip** et de l'associer au composant.

```
import javax.swing.*;  
public class ExempleJToolTip extends JFrame {  
    public ExempleJToolTip() {  
        JButton button = new JButton("Bouton");  
        button.setToolTipText("Exemple d'info-bulles");  
        JPanel panel=new JPanel();  
        panel.add(button);  
        getContentPane().add(panel);  
        setSize(300, 100);  
        setVisible(true);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); }  
    public static void main(String args[]) {  
        new ExempleJToolTip();}}}
```

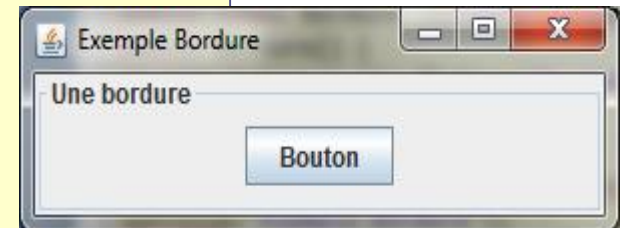


Les bordures

Bordure

- Les composants peuvent être encadrés par une bordure souvent utilisée pour matérialiser des blocs d'informations
- La classe **BorderFactory** fournit de nombreuses méthodes statiques pour créer des bordures prédéfinies.

```
import javax.swing.*;
public class Bordure extends JFrame{
public Bordure() {
    JButton button = new JButton("Bouton");
    JPanel panel=new JPanel();
    panel.add(button);
    panel.setBorder(BorderFactory.createTitledBorder("Une bordure"));
    setTitle("Exemple Bordure");
    setSize(300, 100);
    setVisible(true);
    setContentPane(panel);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
    public static void main(String[] args) {
        new Bordure();
    }
}
```



Les boites de dialogues

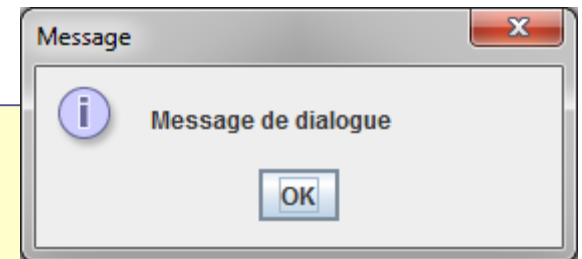
- Swing propose des boites de dialogues afin de communiquer rapidement avec l'utilisateur.
- La classe **JOptionPane** permet de créer des boites de dialogues génériques mais aussi modifiable en fonction des besoins de l'application.
- Les méthodes d'affichage sont statiques
 - ➔ il n'est donc pas nécessaire d'instancier une classe pour les utiliser.

Les boites de dialogues

Les messages de dialogue

- Informent l'utilisateur en affichant un texte simple et un bouton de confirmation.
- On peut aussi afficher un icône correspondant au message (point d'exclamation, symbole d'erreur,...).
- Pour afficher un message on utilise la méthode `showMessageDialog` de la classe `JOptionPane`.

```
import javax.swing.JFrame;  
import javax.swing.JOptionPane;  
  
public class MsgDialogue1 {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame();  
        JOptionPane.showMessageDialog(frame, "Message de dialogue");  
        System.exit(0);  
    }  
}
```

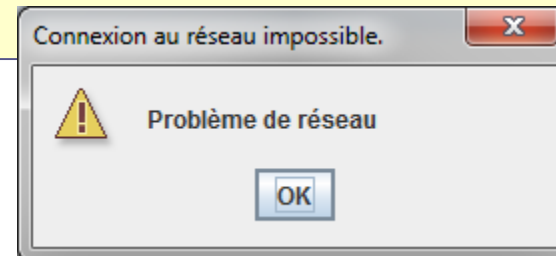


Les boîtes de dialogues

Les messages de dialogue

- On peut modifier l'aspect de la fenêtre en fonction du type de message.
- **Exemple** : Afficher un dialogue d'avertissement .

```
import javax.swing.JFrame;  
import javax.swing.JOptionPane;  
  
public class MsgDialogue2 {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame();  
        JOptionPane.showMessageDialog(frame, "Problème de réseau",  
            "Connexion au réseau impossible.", JOptionPane.WARNING_MESSAGE);  
        System.exit(0);  
    }  
}
```



Les boites de dialogues

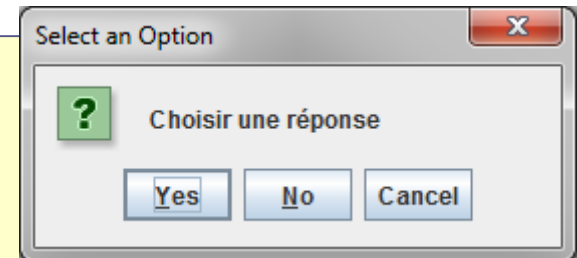
Les messages de dialogue

Les dialogues de confirmation/question

- Les boites de dialogues peuvent aussi être utilisées pour demander un renseignement à l'utilisateur.
 - Le cas le plus courant est une question fermée.
- La méthode `showConfirmDialog` affiche ce type de boite.

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class MsgDialogue3 {
    public static void main(String [] args) {
        JFrame frame=new JFrame();
        JOptionPane.showConfirmDialog(frame, "Choisir une réponse");
        System.exit(0);
    }
}
```



Les boites de dialogues

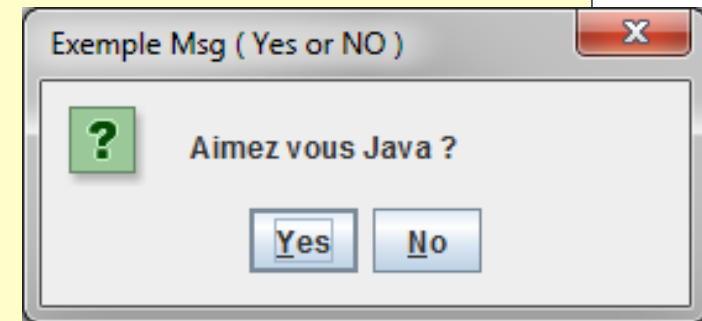
Les messages de dialogue

Les dialogues de confirmation/question

- La boîte de dialogue renvoie un entier en fonction du choix de l'utilisateur et peut être utilisé pour modifier le comportement du programme.
- Cet entier est défini comme une constante de la classe `JOptionPane`.

```
import javax.swing.*;

public class MsgDialogue4 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        int rep = JOptionPane.showConfirmDialog(frame, " Aimez vous Java ? ",
        " Exemple Msg (Yes or NO) ", JOptionPane.YES_NO_OPTION);
        if (rep == JOptionPane.YES_OPTION) {
            // ...
        }
        if (rep == JOptionPane.NO_OPTION) {
            // ...
        }
        System.exit(0);
    }
}
```



Les boites de dialogues

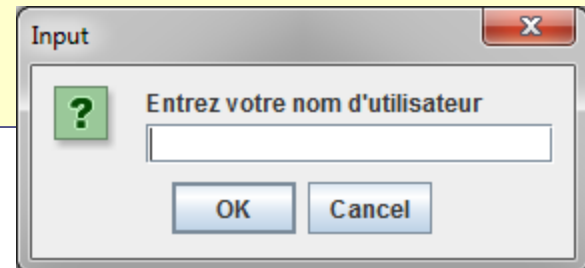
Les messages de dialogue

Les dialogues de saisie

- La méthode `showInputDialog` affiche une boite de dialogue comprenant une entrée de saisie (`JTextField`) en plus des boutons de validation.
- Après validation elle retourne une chaîne de caractères (`String`) si l'utilisateur a cliqué sur OK, sinon elle renvoie `null`.

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class MsgDialogue5 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        String rep = JOptionPane.showInputDialog(frame, "Entrez votre nom d'utilisateur");
        System.exit(0);
    }
}
```



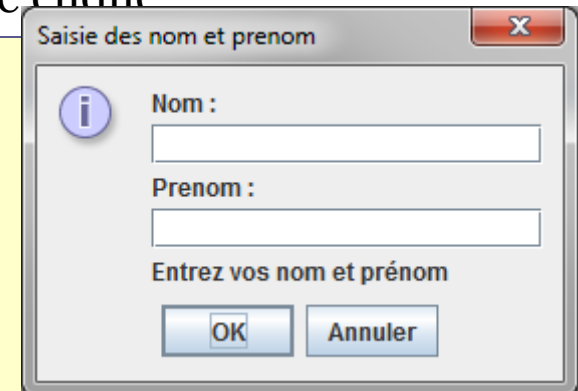
Les boites de dialogues

Les messages de dialogue

Construction de dialogues personnalisés

- Pour construire des boites de dialogue plus complexes on utilise la méthode `showOptionDialog`.
- Cette méthode admet 8 paramètres (certains pouvant être à `null`).
- Elle renvoie un entier qui correspond au bouton qui a été cliqué

```
import javax.swing.*;
public class MsgDialogue6{
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        JLabel labelNom = new JLabel("Nom :");
        JLabel labelPrenom = new JLabel("Prenom :");
        JTextField nom = new JTextField();
        JTextField prenom = new JTextField();
        JLabel lab = new JLabel("Entrez vos nom et prénom ");
        Object[] tab = new Object[] { labelNom, nom, labelPrenom, prenom, lab };
        int rep = JOptionPane.showOptionDialog(frame, tab, "Saisie des nom et prenom ",
        JOptionPane.OK_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE, null, null, null);
        System.exit(0);
    }
}
```



Les conteneurs secondaires

- Swing propose de nombreux conteneurs secondaires pour créer des interfaces ergonomiques.
→ des composants légers.
- Il est possible d'en créer d'autre à l'aide des méthodes de programmation orientée objets.
- La plupart des interfaces possibles dans les systèmes d'exploitation usuels sont présentes comme par exemple les panneaux à onglets, les panneaux défilant,...

Les conteneurs secondaires

Le panneau : JPanel

- Le conteneur léger le plus simple de Swing et le panneau (**JPanel**), c'est le conteneur par défaut de **JFrame** et de **JWindow**.
- Permet de grouper des composants selon une politique de placement.
- Pour ajouter un composant à un panneau, on utilise la méthode **add** (ou l'une de ses surcharges).
- La méthode réciproque **remove** permet d'enlever un composant.

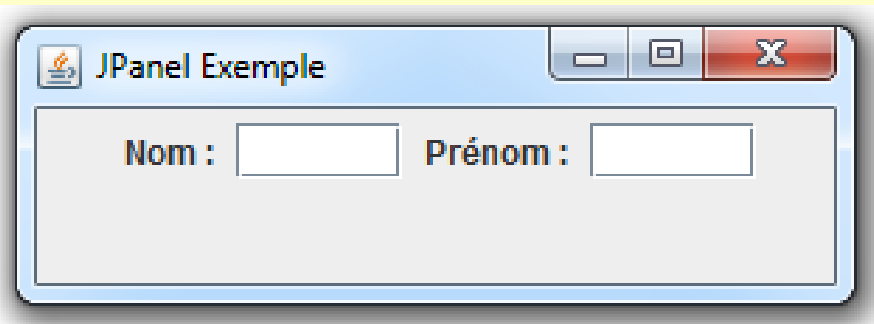
Les conteneurs secondaires

Le panneau : JPanel

```
import javax.swing.*;
public class FicheIdentite extends JPanel {
    private JTextField nom;
    private JLabel labelNom;
    private JTextField prenom;
    private JLabel labelPrenom;

    public FicheIdentite() {
        super();
        labelNom = new JLabel(" Nom : ");
        labelPrenom = new JLabel(" Prénom : ");
        nom = new JTextField(5);
        prenom = new JTextField(5);
        this.add(labelNom);
        this.add(nom);
        this.add(labelPrenom);
        this.add(prenom);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("JPanel Exemple");
        frame.setSize(300, 100);
        frame.setContentPane(new FicheIdentite());
        frame.setVisible(true);
    }
}
```



Les conteneurs secondaires

Le panneau a défilement : JScrollPane

- Les applications traitant du texte ou des images n'affichent souvent qu'une partie du document pour éviter de monopoliser toute la surface d'affichage.
 - Des ascenseurs sont affichées sur les cotés afin de pouvoir se déplacer dans le document.
 - Le composant **JScrollPane** permet d'implémenter cette fonction.

```
import javax.swing.*;
public class TestScrollPane {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        JLabel monImage = new JLabel(new ImageIcon("Koala.jpg"));
        JScrollPane lePanneau = new JScrollPane(monImage);
        frame.setContentPane(lePanneau);
        frame.setTitle("Image de Koala");
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.setVisible(true);
    }
}
```

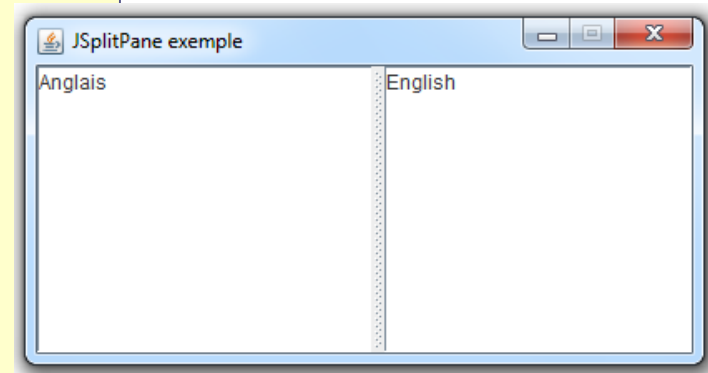


Les conteneurs secondaires

Le panneau divise : JSplitPane

- Séparer une interface en deux volets est utilisé pour mettre en vis-à-vis deux documents, ou un document et une barre d'outils.
- La séparation peut être horizontale ou verticale selon les interfaces.
- Ce type d'interface fait appel au **JSplitPane**.

```
import javax.swing.*;
public class TestSplitPane {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JSplitPane exemple");
        JTextArea source = new JTextArea();
        JTextArea traduction = new JTextArea();
        JSplitPane lePanneau = new
        JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
        source, traduction);
        frame.setContentPane(lePanneau);
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.setVisible(true);
    }
}
```

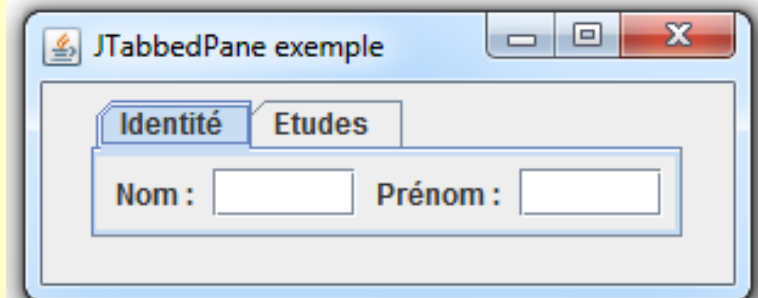


Les conteneurs secondaires

Le panneau a onglets : JTabbedPane

- Le composant **JTabbedPane** permet de construire des interfaces en utilisant des onglets.
- Les composants peuvent ainsi être regroupés de manière thématique pour obtenir des interfaces allégées.
- La méthode **addTab** permet d'ajouter un composant dans une nouvelle feuille.
- Généralement on utilise le plus souvent **JPanel** comme conteneur.

```
public class MenuOnglets extends JTabbedPane {
    private FicheIdentite identite;
    private FicheIdentite etudes;
    public MenuOnglets() {
        identite = new FicheIdentite();
        etudes = new FicheIdentite();
        this.addTab("Identité", identite);
        this.addTab("Etudes ", etudes);
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTabbedPane exemple");
        JPanel panel = new JPanel();
        panel.add(new MenuOnglets());
        frame.setContentPane(panel);
        frame.setSize(300, 120);
        frame.setVisible(true);
    }
}
```

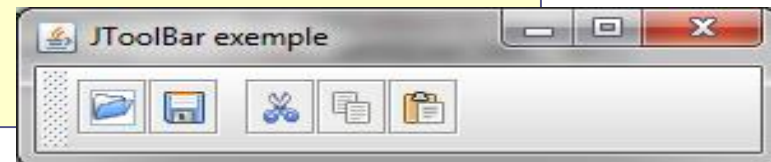


Les conteneurs secondaires

Les barres d'outils : JToolBar

- Les applications complexes font souvent appel à des barres d'outils pour pouvoir accéder facilement aux fonctions les plus utilisées.
- Swing propose un conteneur (**JToolBar**) pour construire des barres d'outils regroupant n'importe quel composant.

```
import javax.swing.*;
public class JToolBarExemple {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTabbedPane exemple");
        JButton bOuvrir = new JButton(new ImageIcon("openfile.png"));
        JButton bEnregistrer = new JButton(new ImageIcon("savefile.png"));
        JButton bCouper = new JButton(new ImageIcon("editcut.png"));
        JButton bCopier = new JButton(new ImageIcon("editcopy.png"));
        JButton bColler = new JButton(new ImageIcon("editpaste.png"));
        bOuvrir.setToolTipText("Ouvrir un fichier");
        bEnregistrer.setToolTipText("Enregistrer le fichier");
        bCouper.setToolTipText("Couper vers le presse - papier");
        bCopier.setToolTipText("Copier vers les presse - papier");
        bColler.setToolTipText("Coller depuis le presse - papier");
        JToolBar tb = new JToolBar(); tb.add(bOuvrir); tb.add(bEnregistrer);
        tb.addSeparator(); tb.add(bCouper); tb.add(bCopier); tb.add(bColler);
        frame.setContentPane(tb); frame.setSize(300, 80);
        frame.setVisible(true);
    }
}
```



Les conteneurs secondaires

Les bureaux JDesktopPane

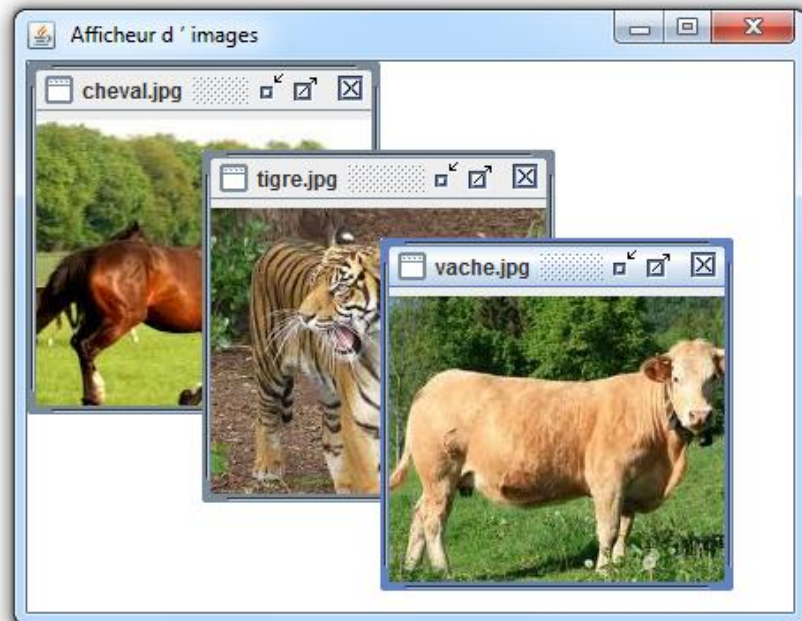
- De nombreuses applications autorisent l'ouverture de plusieurs documents simultanément.
 - Ces interfaces sont nommées MDI (Multiple Document Interface).
- Le composant **JDesktopPane** propose une implémentation de ce comportement.
 - permet d'afficher des fenêtres (**JInternalFrame**) à l'intérieur de l'application.
- Les fenêtres **JInternalFrame** proposent des méthodes proches de celles de **JFrame** (bien qu'il n'existe pas de relation d'héritage).
- Pour des application MDI, une classe fille est souvent créée à partir de **JInternalFrame**.

Les conteneurs secondaires

Les bureaux JDesktopPane

```
import javax.swing.*;

public class FrameMDI extends JInternalFrame {
    public FrameMDI() {
        super("", true, true, true, true);
    }
    public void setImage(String nomImage) {
        JPanel unPanneau = new JPanel();
        JLabel uneImage = new JLabel(new ImageIcon(nomImage));
        unPanneau.add(uneImage);
        this.setContentPane(unPanneau);
        this.setTitle(nomImage);
    }
}
```



Les conteneurs secondaires

Les bureaux JDesktopPane

```
import javax.swing.*;
public class TestDesktop extends JDesktopPane {
    FrameMDI[] animaux;
    String[] nomsFichier = { "cheval.jpg", "tigre.jpg", "vache.jpg" };
    public TestDesktop() {
        animaux = new FrameMDI[3];
        for (int i = 0; i < 3; i++) {
            animaux[i] = new FrameMDI();
            animaux[i].setSize(200, 200);
            animaux[i].setLocation(100 * i, 50 * i);
            animaux[i].setImage(nomsFichier[i]);
            animaux[i].setVisible(true);
            this.add(animaux[i]);
        }
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setContentPane(new TestDesktop());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.setSize(450, 350);
        frame.setTitle("Afficheur d'images");
    }
}
```


Les composants atomiques

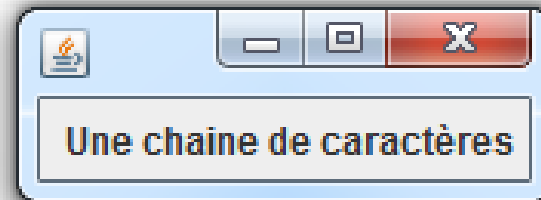
- Les composants atomiques sont tous les composants élémentaire de Swing.
- Ce sont les boutons, les labels, les menus, . . .

Les composants atomiques

Les labels : JLabel

- Un label est une simple chaîne de caractères informative (il peut aussi contenir une image).
- Pour créer un nouveau label il suffit d'appeler le constructeur **JLabel**.

```
import javax.swing.*;  
public class TestJLabel {  
    public static void main(String[] args) {  
        JFrame fen = new JFrame();  
        JPanel pan = new JPanel();  
        JLabel unLabel = new JLabel("Une chaîne de caractères");  
        pan.add(unLabel);  
        fen.setContentPane(pan);  
        fen.pack();  
        fen.setVisible(true);  
    }  
}
```



Les composants atomiques

Les labels : JLabel

- **JLabel** peut afficher aussi un objet **ImageIcon**.
 - Il suffit alors de créer un objet **ImageIcon** à partir de l'image à afficher et d'associer cet icône à un **JLabel**.
- Le constructeur de **ImageIcon** permet de charger directement un fichier de type JPEG, GIF ou PNG en passant le nom du fichier en paramètre lors de l'appel du construc

```
import javax.swing.*;  
public class TestJLabelImage {  
    public static void main(String[] args) {  
        JFrame fen = new JFrame();  
        JPanel pan = new JPanel();  
        ImageIcon img = new ImageIcon("cheval.jpg");  
        JLabel unLabel = new JLabel ( img );  
        pan . add ( unLabel );  
        fen.setContentPane(pan);  
        fen.pack();  
        fen.setVisible(true);  
    }  
}
```



Les composants atomiques

Les boutons : JButton et JToggleButton

- Les boutons sont les composants les plus utilisés pour interagir avec l'utilisateur.
- Swing propose plusieurs types de boutons.
- Tous les boutons héritent de la classe abstraite **AbstractButton** qui fournit de nombreuses méthodes pour paramétrer les boutons.
- Tous les boutons peuvent être accompagnés d'un texte (**setText**).
- Les descendants de **AbstractButton** peuvent être décorés à l'aide d'icônes (**setIcon**).
- La méthode **setMnemonic** permet de définir une touche de raccourcis pour le clavier (combinaisons ALT + touche).

```
monBouton.setText("Un bouton");  
monBouton.setMnemonic(KeyEvent.VK_B);
```

Les composants atomiques

Les boutons : JButton

- Le bouton le plus utilisé est le **JButton**.
- Crée un bouton qui peut être cliqué par l'utilisateur à l'aide de la souris.

```
import javax.swing.*;
public class TestJButton {
    public static void main(String[] args) {
        JFrame fen = new JFrame();
        JPanel pan = new JPanel();
        JLabel unLabel = new JLabel("Un label");
        JButton unBouton1 = new JButton("Un Bouton 1");
        ImageIcon img = new ImageIcon("openfile.png");
        JButton unBouton2 = new JButton("Un Bouton 2", img);
        pan.add(unLabel);
        pan.add(unBouton1);
        pan.add(unBouton2);
        fen.setContentPane(pan);
        fen.pack();
        fen.setVisible(true);
    }
}
```

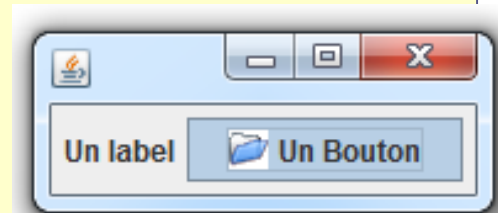
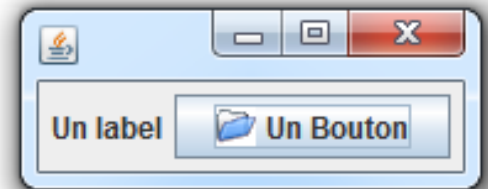


Les composants atomiques

Le composant JToggleButton

- Swing propose un type de bouton particulier, les boutons à bascule : **JToggleButton**.
- Ces boutons conservent leur état après le relâchement de la souris.

```
import javax.swing.*;
public class TestJToggleButton {
    public static void main(String[] args) {
        JFrame fen = new JFrame();
        JPanel pan = new JPanel();
        JLabel unLabel = new JLabel("Un label");
        ImageIcon img = new ImageIcon("openfile.png");
        JToggleButton unBouton = new JToggleButton("Un Bouton", img);
        pan.add(unLabel);
        pan.add(unBouton);
        fen.setContentPane(pan);
        fen.pack();
        fen.setVisible(true);
    }
}
```



Les composants atomiques

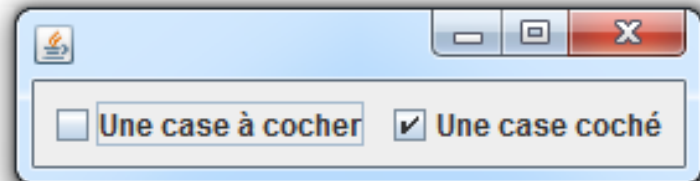
Les cases à cocher : JCheckBox

- Les cases à cocher permettent de matérialiser des choix binaires d'une manière plus usuelle que les boutons à bascules (**JToggleButton**.) Comme les boutons, elles héritent de la classe abstraite **AbstractButton**.

```
import javax.swing.*;

public class TestJCheckBox {

    public static void main(String[] args) {
        JFrame fen = new JFrame();
        JPanel pan = new JPanel();
        JCheckBox casePasCochee = new JCheckBox( "Une case à cocher" ) ;
        JCheckBox caseCochee = new JCheckBox("Une case coché" , true );
        pan.add(casePasCochee);
        pan.add(caseCochee);
        fen.setContentPane(pan);
        fen.pack();
        fen.setVisible(true);
    }
}
```

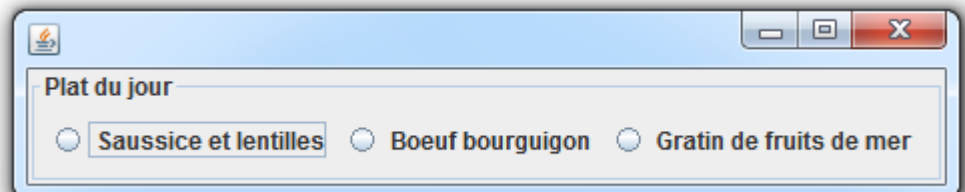


Les composants atomiques

Les boutons radio : JRadioButton

- Les boutons radio **JRadioButton** sont des boutons à choix exclusif, ils permettent de choisir un (et un seul) élément parmi un ensemble.

```
import javax.swing.*;
public class TestJRadioButton {
    public static void main(String[] args) {
        JFrame fen = new JFrame();
        JPanel pan = new JPanel();
        JRadioButton plat1, plat2, plat3;
        ButtonGroup plat;
        plat1 = new JRadioButton(" Saussice et lentilles");
        plat2 = new JRadioButton(" Boeuf bourguignon");
        plat3 = new JRadioButton(" Gratin de fruits de mer ");
        plat = new ButtonGroup();
        plat.add(plat1);    plat.add(plat2);
        plat.add(plat3);    pan.add(plat1);
        pan.add(plat2);
        pan.add(plat3);
        pan.setBorder(BorderFactory.createTitledBorder("Plat du jour"));
        fen.setContentPane(pan);
        fen.pack();
        fen.setVisible(true);
    }
}
```



Les composants atomiques

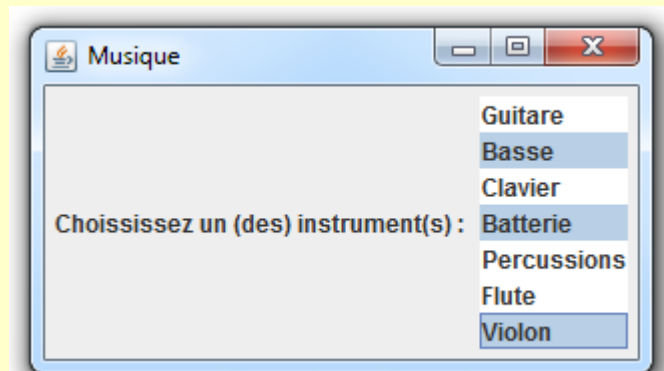
Les listes de choix : JList

- **JList** permet de choisir un (ou plusieurs) élément(s) parmi un ensemble prédéfini.
- Cet ensemble peut être un tableau ou vecteur d'objets quelconques.

```
import javax.swing.*;
public class TestJList {

    public static void main(String[] args) {
        JFrame fen = new JFrame("Musique");
        JPanel pan = new JPanel();
        String[] lesElements = { "Guitare", "Basse", "Clavier", "Batterie",
                                "Percussions", "Flute", "Violon" };
        JList<String> instruments = new JList<String>(lesElements);
        JLabel text = new JLabel("Choisissez un (des) instrument(s) : ");
        pan.add(text);
        pan.add(instruments);

        fen.setContentPane(pan);
        fen.pack();
        fen.setVisible(true);
    }
}
```

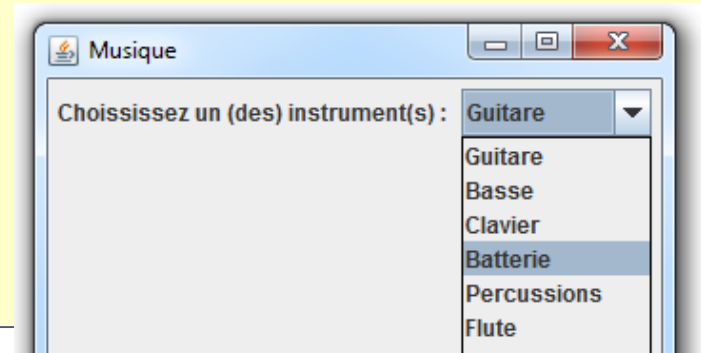


Les composants atomiques

Les boites combo : JComboBox

- Les boites combo permettent de choisir un seul élément parmi une liste proposée.
- Elles ont un comportement proche des boutons radio.
- On les utilise quand l'ensemble des éléments à afficher n'est pas connu lors de la conception.

```
import javax.swing.*;
public class TestJComboBox {
    public static void main(String[] args) {
        JFrame fen = new JFrame("Musique");
        JPanel pan = new JPanel();
        String[] lesElements = { "Guitare", "Basse", "Clavier", "Batterie",
                                "Percussions", "Flute", "Violon" };
        JComboBox<String> instruments = new JComboBox<String>(lesElements);
        JLabel text = new JLabel("Choisissez un (des) instrument(s) : ");
        pan.add(text);
        pan.add(instruments);
        fen.setContentPane(pan);
        fen.pack();
        fen.setVisible(true);
    }
}
```

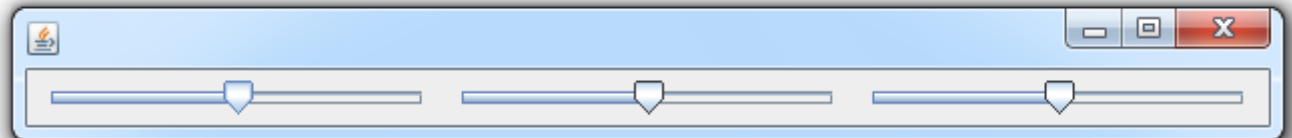


Les composants atomiques

Les glissieres : JSlider

- Les glissières permettent de proposer à l'utilisateur une interface de saisie plus intuitive qu'un champ de texte pour régler certains paramètres.
- Swing propose le composant **JSlider** pour représenter un réglage variable.

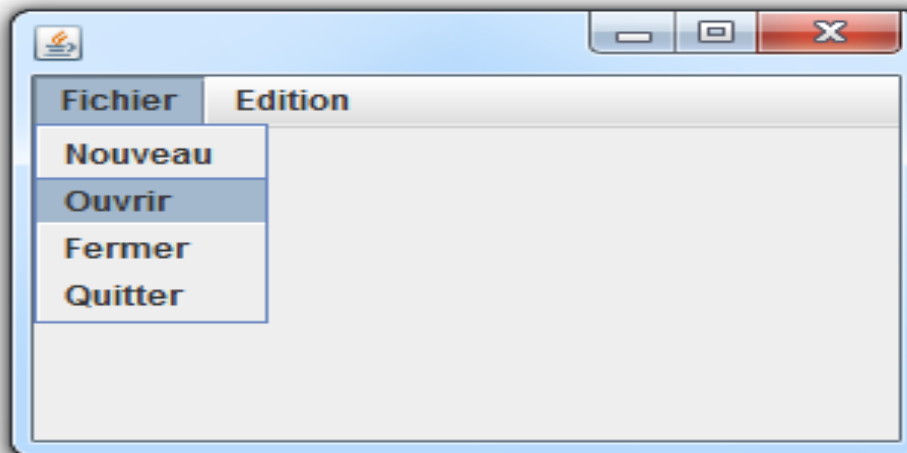
```
import javax.swing.*;  
public class TestJSlider {  
    public static void main(String[] args) {  
        JFrame fen = new JFrame();  
        JPanel pan = new JPanel();  
        JSlider sBlue = new JSlider();  
        JSlider sRed = new JSlider();  
        JSlider sGreen = new JSlider();  
        pan.add(sRed);    pan.add(sGreen);    pan.add(sBlue);  
        fen.setContentPane(pan);  
        fen.pack();  
        fen.setVisible(true);  
    }  
}
```



Les composants atomiques

Les menus

- Les barres de menus (**JMenuBar**) permettent de regrouper de nombreuses fonctions d'une manière ergonomique et hiérarchique.
- Les menus sont construits à partir de la classe **JMenu** et sont placés dans la **JMenuBar** (méthode **add**).
- Ces menus sont constitués d'éléments appartenant à la classe **JMenuItem** ou à l'une de ses classes filles (**JRadioButtonMenuItem**, **JCheckBoxMenuItem**).
- La classe **JMenuItem** est une classe héritant de **JToggleButton**.
- Les éléments de menus sont ajoutés aux menus (méthode **add**).



Les composants atomiques

Les menus

```
import javax.swing.*;
public class TestJMenu {
    public static void main(String[] args) {
        JFrame fen = new JFrame();
        JMenu menuFichier = new JMenu(" Fichier ");
        JMenuItem menuFichierNouveau = new JMenuItem(" Nouveau ");
        JMenuItem menuFichierOuvrir = new JMenuItem(" Ouvrir ");
        JMenuItem menuFichierFermer = new JMenuItem(" Fermer ");
        JMenuItem menuFichierQuitter = new JMenuItem(" Quitter ");
        menuFichier.add(menuFichierNouveau);          menuFichier.add(menuFichierOuvrir);
        menuFichier.add(menuFichierFermer);
        menuFichier.add(menuFichierQuitter);
        JMenu menuEdition = new JMenu(" Edition ");
        JMenuItem menuEditionCouper = new JMenuItem(" Couper ");
        JMenuItem menuEditionCopier = new JMenuItem(" Copier ");
        JMenuItem menuEditionColler = new JMenuItem(" Coller ");
        menuEdition.add(menuEditionCouper);            menuEdition.add(menuEditionCopier);
        menuEdition.add(menuEditionColler);
        JMenuBar barreMenu = new JMenuBar();
        barreMenu.add(menuFichier);                    barreMenu.add(menuEdition);
        fen.setJMenuBar(barreMenu);
        fen.setSize(300, 200);                        fen.setVisible(true);
    }
}
```

Les composants atomiques

Les menus

```
import javax.swing.*;
public class TestJMenu {
    public static void main(String[] args) {
        JFrame fen = new JFrame();
        JMenu menuFichier = new JMenu(" Fichier ");
        JMenuItem menuFichierNouveau = new JMenuItem(" Nouveau ");
        JMenuItem menuFichierOuvrir = new JMenuItem(" Ouvrir ");
        JMenuItem menuFichierFermer = new JMenuItem(" Fermer ");
        JMenuItem menuFichierQuitter = new JMenuItem(" Quitter ");
        menuFichier.add(menuFichierNouveau);          menuFichier.add(menuFichierOuvrir);
        menuFichier.add(menuFichierFermer);
        menuFichier.add(menuFichierQuitter);
        JMenu menuEdition = new JMenu(" Edition ");
        JMenuItem menuEditionCouper = new JMenuItem(" Couper ");
        JMenuItem menuEditionCopier = new JMenuItem(" Copier ");
        JMenuItem menuEditionColler = new JMenuItem(" Coller ");
        menuEdition.add(menuEditionCouper);            menuEdition.add(menuEditionCopier);
        menuEdition.add(menuEditionColler);
        JMenuBar barreMenu = new JMenuBar();
        barreMenu.add(menuFichier);                    barreMenu.add(menuEdition);
        fen.setJMenuBar(barreMenu);
        fen.setSize(300, 200);                        fen.setVisible(true);
    }
}
```

Les composants atomiques

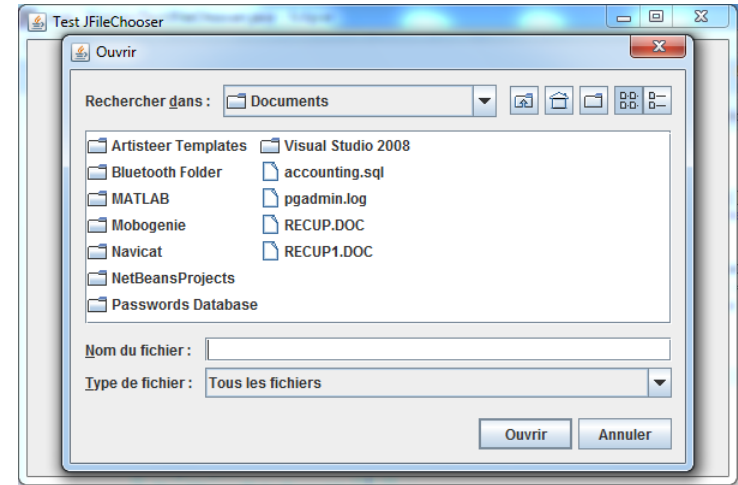
Les dialogues de sélection de fichiers : JFileChooser

- Swing propose une boîte de sélection de fichier(s) avec la classe **JFileChooser**.
- La classe propose trois méthodes pour afficher un dialogue d'ouverture de fichier.
 - **showOpenDialog** : présente une boîte de dialogue pour l'ouverture d'un fichier
 - **showSaveDialog** : pour la sauvegarde d'un fichier.
 - **showDialog** : permet de spécifier des chaînes de caractères pour le bouton de validation et le titre de la fenêtre afin de créer des boîtes personnalisées.
- Les trois méthodes renvoient un entier dont la valeur correspond au bouton qui a été cliqué.

Les composants atomiques

Les dialogues de sélection de fichiers : JFileChooser

- Contrairement aux boîtes de dialogues, un objet doit être instancié.
- La méthode `getSelectedFile` renseigne sur le nom du fichier sélectionné.



```
import javax.swing.*;
public class TestJFileChooser {
    public static void main(String[] args) {
        JFrame fen = new JFrame("Test JFileChooser");
        fen.setVisible(true);
        fen.setSize(500, 500);
        JFileChooser fc = new JFileChooser();
        if (fc.showOpenDialog(fen) == JFileChooser.APPROVE_OPTION){
            System.out.println(" Le fichier est : " + fc.getSelectedFile());
        }
    }
}
```


Les composants atomiques

Les composants orientes texte : `JTextField`, `JTextArea`, `JEditorPane`

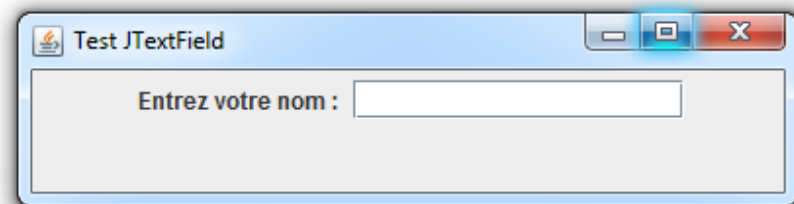
- Swing propose cinq composants pour travailler avec du texte: `JTextField`, `JFormattedTextField`, `JPasswordField`, `JTextArea`, `JEditorPane` et `JTextPane`.
 - descendent (directement ou non) de `JTextComponent`.
- `JTextComponent` : Implémente une architecture MVC (Model-View-Controller).
 - Sans rentrer dans les détails, elle sépare la partie affichage des données des données elles-mêmes.
- Pour la classe `JTextComponent` cela se manifeste sous la forme d'une classe membre interne de type `Document` qui contient les données texte.

Les composants atomiques

Le champ de saisie JTextField

- **JTextField** est utilisé pour saisir une seule ligne de texte.
- Construit un champ de saisie dont la largeur peut être fixée avec **setColumns**.

```
import javax.swing.*;  
public class TestJTextField {  
    public static void main(String[] args) {  
        JFrame fen = new JFrame("Test JTextField");  
        JPanel pan = new JPanel();  
        JLabel lNom = new JLabel("Entrez votre nom : ");  
        JTextField tfNom = new JTextField();  
        tfNom.setColumns(15);  
        pan.add(lNom);  
        pan.add(tfNom);  
        fen.setContentPane(pan);  
        fen.setSize(400, 100);  
        fen.setVisible(true);  
    }  
}
```

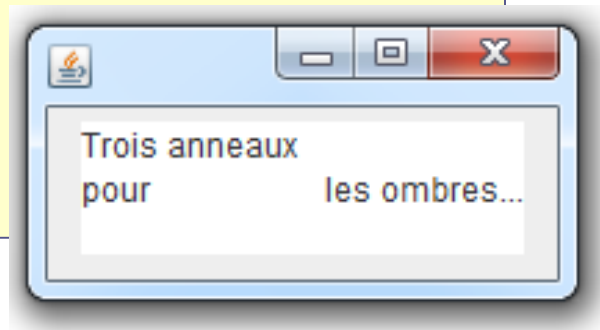


Les composants atomiques

La zone de texte JTextArea

- **JTextArea** est utilisé pour afficher un texte sur plusieurs lignes.
- Le texte est affiché en bloc, avec une police unique (mais modifiable).

```
import javax.swing.*;  
public class TestJTextArea {  
    public static void main(String[] args) {  
        JFrame fen = new JFrame();  
        JPanel pan = new JPanel();  
        JTextArea taNotes = new JTextArea();  
        taNotes.setText("Trois anneaux\npour les ombres...\n");  
        taNotes.setEditable(false);  
        pan.add(taNotes);  
        fen.setContentPane(pan);  
        fen.setSize(400, 100);  
        fen.setVisible(true);  
    }  
}
```

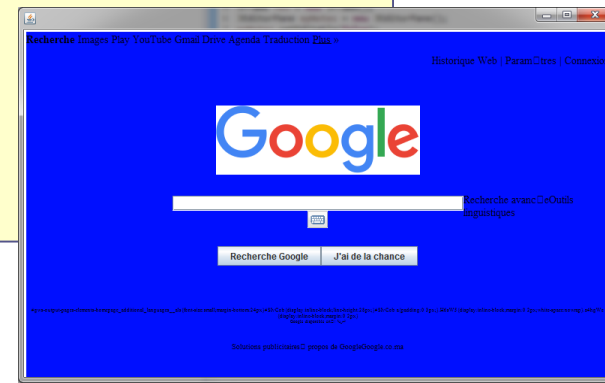


Les composants atomiques

Visualisateur de documents formates JEditorPane

- **JEditorPane** permet de gérer des affichages complexes ou/et des documents plus riches (formats RTF et HTML).

```
import java.io.IOException;
import javax.swing.*.*;
public class TestJEditorPane {
    public static void main(String[] args) {
        JFrame fen = new JFrame();
        JEditorPane epNotes = new JEditorPane();
        epNotes.setEditable(false);
        try {
            epNotes.setPage(" http://www.google.com");
            fen.setContentPane(epNotes);
            fen.setSize(800, 500);
            fen.setVisible(true);
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```



Les composants atomiques

Le composant JTable

- **JTable** permettant d'afficher un tableau formé d'un certain nombre de lignes et de colonnes.
- **JTable** a également une ligne d'en-tête présentant un titre pour chaque colonne.
 - On peut voir les données comme un tableau à deux dimensions dans lequel chaque valeur correspond à la valeur d'une cellule du tableau.
 - Quant aux en-têtes, on peut les voir comme un tableau de chaînes de caractères.
- Le **JTable** utilise différents concepts de Swing :
 - **TableModel** : Un modèle pour stocker les données.
 - **TableCellRenderer** : Un *renderer* pour le rendu des cellules.
 - **TableCellEditor** : Un éditeur pour l'édition du contenu d'une cellule.

Les composants atomiques

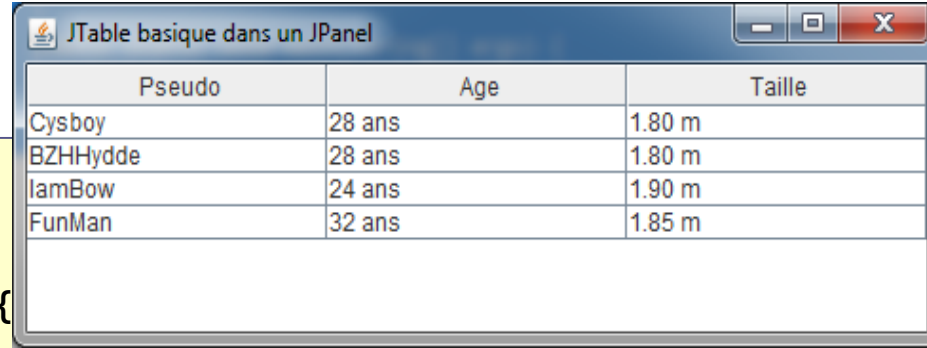
Le composant JTable

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

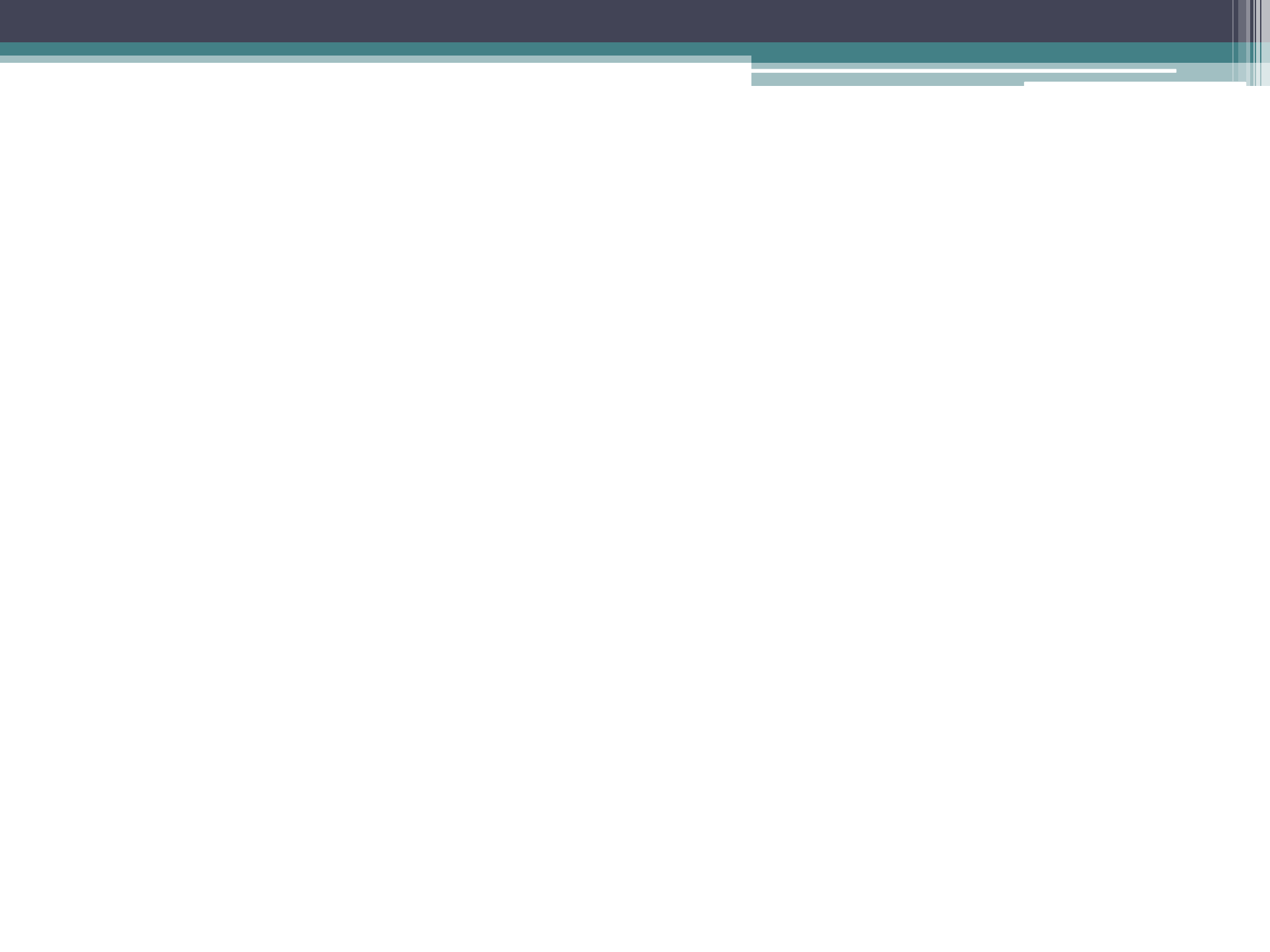
public class TableBasique extends JFrame {
    public TableBasique() {
        setTitle("JTable basique dans un JPanel");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Object[][] donnees = {
            {"Cysboy", "28 ans", "1.80 m"},
            {"BZHHydde", "28 ans", "1.80 m"},
            {"IamBow", "24 ans", "1.90 m"},
            {"FunMan", "32 ans", "1.85 m"}
        };
        String[] titreColonnes = {"Pseudo", "Age", "Taille"};
        JTable tableau = new JTable(donnees, titreColonnes);

        getContentPane().add(tableau.getTableHeader(), BorderLayout.NORTH);
        getContentPane().add(tableau, BorderLayout.CENTER);
        pack();
    }

    public static void main(String[] args) {
        new TableBasique().setVisible(true);
    }
}
```



Pseudo	Age	Taille
Cysboy	28 ans	1.80 m
BZHHydde	28 ans	1.80 m
IamBow	24 ans	1.90 m
FunMan	32 ans	1.85 m



Les composants atomiques

Le positionnement des composants

- Java dispose des composants graphiques en fonction de règles simples qui permettront un aspect visuel quasiment identique d'un système à l'autre.
- Ces règles de placement sont définies à l'aide d'objets :
 - Les gestionnaires de placement.

Conteneur	Gestionnaire par défaut
JPanel, JApplet	FlowLayout
JFrame, JWindow	BorderLayout

- Les gestionnaires de placement implémentent l'interface **LayoutManager** et utilisent les méthodes **getPreferredSize** / **getMinimumSize**, pour connaître la taille préférée/ minimale des composants.

Les composants atomiques

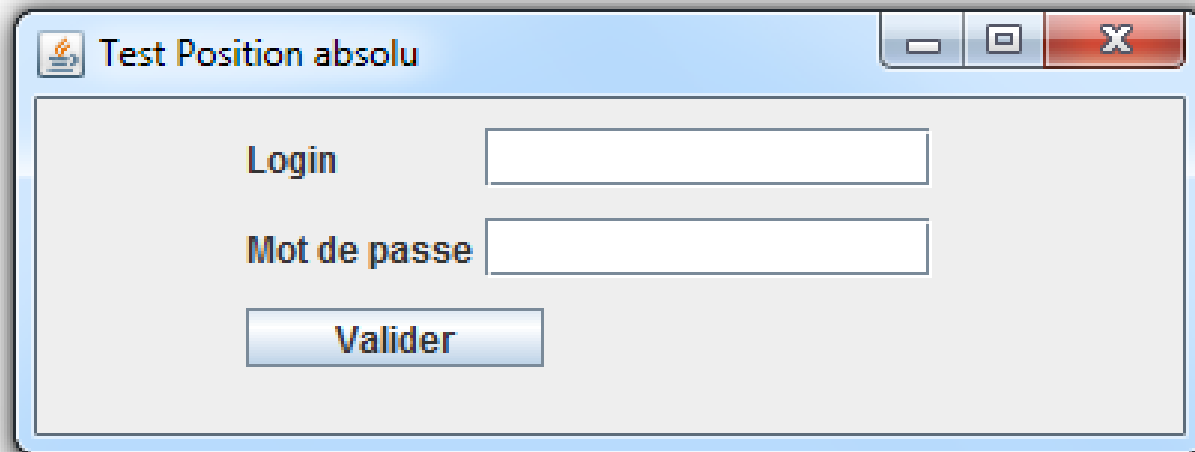
Le positionnement absolu

- Approche peu recommandée.
- On utilise `setLayout(null)` pour désactiver le gestionnaire par défaut du conteneur.
- Les composants sont ensuite placés en utilisant les coordonnées absolues à l'aide de la méthode `setLocation`.
- La taille du composant peut être imposée en utilisant la méthode `setSize`.

Exemple : Position absolu

```
public class TestPositionAbsolu {  
    public static void main(String[] args) {  
        JFrame fen = new JFrame("Test Position absolu");  
        fen.setLayout(null);  
        JLabel loginLabel = new JLabel("Login");  
        JLabel passwordLabel = new JLabel("Mot de passe");  
  
        JTextField loginText = new JTextField(15);  
        JPasswordField passwordText = new JPasswordField(15);  
  
        JButton validerButton = new JButton("Valider");  
  
        loginLabel.setLocation(70, 10);  
        loginLabel.setSize(150, 20);  
        // loginLabel.setBounds(70, 10, 150, 20);  
  
        loginText.setBounds(150, 10, 150, 20);  
        passwordLabel.setBounds(70, 40, 150, 20);  
        passwordText.setBounds(150, 40, 150, 20);  
        validerButton.setBounds(70, 70, 100, 20);  
  
        fen.add(loginLabel);    fen.add(loginText);  
        fen.add(passwordLabel); fen.add(passwordText);  
        fen.add(validerButton);  
        fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        fen.setSize(400, 150);    fen.setVisible(true);  
    }  
}
```

Exemple : Position absolu



The image shows a Java Swing window titled "Test Position absolu". The window has a light blue title bar with standard Windows-style controls (minimize, maximize, close). The main content area is light gray and contains three elements positioned absolutely:

- A label "Login" in black text, followed by a white rectangular text input field.
- A label "Mot de passe" in black text, followed by a white rectangular text input field.
- A blue rectangular button with the text "Valider" in white.

The elements are arranged vertically and horizontally, demonstrating absolute positioning in a GUI.

LayoutManager

- Lorsque vous ajoutez un composant à un composant conteneur, ce dernier doit calculer la position et la taille du nouveau composant.
- Pour cela, le composant conteneur utilise un gestionnaire de disposition appelé « LayoutManager »
- Un « LayoutManager » implante différentes règles pour placer les composants les uns par rapport aux autres.
- Ainsi, la position et la taille des composants dans une interface graphique n'est pas décidée par les composants, mais par un « LayoutManager »

LayoutManager

Plusieurs types de « LayoutManager » sont disponibles:

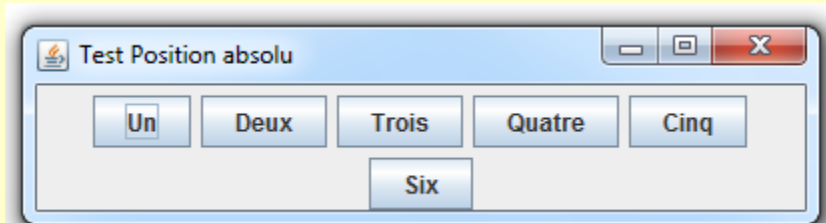
- **BorderLayout** : initialisé pour tous les composants de haut niveau
- **BoxLayout** : simple ligne ou colonne
- **CardLayout** : zone qui contient des composants pouvant changer en cours de fonctionnement
- **FlowLayout** : par défaut pour tous les JPanel – une ligne avec passage à la ligne par manque de place
- **GridLayout** : utilisation d'une grille de cellules ayant la même taille
- **GridBagLayout** : utilisation d'une grille de cellules qui peuvent être de tailles différentes, les composants peuvent en outre être sur plusieurs cellules
- **SpringLayout** : layout dynamique pour le redimensionnement

Le gestionnaire `FlowLayout`

- **`FlowLayout`** est le plus élémentaire.
- Dispose les différents composants de gauche à droite et de haut en bas (sauf configuration contraire du conteneur).
 - Remplit une ligne de composants puis passe à la suivante comme le ferait un éditeur de texte.
 - Le placement peut suivre plusieurs justifications, notamment à gauche, au centre et à droite.
 - Une version surchargée du constructeur permet de choisir cette justification (bien qu'elle puisse aussi être modifiée en utilisant la méthode **`setAlignement`**).
 - Les justifications sont définies à l'aide de variables statiques **`FlowLayout.LEFT`**, **`FlowLayout.CENTER`** et **`FlowLayout.RIGHT`**.

Exemple : Test Position absolu

```
import java.awt.*;
import javax.swing.*;
public class TestFlowLayout extends JPanel {
    private static final long serialVersionUID = 1L;
    public TestFlowLayout() {
        FlowLayout fl = new FlowLayout();
        this.setLayout(fl);
        this.add(new JButton("Un"));
        this.add(new JButton("Deux"));
        this.add(new JButton("Trois"));
        this.add(new JButton("Quatre"));
        this.add(new JButton("Cinq"));
        this.add(new JButton("Six"));
    }
    public static void main(String[] args) {
        JFrame fen = new JFrame("Test Position absolu");
        fen.setContentPane(new TestFlowLayout());
        fen.setSize(400, 100);
        fen.setVisible(true);
    }
}
```

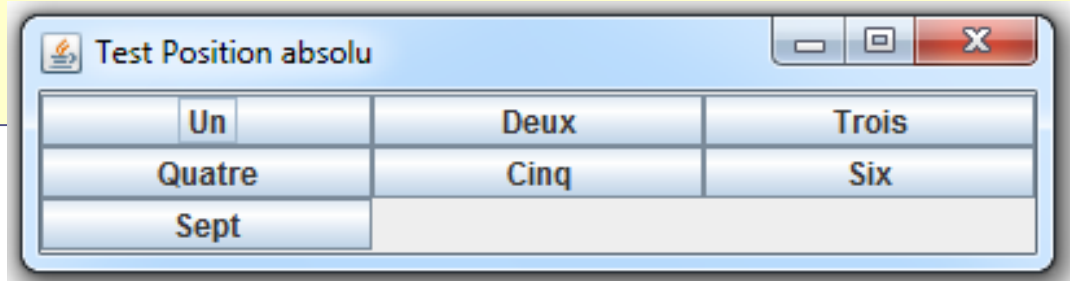


Le gestionnaire GridLayout

- Le gestionnaire **GridLayout** propose de placer les composants sur une grille régulièrement espacée.
- Généralement les composants sont disposés de gauche à droite puis de haut en bas.
 - ➔ Cette disposition peut être modifiée en utilisant la méthode **ComponentOrientation** du conteneur.
- Le nombre de lignes et de colonnes sont fixés à l'aide des méthodes **setRows** et **setColumns**.

Exemple : GridLayout

```
import java.awt.*;
import javax.swing.*;
public class TestGridLayout extends JPanel {
    public TestGridLayout() {
        this.setLayout(new GridLayout(3, 0));
        this.add(new JButton(" Un "));
        this.add(new JButton(" Deux "));
        this.add(new JButton(" Trois "));
        this.add(new JButton(" Quatre "));
        this.add(new JButton(" Cinq "));
        this.add(new JButton(" Six "));
        this.add(new JButton(" Sept "));
    }
    public static void main(String[] args) {
        JFrame fen = new JFrame("Test Position absolu");
        fen.setContentPane(new TestGridLayout());
        fen.setSize(400, 100);
        fen.setVisible(true);
    }
}
```

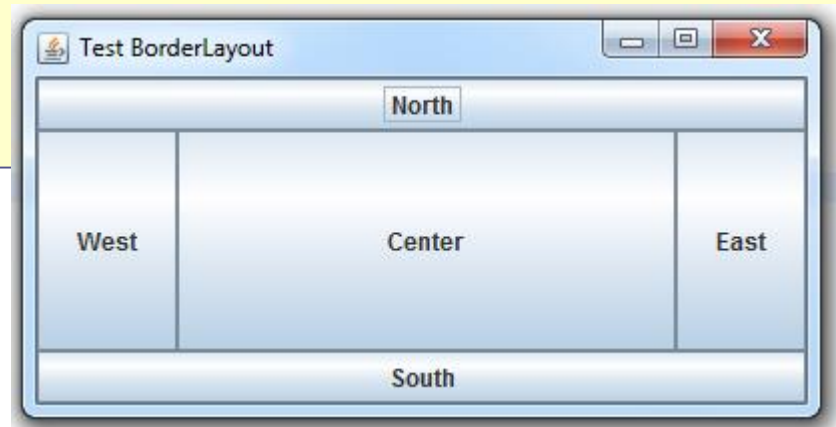


Le gestionnaire BorderLayout

- Le gestionnaire **BorderLayout** définit cinq zones dans le conteneur :
 - une zone centrale (**CENTER**) et quatre zones périphériques (**EAST**, **WEST**, **NORTH**, **SOUTH**).
- Les composants placés dans les zones **NORTH** et **SOUTH** sont dimensionnés à la hauteur qu'ils souhaitent puis étendus en largeur.
- A l'inverse les composants des zones **EAST** et **WEST** sont placés à leurs largeurs voulues et étendus en hauteur.
- Le composant placé dans la zone **CENTER** est utilisé pour combler le vide.
- Pour ajouter un composant, on utilise une version surchargée de la méthode **add**, **add(component, constraints)** où **component** est un composant et **constraints** un objet représentant la position voulue.

Exemple : BorderLayout

```
import java.awt.*;
import javax.swing.*;
public class TestBorderLayout extends JPanel {
    public TestBorderLayout() {
        this.setLayout(new BorderLayout());
        this.add(new JButton(" North "), BorderLayout.NORTH);
        this.add(new JButton(" South "), BorderLayout.SOUTH);
        this.add(new JButton(" East "), BorderLayout.EAST);
        this.add(new JButton(" West "), BorderLayout.WEST);
        this.add(new JButton(" Center "), BorderLayout.CENTER);
    }
    public static void main(String[] args) {
        JFrame fen = new JFrame("Test BorderLayout");
        fen.setContentPane(new TestBorderLayout());
        fen.setSize(400, 200);
        fen.setVisible(true);
    }
}
```



Le gestionnaire GridBagLayout

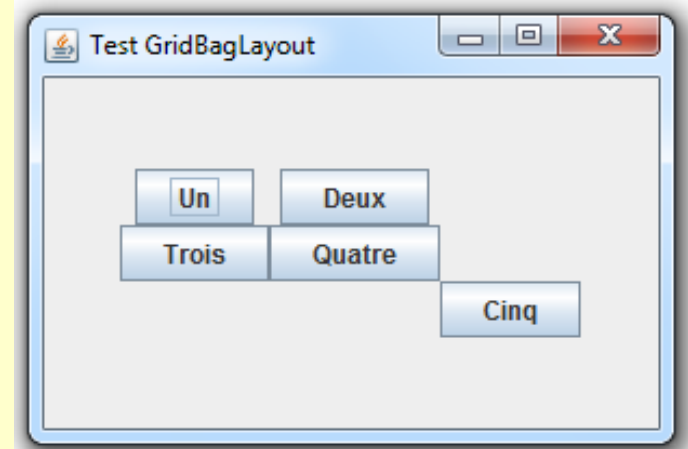
- **GridBagLayout** est le gestionnaire le plus complet et le plus souple.
- Comme le **GridLayout**, il place les composants sur une grille tout en autorisant les composants à prendre des libertés.
 - les composants peuvent occuper plusieurs cellules, se répartir l'espace restant,...
- Le constructeur de **GridBagLayout** n'admet aucun paramètre, ils sont passés par la méthode **add** via un objet de type **GridBagConstraints**.
- Le gestionnaire **GridBagLayout** utilise plus d'une dizaine de paramètres, pour éviter une surcharge trop lourde de la méthode **add**, on passe un objet de la classe **GridBagConstraints** qui représente tout ou partie de ces paramètres.

Exemple : GridBagLayout

```

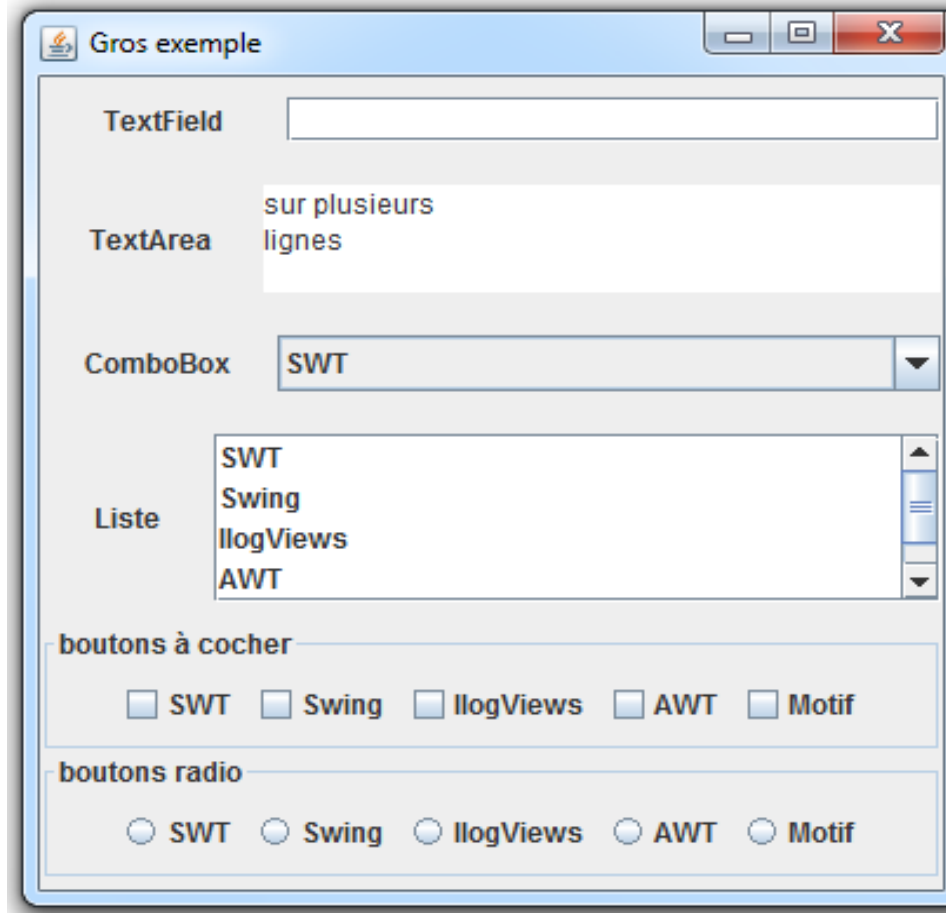
public class TestGridBagLayout extends JPanel {
    public TestGridBagLayout() {
        this.setLayout(new GridBagLayout());
        GridBagConstraints contraintes = new GridBagConstraints();
        contraintes.gridx = 0;
        contraintes.gridy = 0;
        this.add(new JButton(" Un "), contraintes);
        contraintes.gridx = 1;
        contraintes.gridy = 0;
        this.add(new JButton(" Deux "), contraintes);
        contraintes.gridx = 0;
        contraintes.gridy = 1;
        this.add(new JButton(" Trois "), contraintes);
        contraintes.gridx = 1;
        contraintes.gridy = 1;
        this.add(new JButton(" Quatre "), contraintes);
        contraintes.gridx = 2;
        contraintes.gridy = 2;
        this.add(new JButton(" Cinq "), contraintes);
    }
    public static void main(String[] args) {
        JFrame fen = new JFrame("Test GridBagLayout");
        fen.setContentPane(new TestGridBagLayout());
        fen.setSize(300, 200);
        fen.setVisible(true);
    }
}

```



Un exemple complet

- Voici un exemple complet utilisant les composants de base de Swing



Un exemple complet

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.TitledBorder;
public class BigExample {

    ...

    private JPanel createMainPanel(String... list) {
        return createBoxPanel(createTextFieldPanel(), createTextAreaPanel(),
            createComboBoxPanel(list), createListPanel(list),
            createCheckBoxes(createTitledPanel("boutons à cocher"), list),
            createRadioButton(createTitledPanel("boutons radio"), list));
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Gros exemple");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        BigExample example = new BigExample();
        JPanel panel = example.createMainPanel("SWT", "Swing", "IlogViews",
            "AWT", "Motif");
        frame.setContentPane(panel);
        frame.setSize(400, 400);
        frame.setVisible(true);
    }
}
```

Bordure et Assemblage de l'interface

- N'importe quel composant swing peut avoir

```
public class BigExample {  
  
    ...  
  
    private JPanel createBoxPanel(JPanel... panels) {  
        JPanel panel = new JPanel(null);  
        panel.setLayout(new BorderLayout(panel, BorderLayout.Y_AXIS));  
        for (JPanel p : panels)  
            panel.add(p);  
        return panel;  
    }  
  
    private JPanel createTitledPanel(String title) {  
        JPanel panel = new JPanel();  
        TitledBorder border = new TitledBorder(title);  
        panel.setBorder(border);  
        return panel;  
    }  
  
    ...  
  
}
```


Etiquette, champs de texte, zone de texte

- `JLabel`, `TextField` et `TextArea` prennent un texte lors de la construction

```
public class BigExample {
    private JPanel createForm(String text, JComponent c) {
        JPanel panel = new JPanel(new GridBagLayout());
        GridBagConstraints constraints = new GridBagConstraints();
        constraints.weightx = 1.0;
        panel.add(new JLabel(text), constraints);
        constraints.weightx = 5.0;
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.gridwidth = GridBagConstraints.REMAINDER;
        panel.add(c, constraints);
        return panel;
    }
    public JPanel createTextFieldPanel() {
        JTextField textField = new JTextField();
        return createForm("TextField", textField);
    }
    public JPanel createTextAreaPanel() {
        JTextArea textArea = new JTextArea("sur plusieurs\nlignes\n");
        return createForm("TextArea", textArea);
    }
    ...
}
```

Liste et Liste déroulante

- **JComboBox** et **JList** peuvent prendre un tableau (**String[]**) à la construction

```
public class BigExample {  
    ...  
    public JPanel createComboBoxPanel(String... list) {  
        JComboBox<String> comboBox = new JComboBox<String>(list);  
        return createForm("ComboBox", comboBox);  
    }  
  
    public JPanel createListPanel(String... array) {  
        JList<String> list = new JList<String>(array);  
        list.setVisibleRowCount(4);  
        JScrollPane scrollPane = new JScrollPane(list);  
        return createForm("Liste", scrollPane);  
    }  
    ...  
}
```

Boutons à cocher

- **JCheckBox** correspond à une case à cocher
- **JRadioButton** correspond à un bouton radio

```
public class BigExample {  
    ...  
    private JPanel createCheckBoxes(JPanel panel, String... list) {  
        for (String s : list) {  
            JCheckBox checkBox = new JCheckBox(s);  
            panel.add(checkBox);  
        }  
        return panel;  
    }  
    ...  
}
```

- **JCheckBox**, **JRadioButton** comme **JButton** héritent d'**AbstractButton**

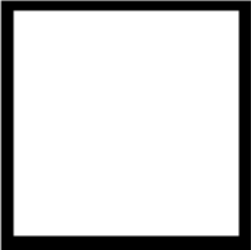
Boutons radio et groupe de boutons

- Pour avoir un seul bouton sélectionné à la fois, le bouton doit faire partie d'un **ButtonGroup**

```
public class BigExample {  
    ...  
    private JPanel createRadioButton(JPanel panel, String... list) {  
        ButtonGroup group = new ButtonGroup();  
        for (String s : list) {  
            JRadioButton radioButton = new JRadioButton(s);  
            panel.add(radioButton);  
            group.add(radioButton);  
        }  
        return panel;  
    }  
    ...  
}
```

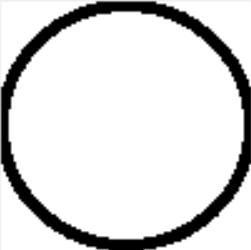
- **ButtonGroup** est un composant logique pas graphique

Java Contrôle 1

 Carré = Calculer

Périmètre = 0

Surface = 0

 Cercle = Calculer

Périmètre = 0

Surface = 0

Supprimer