

# Chapitre :

# Les interfaces graphiques en JAVA

A series of horizontal lines in teal and light blue colors, with some lines having a slight 3D effect, extending across the width of the slide.

# Les composants atomiques

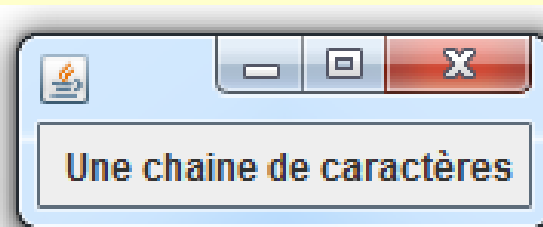
- Les composants atomiques sont tous les composants élémentaire de Swing.
- Ce sont les boutons, les labels, les menus, . . .

# 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 constructeur.

```
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);
    }
}
```

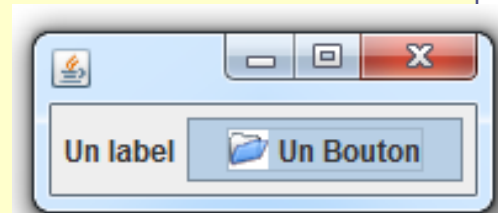
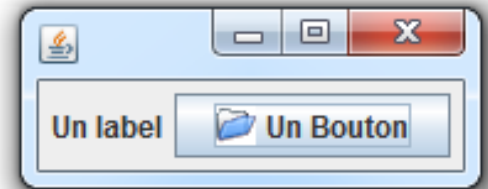


# Les composants atomiques

## 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

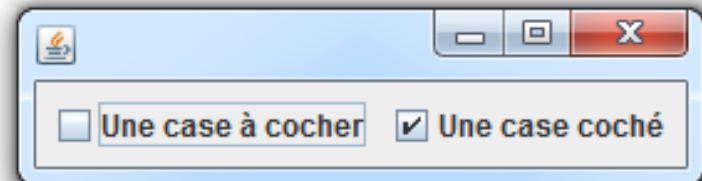
## Les cases a cocher : JCheckBox

- 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);
    }
}
```



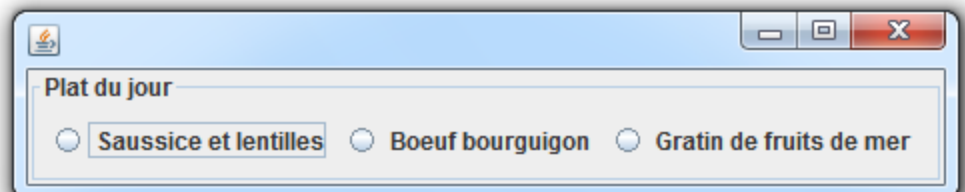


# Les composants atomiques

## 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

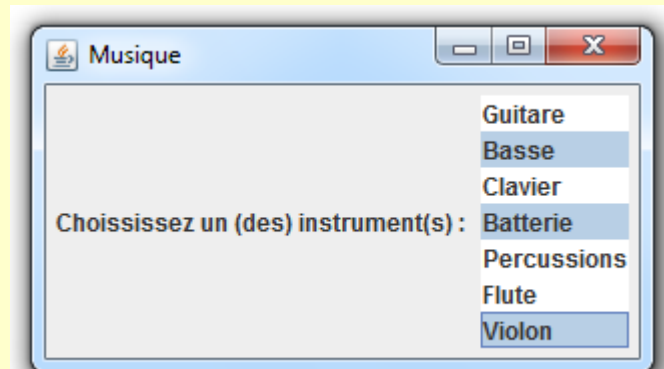
## Les listes de choix : JList

- **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);
    }
}
```

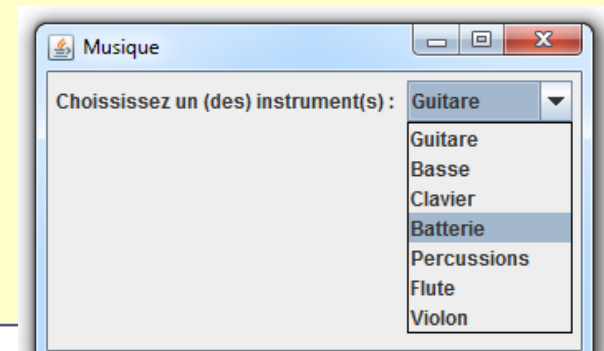


# Les composants atomiques

## 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);
    }
}
```

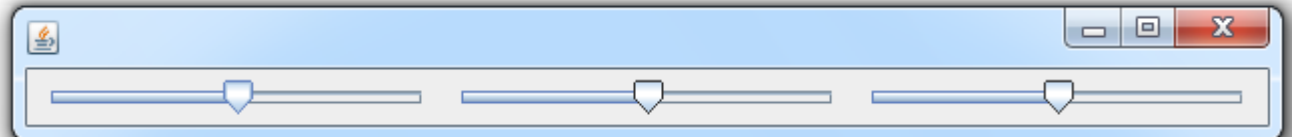


# Les composants atomiques

## 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.

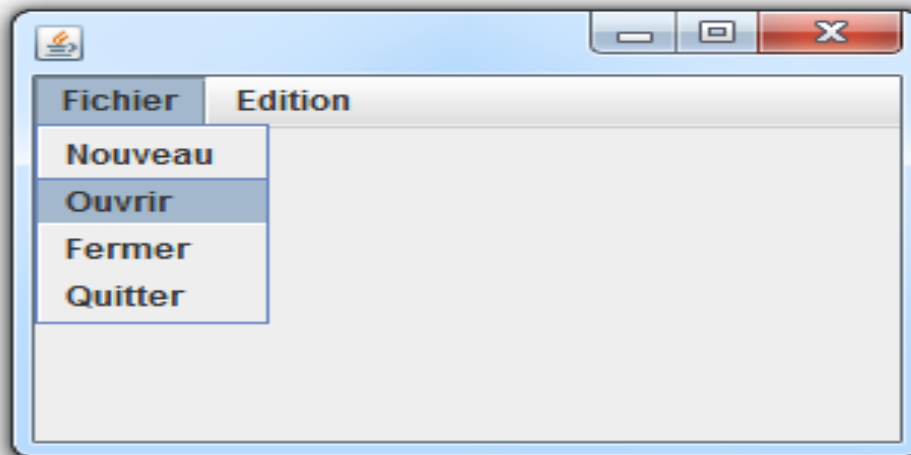
```
import javax.swing.*;  
public class TestJSlider {  
    public static void main(String[] args) {  
        JFrame fen = new JFrame();  
        JPanel pan = new JPanel();  
        JSlider sBlue = new JSlider();  
        JSlider sRed = new JSlider();  
        JSlider sGreen = new JSlider();  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

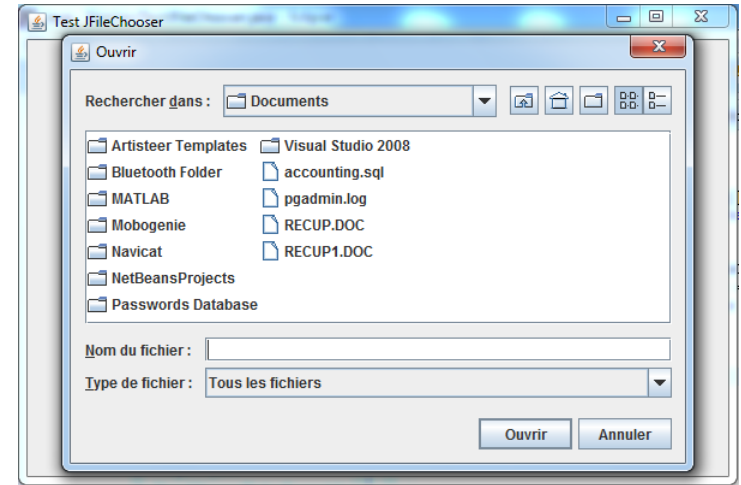
- Swing propose une boîte de sélection de fichier(s) avec la classe **JFileChooser**.
- La classe propose trois méthodes pour afficher un dialogue d'ouverture de fichier.
  - **showOpenDialog** : présente une boîte de dialogue pour l'ouverture d'un fichier
  - **showSaveDialog** : pour la sauvegarde 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`

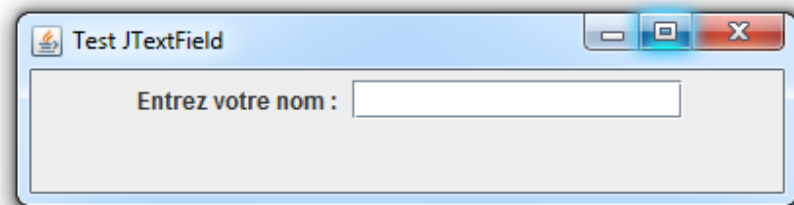
- Swing propose cinq composants pour travailler avec du texte: `JTextField`, `JFormattedTextField`, `JPasswordField`, `JTextArea`, `JEditorPane` et `JTextPane`.
  - descendent (directement ou non) de `JTextComponent`.
- `JTextComponent` : Implémente une architecture MVC (Model-View-Controller).
  - Sans 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);
    }
}
```

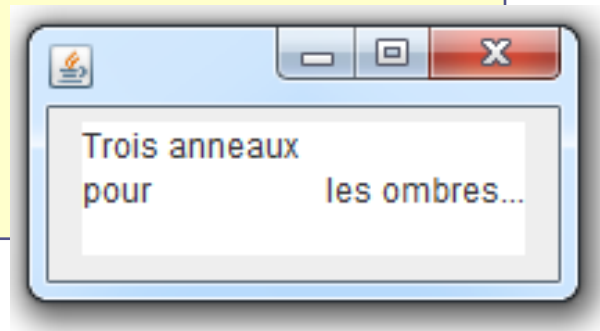


# Les composants atomiques

## 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);
    }
}
```

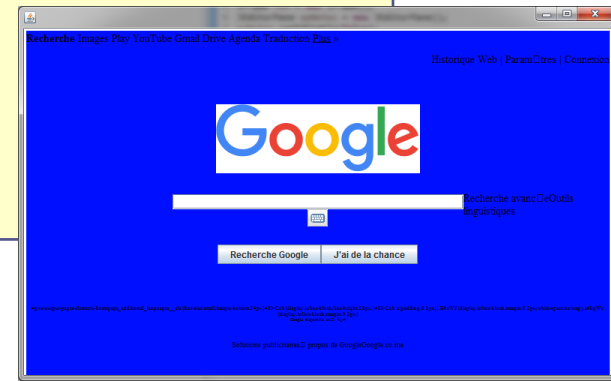


# Les composants atomiques

## 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

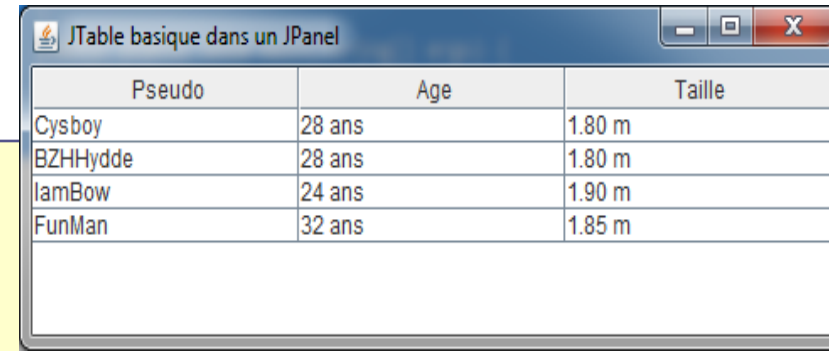
## Le composant JTable

```
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

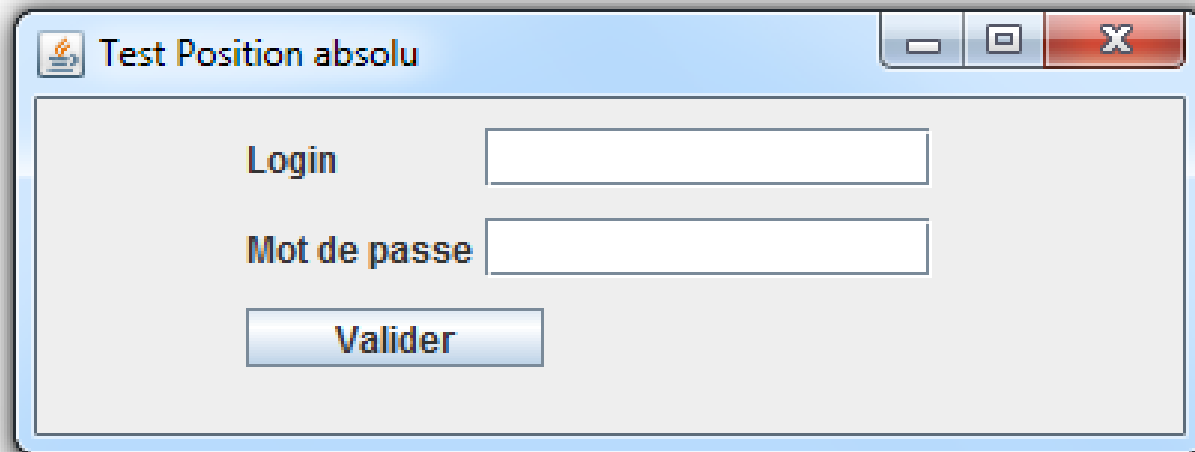
## Le positionnement absolu

- 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 has a light gray background. It contains two text input fields and a button, all positioned using absolute coordinates. The first field is labeled "Login" and the second is labeled "Mot de passe" (Password). Below these fields is a button labeled "Valider" (Validate).

Test Position absolu

Login

Mot de passe

Valider

# 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:

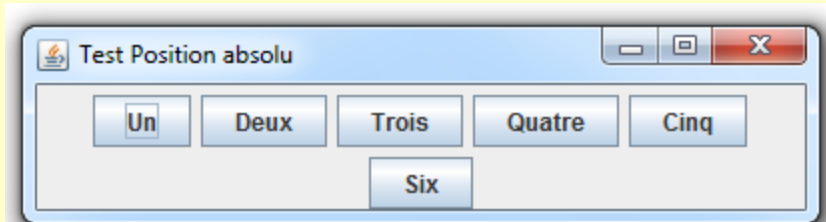
- **BorderLayout** : initialisé pour tous les composants de haut niveau
- **BoxLayout** : simple ligne ou colonne
- **CardLayout** : zone qui contient des composants pouvant changer en cours de fonctionnement
- **FlowLayout** : par défaut pour tous les JPanel – une ligne avec passage à la ligne par manque 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);
    }
}
```



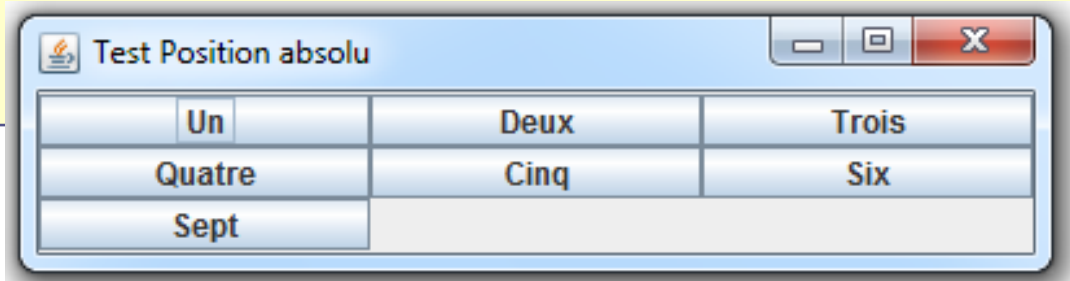
# Le gestionnaire GridLayout

- 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);
    }
}
```

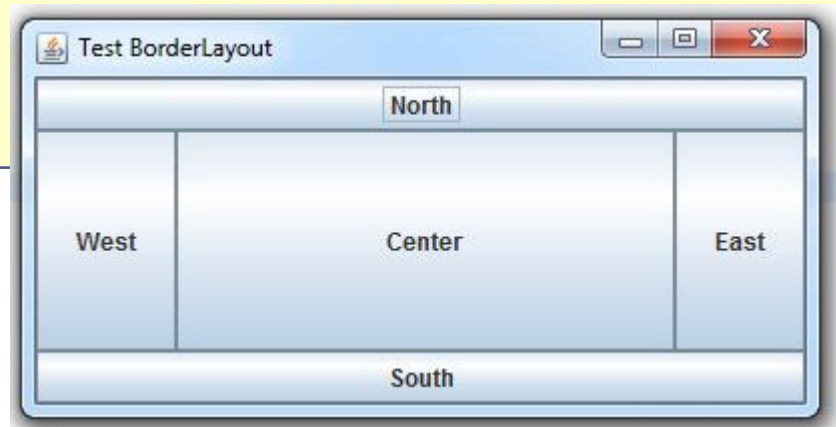


# Le gestionnaire BorderLayout

- 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.

# Example : 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);
    }
}
```

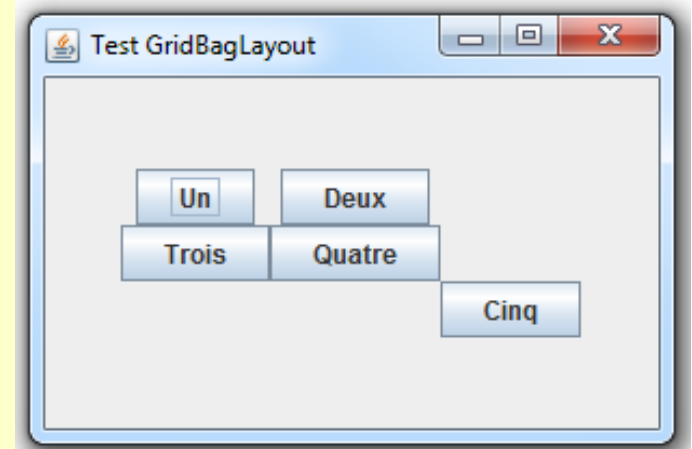


# Le gestionnaire GridBagLayout

- **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.

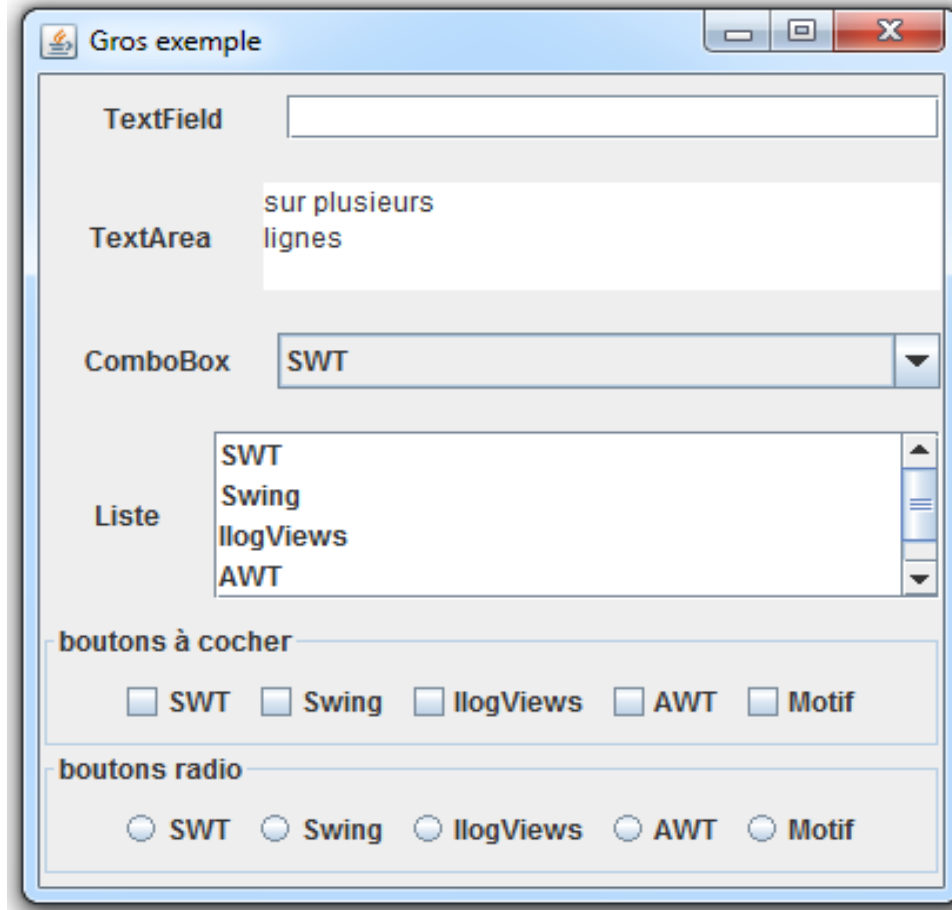
# Exemple : GridBagLayout

```
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