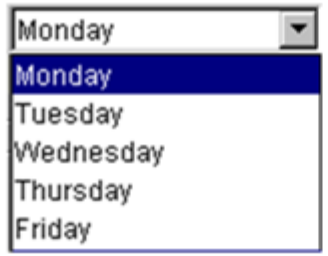
Interface graphique SWING

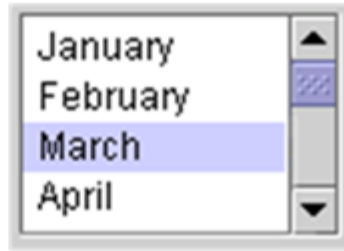
ISET BIZERTE

CHALOUAH Anissa

Buttons



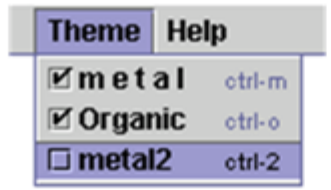
Combo Box



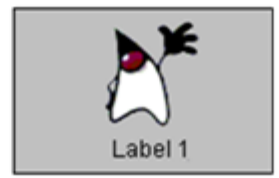
List

TextField

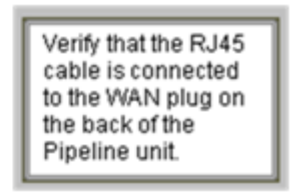
Slider



Menu



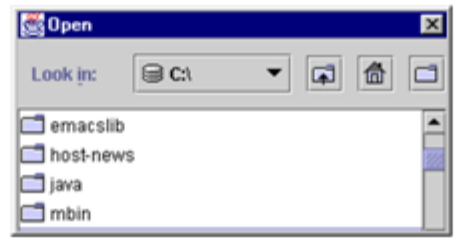
Label



Text Area

Tool Tip

Progress Bar

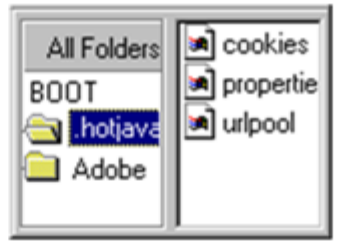


File Chooser

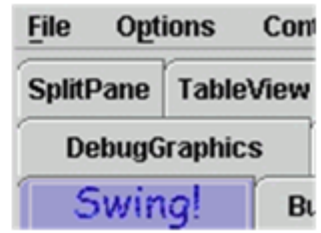
Color Chooser

Table

Tree



Split Pane



Tabbed Pane

Interfaces graphiques

- Les interfaces graphiques assurent le dialogue entre les utilisateurs et une application.
- 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.***

Modèle d'une application graphique

- Une application graphique est composée de 3 éléments:
  - Les composants graphiques élémentaires de l'application (Boutons, menu, Etiquettes, zones de textes, zones de choix, …). Ce sont les widgets.
  - Le conteneur graphique global qui est une zone spatiale sur laquelle les composants sont placés. Parmi les conteneurs distingue les Fenêtres, les Frames, les panneaux, les boîtes de dialogues…
  - Le gestionnaire de présentation, dit LayoutManager, qui fixe la politique de placement des composants sur le conteneur. Parmi les gestionnaires, on distingue le FlowLayout, BorderLayout, le GridLayout, …

Les APIs

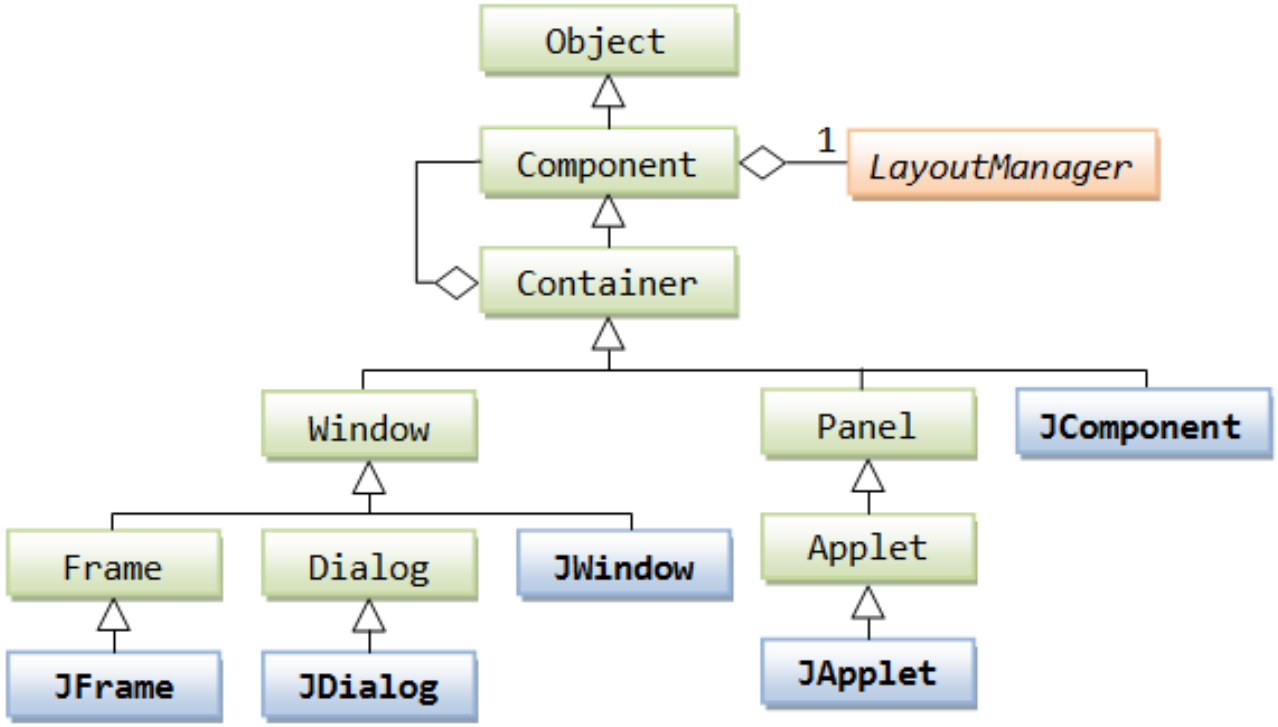
- Plusieurs packages permettent de gérer les interfaces graphiques : **AWT** et **SWING** (intégré à partir de la version 1.2).
- AWT utilise des composants lourds, c'est à dire utilisant les ressources du système d'exploitation, alors que Swing utilise des composants dits légers n'utilisant pas ces ressources.
- Swing est plus robuste que l'AWT, plus portable, et plus facile à utiliser.
- Swing ne remplace pas complètement AWT mais fournit des composants d'interface plus performants et plus ergonomiques.

Concepts de bases

- Les composants (components)
  - Les éléments constituant les interfaces graphiques sont appelés composants (ce sont des boutons, des textes, des images, ...).
- Les conteneurs (containers)
  - Certains composants sont capables d'en accueillir d'autres, ce sont les conteneurs.
  - les conteneurs sont des descendants de la classe Container.
- Les gestionnaires de placement(Layouts)
  - Le placement des composants dans les container est souvent confié à un gestionnaire de placement.
- Les gestionnaires d'évènements
  - Les actions de l'utilisateur sont représentées par des événements.
  - Le programme peut modifier son comportement en fonction de certains événements.

Conteneurs primaires

Hiérarchie

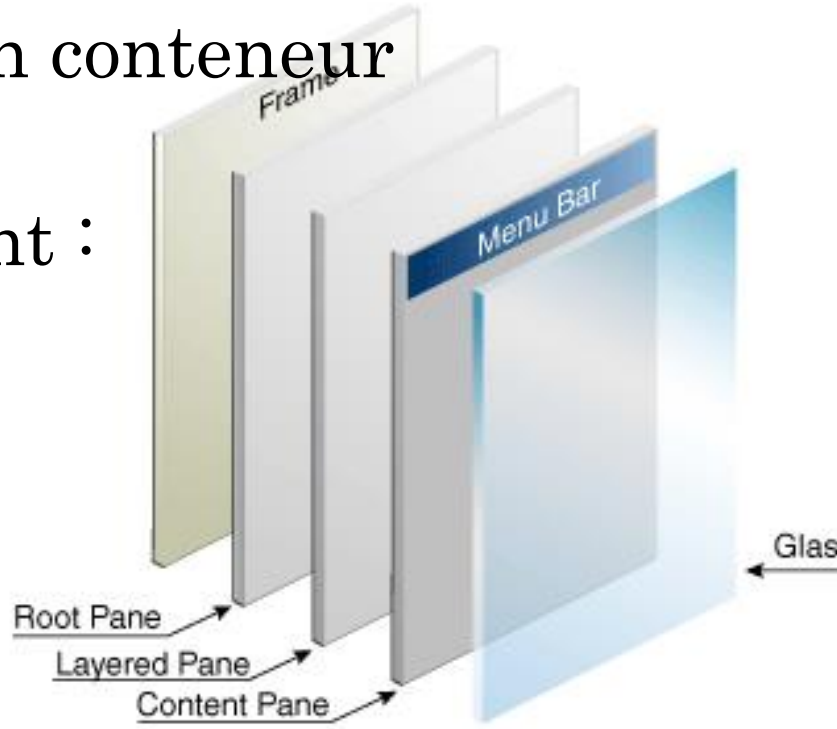

Object
Component
1
LayoutManager
Container
Window
Panel
JComponent
Frame
Dialog
JWindow
Applet
JFrame
JDialog
JApplet

Méthodes de la classe Component

- **`setVisible(boolean visible)`** affiche ou masque le composant
- **`getSize()`** donne la dimension actuelle du composant, retourne le type Dimension qui est utilisable ainsi : getSize().height et getSize().width
- **`getPreferedSize()`** donne la taille "idéale" du composant, retourne le type Dimension
- **`setSize(Dimension d)`** redimensionne le composant à la dimension indiquée
- **`setSize(int largeur, int hauteur)`** redimensionne le composant
- **`move(int coordX, int coordY)`** déplace le composant au point indiqué (coin haut et gauche)
- **`setEnabled(boolean actif)`** active ou non le composant, c'est à dire le rend sensible aux événements.
- **`setForeground(Color couleur)`** définit la couleur d'avant-plan (de dessin) du composant
- **`setBackground(Color couleur)`** définit la couleur de fond

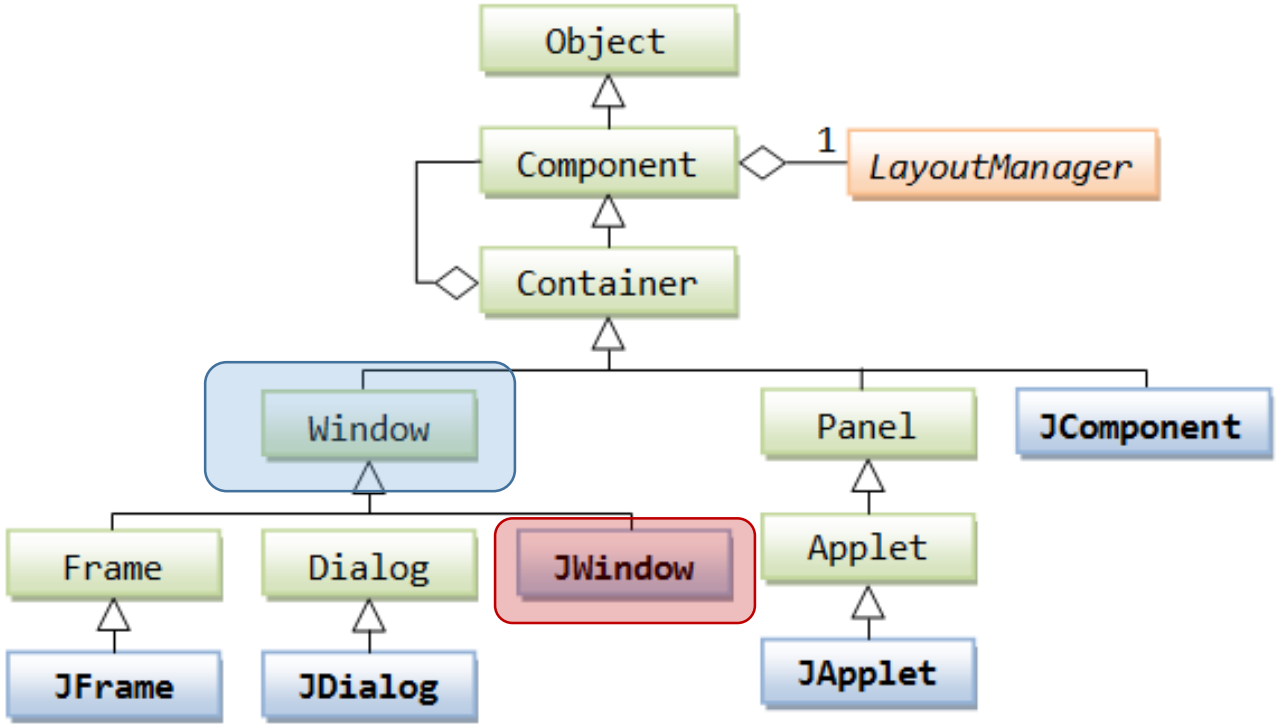
Les conteneurs primaires (Top-Level containers)

- Les conteneurs primaires sont les seuls composants qui sont dessinés par le système d'exploitation, ce sont donc des composants lourds.
- Toute application graphique doit utiliser un conteneur primaire.
- Trois types de conteneurs primaires existent :
  - les fenêtres (JFrame et Jwindow)
  - les boites de dialogues (JDialog)
  - les applets (JApplet)



Les conteneurs primaires (Top-Level containers)

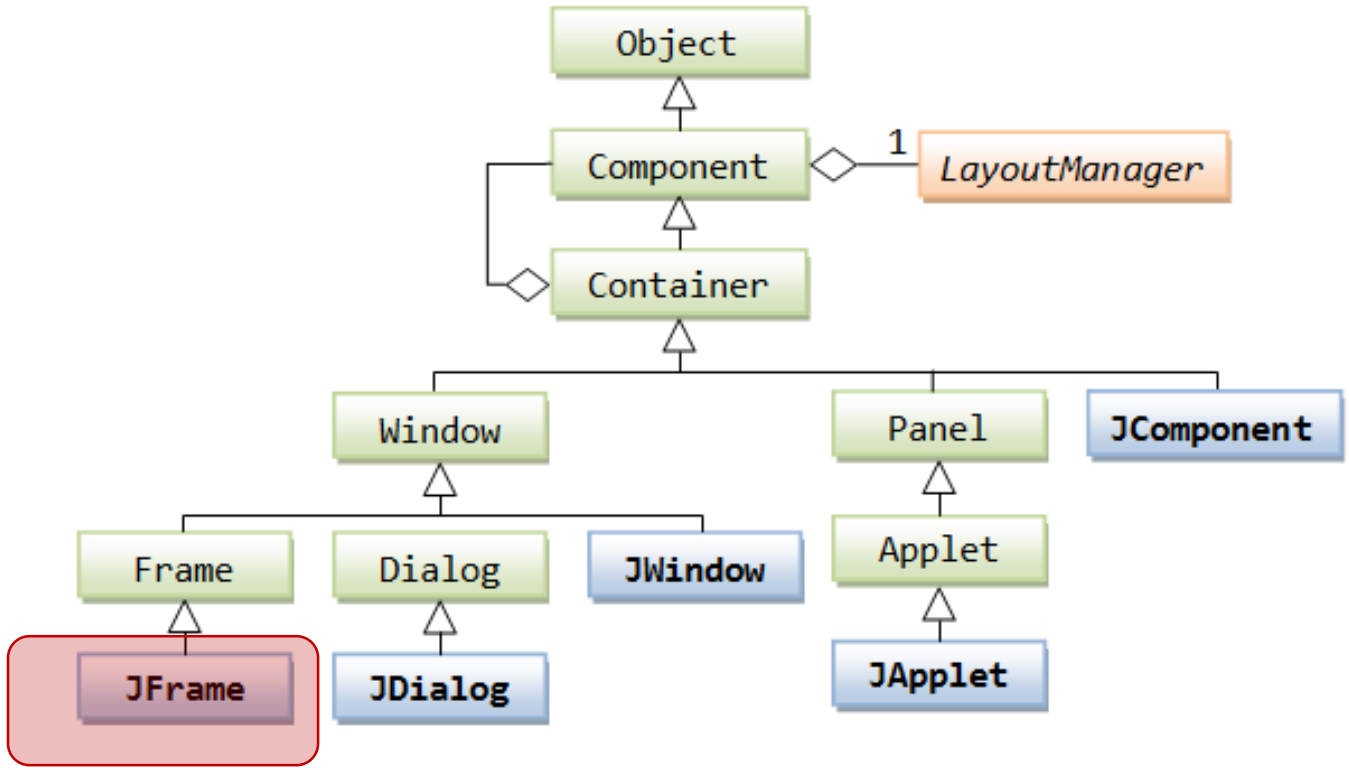
- Les quatre conteneurs primaires sont construit sur le même modèle.
  - La couche supérieure est un le **GlassPane**, un panneau transparent qui est utilisé pour la gestion des évènements.
  - Sous ce panneau se situe le **contentPane** qui accueille les composants graphiques. Le **contentPane** contient aussi une barre de menu, accessible via la méthode setJMenuBar,
  - Il est contenu par le **LayeredPane** qui peut être utilisé pour empiler des composants à des "profondeurs" différentes.
  - Tous ces éléments sont contenu dans un élément principal de type **JRootPane**.


Object
Component
1
LayoutManager
Container
Window
Panel
JComponent
Frame
Dialog
JWindow
Applet
JFrame
JDialog
JApplet

JWindow

- La classe JWindow permet de créer une fenêtre graphique dans le système de fenêtrage utilisée.
- Cette fenêtre n'a aucune bordure et aucun bouton. Elle ne peut être fermée que par le programme qui l'a construite

```
import javax. swing. JWindow ;
public class TestJWindow {
        public static void main( String [] args) {
                JWindow fen = new JWindow ();
                fen.setSize (300 ,300);
                fen.setLocation(500 ,500);
                fen.setVisible( true);
        }
}
```


Object
Component
1
LayoutManager
Container
Window
Panel
JComponent
Frame
Dialog
JWindow
Applet
JFrame
JDialog
JApplet

Fenêtre : JFrame

- Conteneur d'une application avec barre de titre et des boutons de fermeture, plein écran, iconification.
- Importer la classe javax. swing. JFrame ;

Constructeur	Rôle
JFrame()	
JFrame(String)	Création d'une instance en précisant le titre

Fenêtre : JFrame

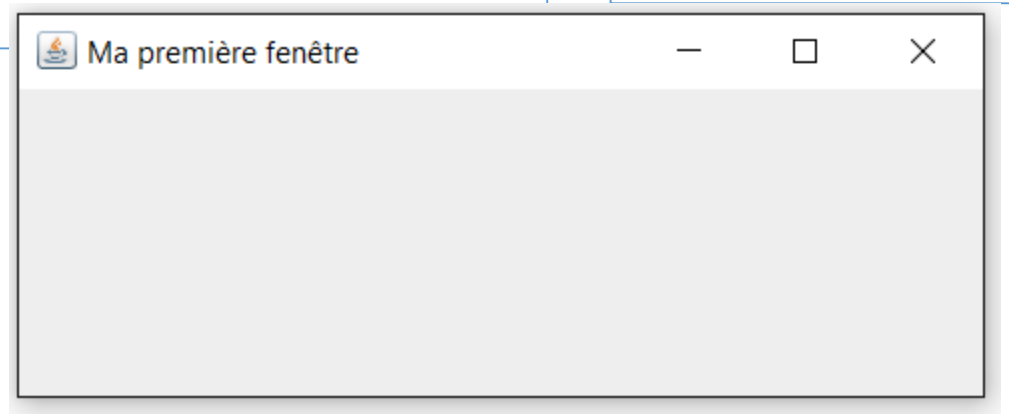
Méthode	Rôle
setTitle(String titre)	spécifie le titre du cadre
String getTitle()	Obtient le titre du cadre
void setIconImage(Image img)	définit l'image qu'il faut afficher quand ce cadre est iconifié
setMenuBar(MenuBar)	applique une barre de menu en haut du cadre
setResizable(boolean)	définit la possibilité pour l'utilisateur de modifier la taille de la fenêtre

- Les événements générés par le composant JFrame sont : ***WindowOpened, WindowClosing, WindowClosed, WindowIconified, WindowDeiconified, WindowActivated, WindowDeactivated***.

JFrame : 2 manières de créer une JFrame

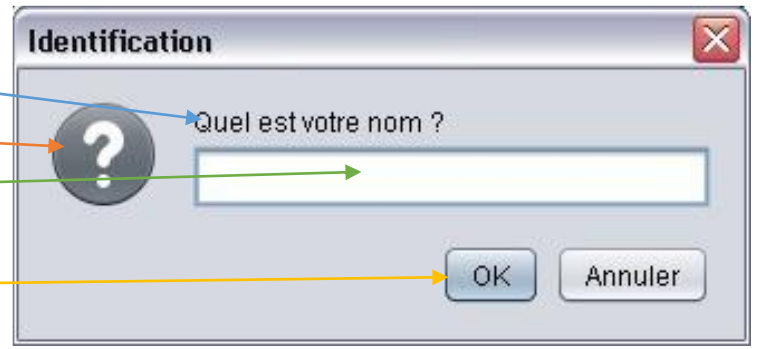
```
import javax. swing. JFrame ;
public class MaFenetre {

public MaFenetre(){
 JFrame fen = new JFrame ("Ma première fenêtre");
 fen . setSize (500 ,200);
 fen . setVisible( true);
 fen . setLocation(500 ,500);
}

public static void main( String [] args) {
         MaFenetre f=new MaFenetre();
   }
}
```

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

 public MaFenetre(){
         super ("Ma première fenêtre");
         setSize (500 ,200);
         setVisible( true);
         setLocation(500 ,500);
}
public static void main( String [] args) {
         MaFenetre f=new MaFenetre();
}
}
```



Boites de dialogue : JOptionPane(1)

- Le composant complexe **JOptionPane** permet de créer une **fenêtre de dialogue** (fenêtre séparée ou *pop-up*) qui servira à **afficher un message** ou à **obtenir des informations** de la part de l'utilisateur.
- La classe **JOptionPane** crée automatiquement un conteneur de haut niveau **JDialog** pour afficher les informations et recevoir les réponses de l'utilisateur.
- Une fenêtre **JOptionPane** comprend 4 zones (optionnelles)
  - Message
  - Icône
  - Entrée
  - Bouton



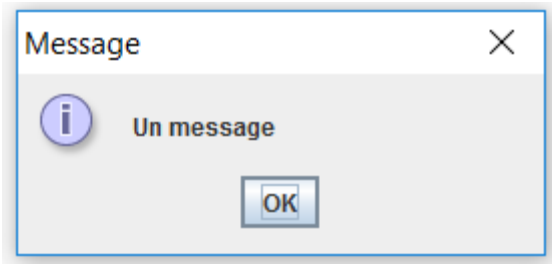
Boites de dialogue : JOptionPane (2)

- Les fenêtres de dialogue peuvent être divisées en quatre catégories différentes :
  - **Message** : Affichage d'un message d'information
  - **Input** : Saisie d'une information entrée par l'utilisateur
  - **Confirm** : Demande de confirmation (*Ok, Oui, Non, Annulation*)
  - **Option** : Choix d'une option spécifique (parmi une liste de boutons)
- Les méthodes statiques **show***Typ***Dialog()** créent des fenêtres de dialogue de haut-niveau. Ces fenêtres sont **modales** (ce qui signifie que le focus ne peut pas être transféré à un composant d'une autre fenêtre de l'application sans fermer la fenêtre de dialogue).
- Le premier paramètre (de toutes les méthodes **show***Typ***Dialog()**) indique à quel composant (parent) la fenêtre de dialogue est liée ce qui détermine sa dépendance et son positionnement (si **null**, la fenêtre est centrée à l'écran).

Boites de dialogues
JOptionPane : message

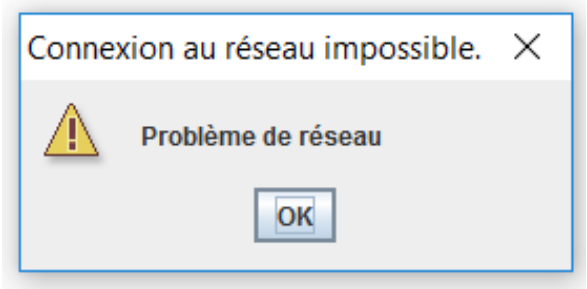
- Ce sont les boites de dialogue les plus simple, elles informent l'utilisateur en affichant un texte simple et un bouton de confirmation.

```
JOptionPane.showMessageDialog(fen,"Un message");
```



- On peut modifier l'aspect de la fenêtre en fonction du type de message, par exemple pour afficher un dialogue d'avertissement :

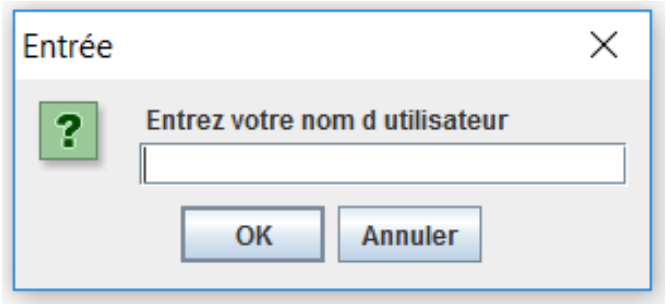
```
JOptionPane.showMessageDialog(fen,"Problème de réseau",
"Connexion au réseau impossible.",JOptionPane.WARNING_MESSAGE);
```



Boites de dialogues
JOptionPane : input

- 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.

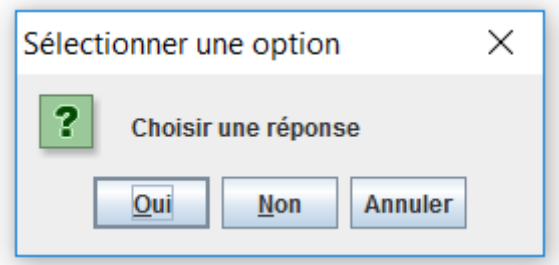
```
String rep = JOptionPane. showInputDialog(fen," Entrez votre nom d utilisateur");
```



Boites de dialogues
JOptionPane : confirm

- Les boites de dialogues peuvent aussi être utilisées pour demander un renseignement à l'utilisateur.
- Les méthodes **showConfirmDialog()** affichent des fenêtres qui comportent des boutons de type *Cancel", "Yes", "No"* (différentes combinaisons sont possibles sur la base des constantes *(option type)*
- Définies dans la classe **JOptionPane**. Ces méthodes retournent une valeur entière correspondant à une des constantes de **JOptionPane** (**CANCEL_OPTION, CLOSED_OPTION, NO_OPTION, YES_OPTION, OK_OPTION**).

```
int i=JOptionPane.showConfirmDialog(fen,"Choisir une réponse");
if (i==JOptionPane.YES_OPTION) {...}
```



Boites de dialogues
JOptionPane : option

- Les méthodes **showOptionDialog()** permettent de transmettre un tableau d'objets qui seront affichés sous forme deboutons (ils remplacent les boutons standard de **showConfirmDialog()**).
- Ces méthodes retournent une valeur entière correspondant à l'indice du tableau identifiant l'objet qui a été sélectionné (ou **CLOSED_OPTION**).

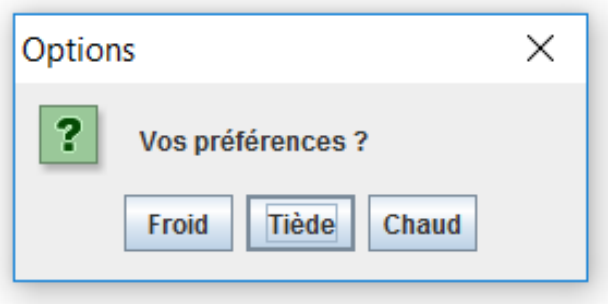
```
String[] tab = {"Froid", "Tiède", "Chaud"};
int i = JOptionPane.showOptionDialog(fen,"Vos préférences ?", "Options"
,JOptionPane.DEFAULT_OPTION,JOptionPane.QUESTION_MESSAGE, null,
tab, tab[1]);
```

Type d'option : pas utilisé si options définies

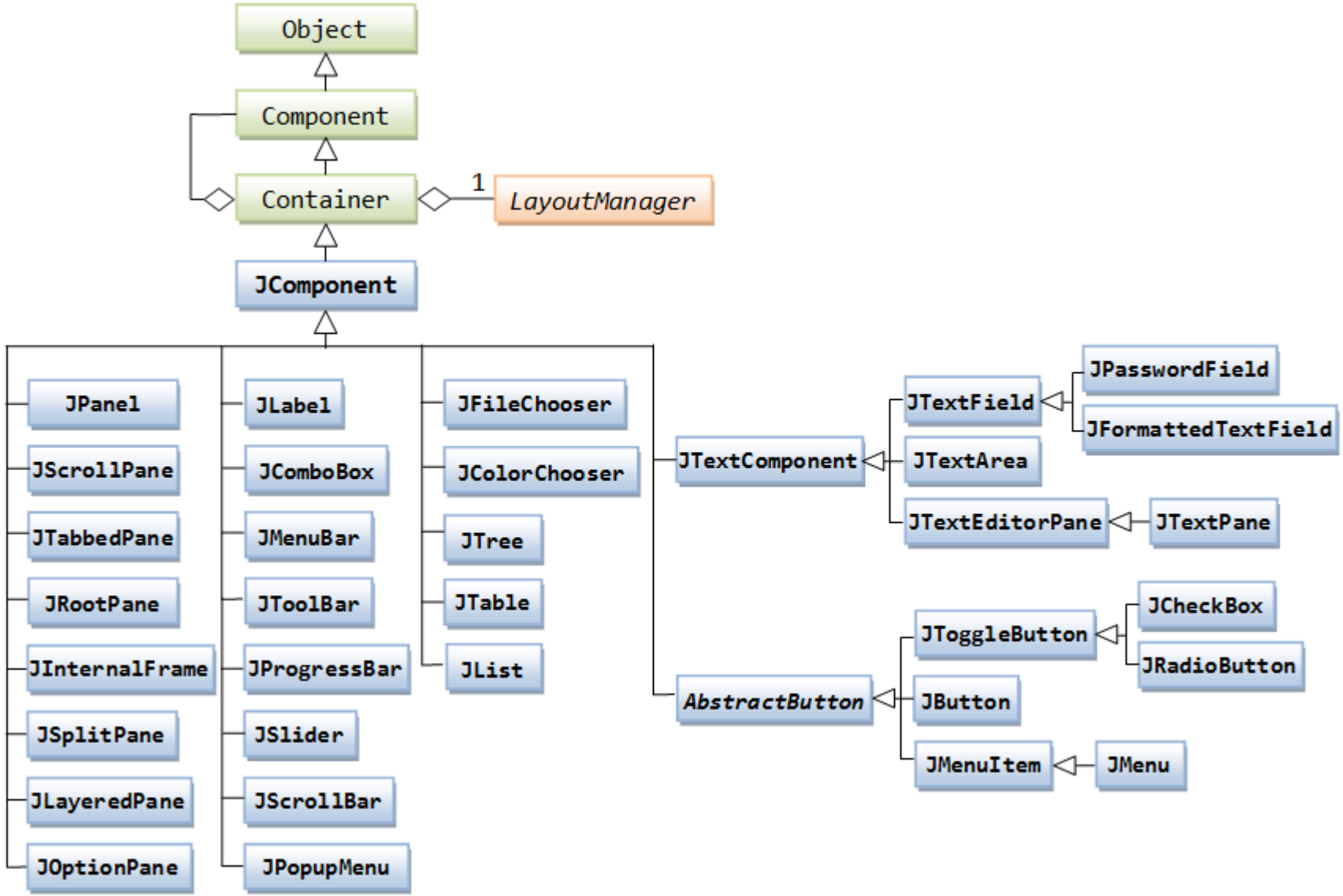
Type de message : question

icône

Valeur initiale



Les composants (component)


Object
Component
Container
1
LayoutManager
JComponent
JPanel
JScrollPane
JTabbedPane
JRootPane
JInternalFrame
JSplitPane
JLayeredPane
JOptionPane
JLabel
JComboBox
JMenuBar
JToolBar
JProgressBar
JSlider
JScrollBar
JPopupMenu
JFileChooser
JColorChooser
JTree
JTable
JList
JTextComponent
JTextField
JPasswordField
JFormattedTextField
JTextArea
JTextEditorPane
JTextPane
AbstractButton
JToggleButton
JCheckBox
JRadioButton
JButton
JMenuItem
JMenu

Les étiquettes : JLabel

- Les étiquettes sont des composants graphiques capables d'afficher un texte et/ou une image
- **Importer javax.swing.JLabel**

Constructeurs	Rôle
JLabel()	Création d'une instance sans texte ni image
JLabel(Icon)	Création d'une instance en précisant l'image
JLabel(Icon, int)	Création d'une instance en précisant l'image et l'alignement horizontal
JLabel(String)	Création d'une instance en précisant le texte
JLabel(String, Icon, int)	Création d'une instance en précisant le texte, l'image et l'alignement horizontal
JLabel(String, int)	Création d'une instance en précisant le texte et l'alignement horizontal

Jlabel : Exemple

```
JLabel  l1 = new JLabel("Etiquette 1");
JLabel l2 = new JLabel( ); l2.setText("Etiquette 2");
JLabel l3 = new JLabel("Etiquette et image", new ImageIcon("/img/duke.gif"),JLabel.CENTER );
```

Les boutons : JButton

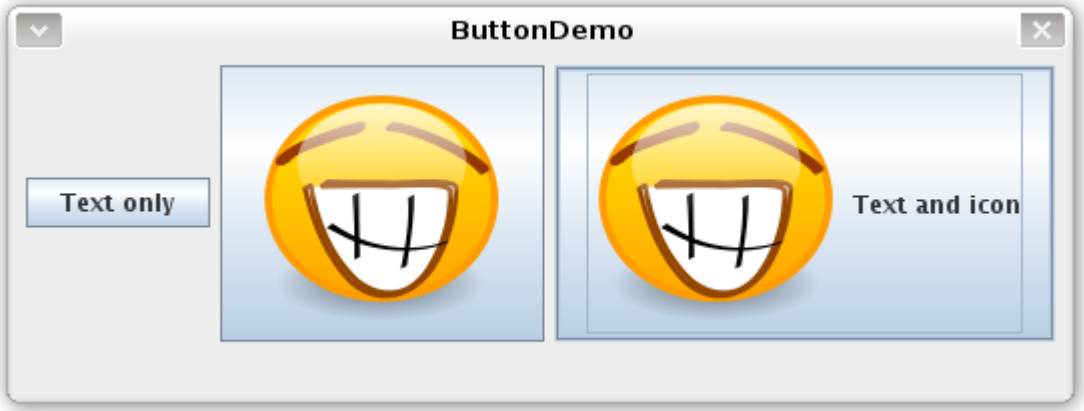
- Le composant **JButton** permet d'afficher un bouton qui peut comprendre un **texte** (libellé), une **icône** (image) ou **les deux à la fois**.
- **Importer javax.swing.JButton**

Constructeur	Rôle
JButton()	
JButton(String)	préciser le texte du bouton
JButton(Icon)	préciser une icône
JButton(String, Icon)	préciser un texte et une icône

- Un bouton permet donc de **déclencher des actions** (activités).

Jbutton : Exemple

```
JButton bt1 = new JButton("Text only");
JButton bt2 = new JButton(new ImageIcon("icone.png"));
JButton bt3 = new JButton("Text and icon",new ImageIcon("icone.png"));
```



Les cases à cocher : JCheckBox

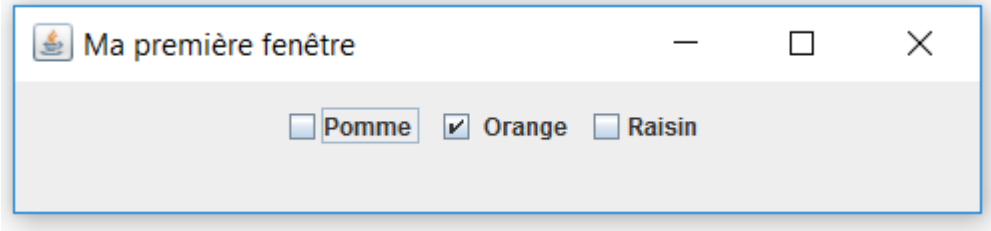
- Le composant **JCheckBox** est une sous-classe de **JToggleButton** et représente une **case à cocher** qui peut prendre deux états (sélectionné/non-sélectionné).
- L'utilisation des composants **JCheckBox** permet de saisir un certain nombre d'**options parmi une liste de choix non-exclusifs**.
- **Importer javax.swing.JCheckBox**

Constructeur	Rôle
JCheckBox(String)	précise l'intitulé
JCheckBox(String, boolean)	précise l'intitulé et l'état
JCheckBox(Icon)	spécifie l'icône utilisée
JCheckBox(Icon, boolean)	précise l'intitulé et l'état du bouton
JCheckBox(String, Icon)	précise l'intitulé et l'icône
JCheckBox(String, Icon, boolean)	précise l'intitulé, une icône et l'état

JCheckBox : Exemple

- Méthodes
  - boolean isSelected() : retourne true si la case est sélectionnée, false sinon,
  - Void setSelected(boolean) : permet de modifier l'état d'une case à cocher

```
JCheckBox c1 = new JCheckBox("Pomme ");
JCheckBox c2 = new JCheckBox(" Orange", true);
JCheckBox c3 = new JCheckBox("Raisin ");
pan.add(c1);
pan.add(c2);
pan.add(c3);
if (c1.isSelected()==true)
 System.out.println("Pomme séléctionnée");
else if (c2.isSelected()==true)
 System.out.println("Orange séléctionnée");
else
 System.out.println("Raisin séléctionné");
```



Les boutons radio : JRadioButton

- Les boutons radio **JRadioButton** sont des boutons à choix **exclusif**, il permettent de choisir un (et **un seul**) élément parmi un ensemble.
- Pour obtenir ce comportement, il suffit d'insérer les composants **JRadioButton** (mutuellement exclusifs) dans un **groupe de boutons** représenté par un objet de la classe **ButtonGroup**.
- **Importer javax.swing.JRadioButton**

Constructeur	Rôle
JRadioButton(String)	précise l'intitulé du bouton radio
JRadioButton(String, boolean)	précise l'intitulé et l'état
JRadioButton(Icon)	spécifie l'icône utilisée
JRadioButton(String, Icon)	précise l'intitulé et l'icône
JRadioButton(String, Icon, boolean)	précise l'intitulé, une icône et l'état

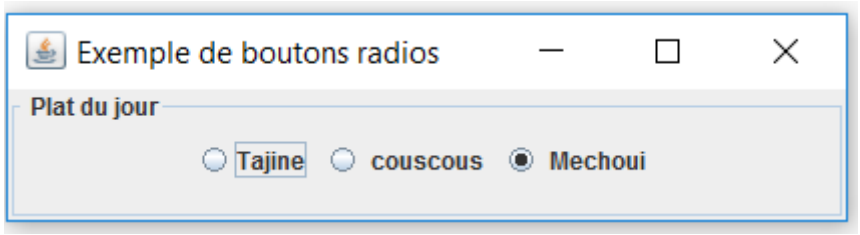
ButtonGroup

- La classe **ButtonGroup** permet de grouper (logiquement) des boutons qui sont enregistrés dans le groupe à l'aide de la méthode **add()**.
- Si un bouton est inséré dans un groupe de boutons, la **sélection de l'un d'eux** provoquera automatiquement la **désélection de tous les autres** membres du groupe (permet de garantir une seule sélection parmi un choix d'options mutuellement exclusives).
- En principe, l'insertion dans un groupe de boutons devrait être **réservée exclusivement aux boutons de type JRadioButton**.

ButtonGroup & JRadioButton : Exemple

```
public class TestRadio extends JPanel {

JRadioButton plat1 , plat2 , plat3;
ButtonGroup plat;
public TestRadio(){
plat1 = new JRadioButton("Tajine");
plat2 = new JRadioButton(" couscous");
plat3 = new JRadioButton(" Mechoui ",true);
plat = new ButtonGroup();
plat.add ( plat1);
plat.add ( plat2);
plat.add ( plat3);
this.add ( plat1);
this.add ( plat2);
this.add ( plat3);
this. setBorder( BorderFactory.
createTitledBorder(" Plat du jour"));
};
}
```

```
public class PlatDuJour{
public static void main( String [] args) {
JFrame fen = new JFrame ("Exemple de boutons radios");
TestRadio panneau = new TestRadio();
fen. setContentPane( panneau );
fen. pack ();
fen. setVisible( true);
}
}
```



Les listes de choix : JList

- Le composant **JList** permet de présenter à l'utilisateur une **liste d'éléments** parmi lesquels il peut en **sélectionner un** ou **plusieurs**.
- Le composant **JList** affiche en permanence le contenu de la liste. Si tous les éléments ne peuvent être affichés, il faut offrir une barre de défilement (**JScrollPane**) pour parcourir les éléments de la liste
- **Importer javax.swing.JList**

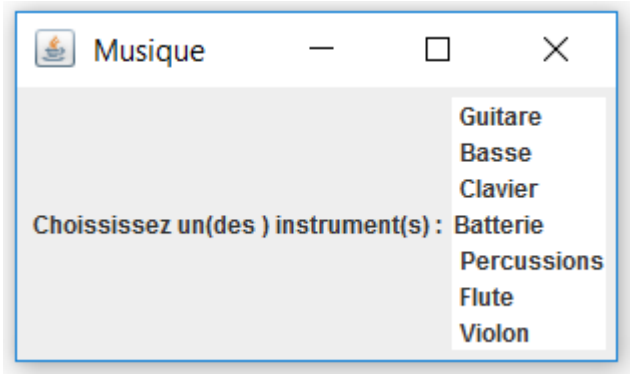
Constructeur	Rôle
Jlist( )	Crée un JList vide.
JList(Object [] donnees)	Crée un JList qui affiche les données contenues dans le tableau données.

JList : Méthodes

int getSelectedIndex()	Retourne l'indice de l'item sélectionné.
int[] getSelectedIncices()	Retourne les indices des items sélectionnés.
void setSelectedIndex(int i)	Affecte l'indice de l'item sélectionné.
void setSelectedIndices(int [] i)	Affecte les indices des items sélectionnés.
Object getSelectedValue()	Retourne la valeur de l'item sélectionné.
void setSelectedValue(Object v, boolean b)	L'item sélectionné devient v. Si b vaut true, il est rendu visible dans le JList.
Object[] getSelectedValues()	Retourne un tableau des items sélectionnés.
void setSelectionMode(int sm)	Affecte le mode de sélection des données. (Sélection unique, multiple ou dans un intervalle)

Jlist : Exemple

```
public class ExempleJList extends JFrame {
JList<String> instruments;
JPanel pan ;
JLabel text;

public ExempleJList(){
  super(" Musique ");
  Container c=this.getContentPane();
String [] lesElements={" Guitare " , " Basse " , " Clavier " , "Batterie " , " Percussions" , "
Flute" , " Violon "};
  instruments = new JList<>(lesElements);
  pan = new JPanel ();
  text = new JLabel (" Choississez un(des ) instrument(s) :");
  pan .add ( text);
  pan .add ( instruments);
  c.add(pan);
  this.pack();
  this.setVisible(true); }

public static void main( String [] args) {
ExempleJList jl=new ExempleJList();
}
}
```



Les boites combo : JComboBox

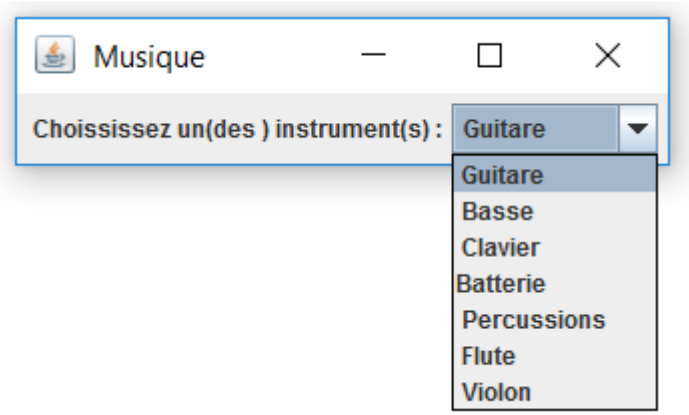
- Les boites combo permettent de choisir **un seul élément** parmi une liste proposée.
- Elles ont un comportement proche des boutons radio. On les utilise quand l'ensemble des éléments à afficher n'est pas connu lors de la conception. En effet, il est difficile de concevoir une interface avec un nombre de boutons radio variable.
- **Importer javax.swing.JComboBox**

Constructeur	Rôle
JComboBox( )	Crée un JComboBox vide.
JComboBox(Object [] donnees)	Crée un JComboBox qui affiche les données contenues dans le tableau données.

- Les méthodes getSelectedIndex et getSelectedItem permettent de connaître l'indice et l'objet sélectionnée.

JComboBox : Exemple

```
public class ExempleJComboBox extends JFrame{
 JComboBox instruments;
 JPanel pan ;
 JLabel text;
 public ExempleJComboBox(){
  super(" Musique ");
  Container c=this.getContentPane();
  String [] lesElements={" Guitare " , " Basse " , " Clavier " , "Batterie " , " Percussions" , "
Flute" , " Violon "};
  instruments = new JComboBox(lesElements);
  pan = new JPanel ();
  text = new JLabel (" Choisissez un instrument :");
  pan .add ( text);
  pan .add ( instruments);
  c.add(pan);
  this.pack();
  this.setVisible(true);
}
public static void main( String [] args) {
  ExempleJComboBox jcb=new ExempleJComboBox();
}

}
```



Les champs textes : JTextField

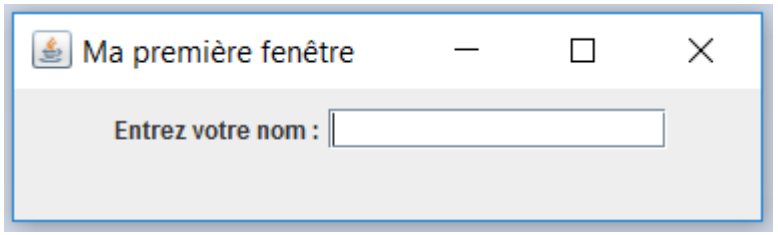
- Un composant pour afficher, saisir et éditer une ligne de texte simple.
- **Importer javax.swing.JTextField**

Constructeur	Rôle
JTextField()	Création d'un champ texte
JTextField(int)	spécification du nombre de caractères à saisir
JTextField(String)	avec texte par défaut
JTextField(String,int)	avec texte par défaut et nombre de caractères à saisir.

- **Méthodes** :
  - String getText( ) : lecture de la chaine saisie.
  - Void setColumns () : Fixer la largeur du champ
  - int getColumns( ) : lecture du nombre de caractères prédéfini

JTextField : Exemple

```
JPanel pan = new JPanel ();
JLabel lNom = new JLabel (" Entrez votre nom :");
JTextField tfNom = new JTextField();
tfNom. setColumns(15);
pan .add ( lNom);
pan .add ( tfNom);
```



Zone texte : JTextArea

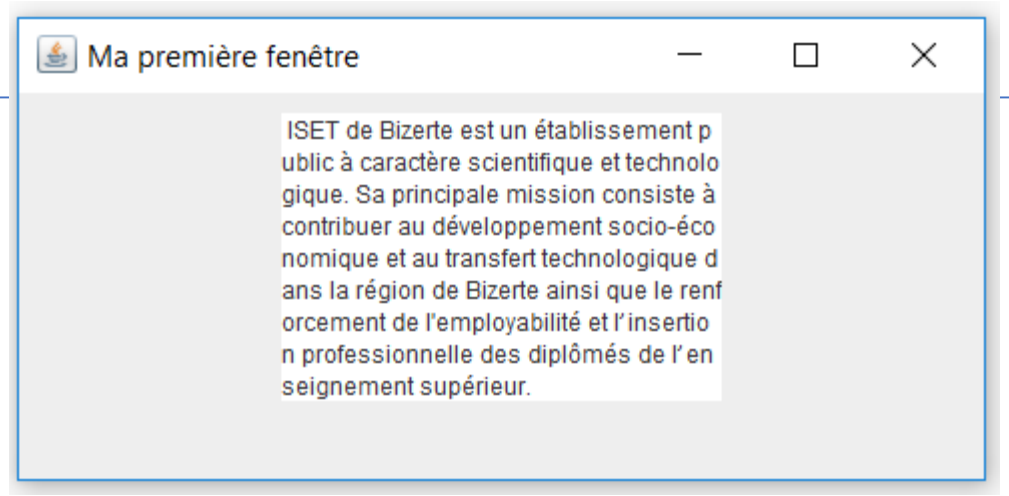
- Le composant **JTextArea** permet d'afficher un texte sur plusieurs lignes
- **Importer javax.swing.JTextArea**

Constructeur	Rôle
JTextArea()	
JTextArea(int, int)	avec spécification du nombre de lignes et de colonnes
JTextArea(String)	avec texte par défaut
JTextArea(String, int, int)	avec texte par défaut et taille

- **Méthodes**
  - Void setLineWrap(boolean) : gère le retour à la ligne automatique
  - Void setEditable (boolean) : Pour rendre la zone de texte éditable ou pas

JTextArea : Exemple

```
JTextArea taNotes = new JTextArea();
taNotes.setColumns(20);
taNotes.setLineWrap(true);
taNotes . setText (" ISET de Bizerte est un établissement public à caractère
scientifique et technologique. Sa principale mission consiste à contribuer au
développement socio-économique et au transfert technologique dans la région de
Bizerte ainsi que le renforcement de l'employabilité et l'insertion professionnelle
des diplômés de l'enseignement supérieur. ");
taNotes . setEditable( false );
pan .add ( taNotes );
```



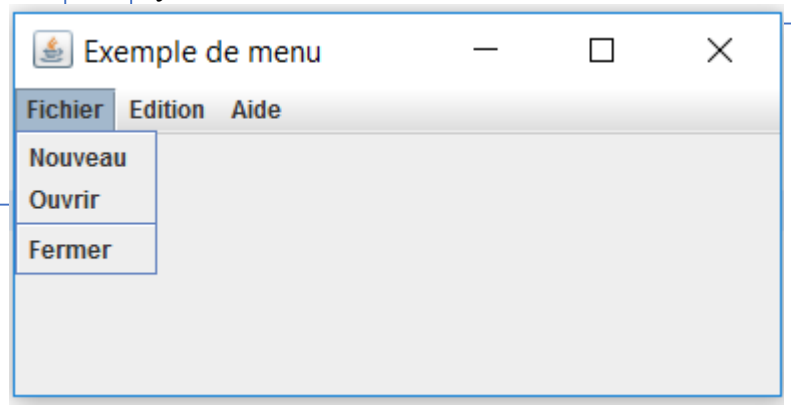
Les menus : JMenuBar, JMenu, JMenuItem

- La construction de menus est hiérarchisée. On utilise un composant JMenuBar pour construire une barre de menus qu'on affecte à la fenêtre grâce à la méthode setJMenuBar,
- Les menus sont construits à partir de la classe JMenu. Ils sont placés dans la JMenuBar avec la méthode add.
- Ces menus sont constitués d'éléments appartenant à la classe JMenuItem ou à l'une de ses classes filles (JRadioButtonMenuItem, JCheckBoxMenuItem).
- Les éléments de menus sont ajoutés aux menus à l'aide de la méthode add.

Les menus : Exemple

```
public class ExempleMenu {
JFrame fr;
JMenuBar mb;
JMenu menuFichier,menuEdition, menuAide;
JMenuItem itemNouveau,itemOuvrir, itemFermer;

public ExempleMenu(){
fr = new JFrame("Exemple de menu");
mb = new JMenuBar();
fr.setJMenuBar(mb);

 menuFichier = new JMenu ("Fichier");
 menuEdition = new JMenu ("Edition");
 menuAide = new JMenu ("Aide");

mb.add(menuFichier ); mb.add(menuEdition );
mb.add(menuAide);
```

```
itemNouveau = new JMenuItem ("Nouveau");
itemOuvrir = new JMenuItem ("Ouvrir");
itemFermer = new JMenuItem ("Fermer");
menuFichier.add(itemNouveau);
menuFichier.add(itemOuvrir);
menuFichier.addSeparator();
menuFichier.add(itemFermer );

fr.setBounds(0,0,400, 200);
fr.setVisible(true);
}
public static void main( String [] args) {
ExempleMenu mn=new ExempleMenu();
}
```

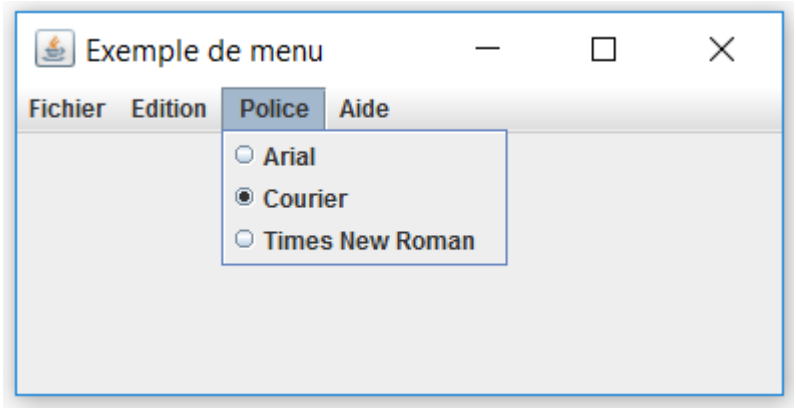


Les menus : JRadioButtonItemMenu & JCheckBoxMenuItem

- La classe JMenuItem a deux classes filles qui implémentent des **boutons radio** et des **cases à cocher à l'intérieur des menus**.
- La classe JRadioButtonItemMenu permet de créer des **boutons radio** adaptés aux menus, comme pour les JRadioButton, les éléments sont regroupés dans des ButtonGroup.
- De même, la classe JCheckBoxMenuItem fournit des **cases à cocher** adaptées aux menus.

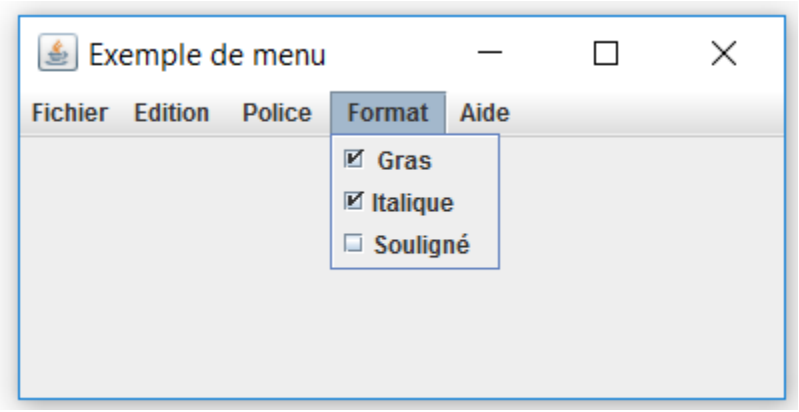
JRadioButtonItemMenu : Exemple

```
JMenu menuFonte = new JMenu(" Police ");
mb.add(menuFonte);
JRadioButtonMenuItem policeArial = new JRadioButtonMenuItem("Arial");
JRadioButtonMenuItem policeCourier = new JRadioButtonMenuItem("Courier ");
JRadioButtonMenuItem policeTimes = new JRadioButtonMenuItem("Times New Roman");

ButtonGroup fontes = new ButtonGroup();
fontes.add ( policeArial);
fontes.add ( policeCourier);
fontes.add ( policeTimes);
```



JCheckBoxMenuItem : Exemple

```
JMenu menuFormat = new JMenu(" Format ");
JCheckBoxMenuItem formatGras = new JCheckBoxMenuItem(" Gras");
JCheckBoxMenuItem formatItalique = new JCheckBoxMenuItem("Italique ");
JCheckBoxMenuItem formatSouligne = new JCheckBoxMenuItem("Souligné");
menuFormat.add ( formatGras);
menuFormat.add ( formatItalique);
menuFormat.add ( formatSouligne);

mb.add(menuFichier ); mb.add(menuEdition);
mb.add(menuFonte);mb.add(menuFormat);
mb.add(menuAide);
```



Les gestionnaires de position (Layout Manager)

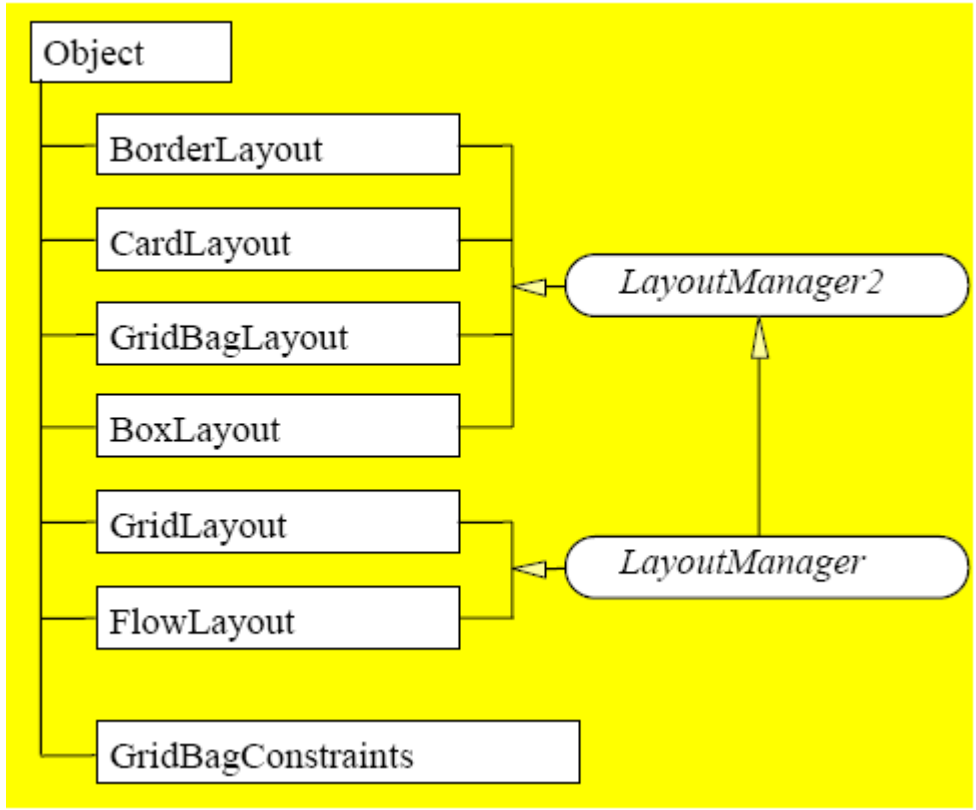
Gestionnaire de position

- Dans un **conteneur**, la disposition des **composants** (**position** et **dimension**) est déléguée à un **gestionnaire de disposition** (ou **Layout Manager**) qui se charge d'arranger les enfants à l'intérieur du conteneur.
- Il est possible de travailler sans gestionnaire de disposition et de se charger soi-même de la taille et de la position des composants (valeurs absolues en pixels). Cependant il est toujours préférable de recourir à un Layout Manager.

Les gestionnaires par défaut

- Gère la disposition des composantes filles dans un conteneur
- Les gestionnaires par défaut sont
  - **BorderLayout pour**
    - Window
    - Frame
    - Dialog
  - **FlowLayout pour**
    - Panel
    - Applet
- Pour modifier le gestionnaire de position par défaut d'un container, il suffit de faire appel à la méthode setLayout () et de lui passer en paramètre une **instance** du Layout qu'on veut appliquer.

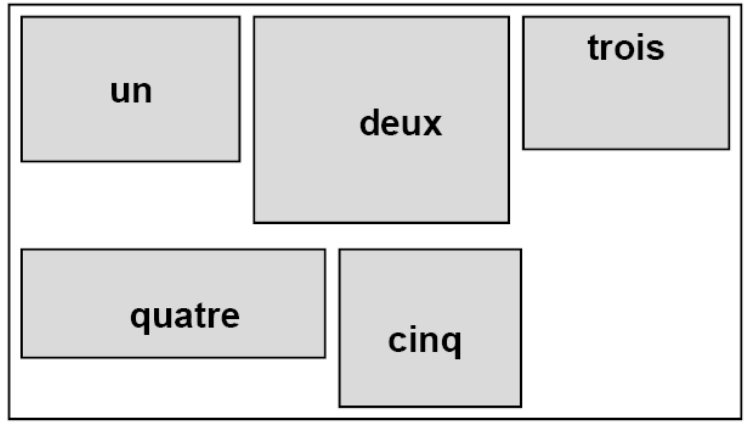
Gestionnaires de position (Layouts)



FlowLayout (java.awt)

- FlowLayout : range les composants de gauche à droite et de haut en bas.

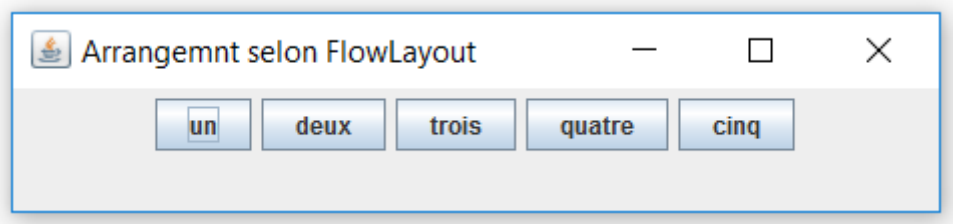
Constructeur	Rôle
FlowLayout()	range les composants en les centrant avec un "vspacing et hspacing" (espace vertival, respectivement horizontal) de 5 pixels.
FlowLayout(int aligne)	range les composants en les alignant selon aligne : FlowLayout.LEFT, FlowLayout.CENTER, FlowLayout.RIGHT avec un vspacing et hspacing de 5 pixels.
FlowLayout(int aligne, int vspacing, int hspacing)	range selon l'alignement et le vspacing et le hspacing spécifiés.



FlowLayout : Exemple

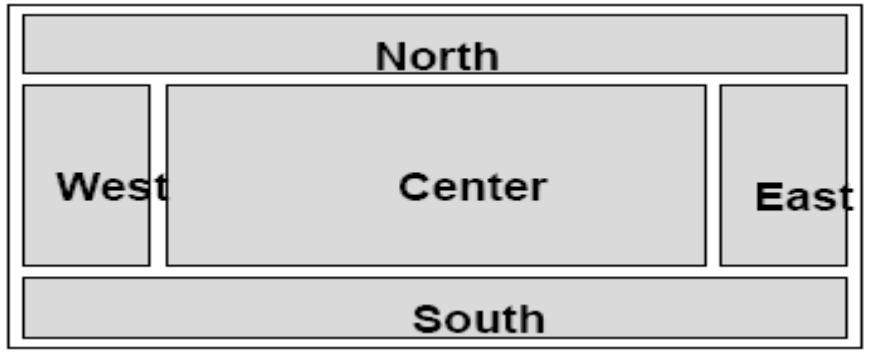
```
public class FlowLayoutExemple {
JFrame f;
JButton b1,b2,b3,b4,b5;
Container c;

public FlowLayoutExemple(){
f=new JFrame("Arrangemnt selon FlowLayout");
f.setLayout(new FlowLayout());
c=f.getContentPane();

b1=new JButton("un");
b2=new JButton("deux");
b3=new JButton("trois");
b4=new JButton("quatre");
b5=new JButton("cinq");

c.add(b1);c.add(b2);c.add(b3);
c.add(b4);c.add(b5);

f.pack();
f.setVisible(true);}
```

```
public static void main(String[] args) {
FlowLayoutExemple fle=new FlowLayoutExemple();


}
```



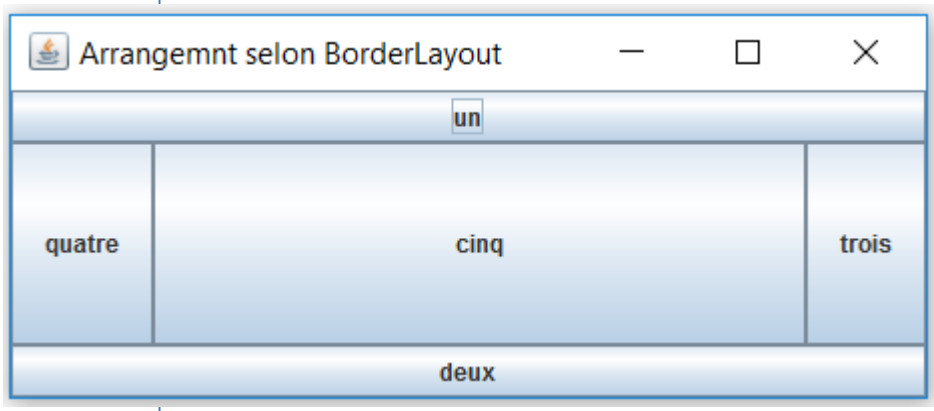
BorderLayout

- **BorderLayout :** divise un container en 5 zones : **North**, **East**, **South**, **West**, **Center**
  - BorderLayout() crée 5 zones
  - BorderLayout(int hspacing, int vspacing) idem avec un espacement spécifié.
- la méthode **add** est par exemple add ("North", composant)



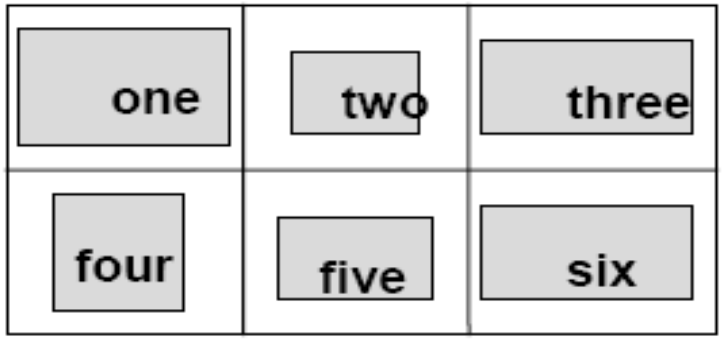
BorderLayout : Exemple

```
public class BorderLayoutExemple {
JFrame f;
JButton b1,b2,b3,b4,b5;
Container c;

public BorderLayoutExemple(){
f=new JFrame("Arrangemnt selon BorderLayout");
c=f.getContentPane();

b1=new JButton("un");
b2=new JButton("deux");
b3=new JButton("trois");
b4=new JButton("quatre");
b5=new JButton("cinq");

c.add("North",b1);c.add("South",b2);c.add("East",b3);
c.add("West",b4);c.add("Center",b5);

f.setSize(500, 200); f.setVisible(true);
}
public static void main(String[] args) {
BorderLayoutExemple ble=new BorderLayoutExemple();}
```



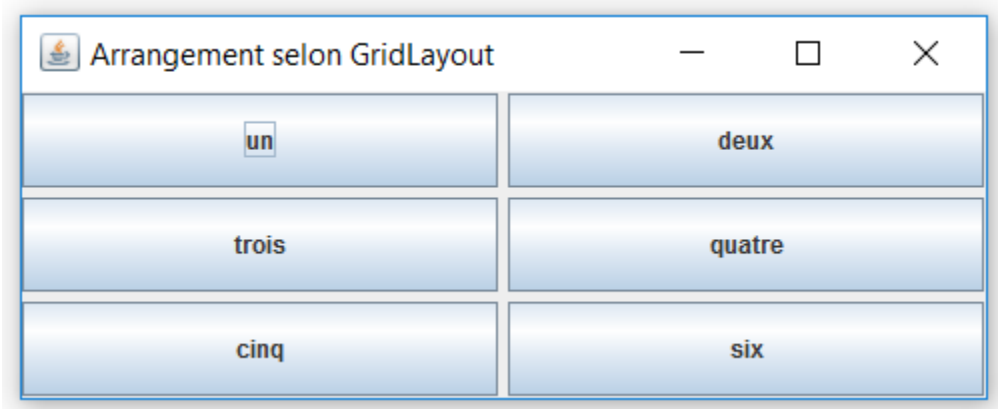
GridLayout

- **GridLayout :** range dans une grille/matrice de gauche à droite et de haut en bas
  - GridLayout(int ligne, int colonne) Les deux premiers entiers spécifient le nombre de lignes et de colonnes de la grille
  - GridLayout(int ligne, int colonne, int hspacing, int vspacing) idem avec l'espacement vertical et horizontal spécifiés.



GridLayout : Exemple

```
public class GridLayoutExemple {
JFrame f;
JButton b1,b2,b3,b4,b5,b6;
Container c;
public GridLayoutExemple(){
f=new JFrame("Arrangement selon GridLayout");
f.setLayout(new GridLayout(3,2,5,5));
c=f.getContentPane();

b1=new JButton("un");b2=new JButton("deux");
b3=new JButton("trois");b4=new JButton("quatre");
b5=new JButton("cinq");b6=new JButton("six");

c.add(b1);c.add(b2);c.add(b3);
c.add(b4);c.add(b5);c.add(b6);

f.setSize(500, 200);
f.setVisible(true);
}
public static void main(String[] args) {
GridLayoutExemple gle=new GridLayoutExemple();}
```



CardLayout

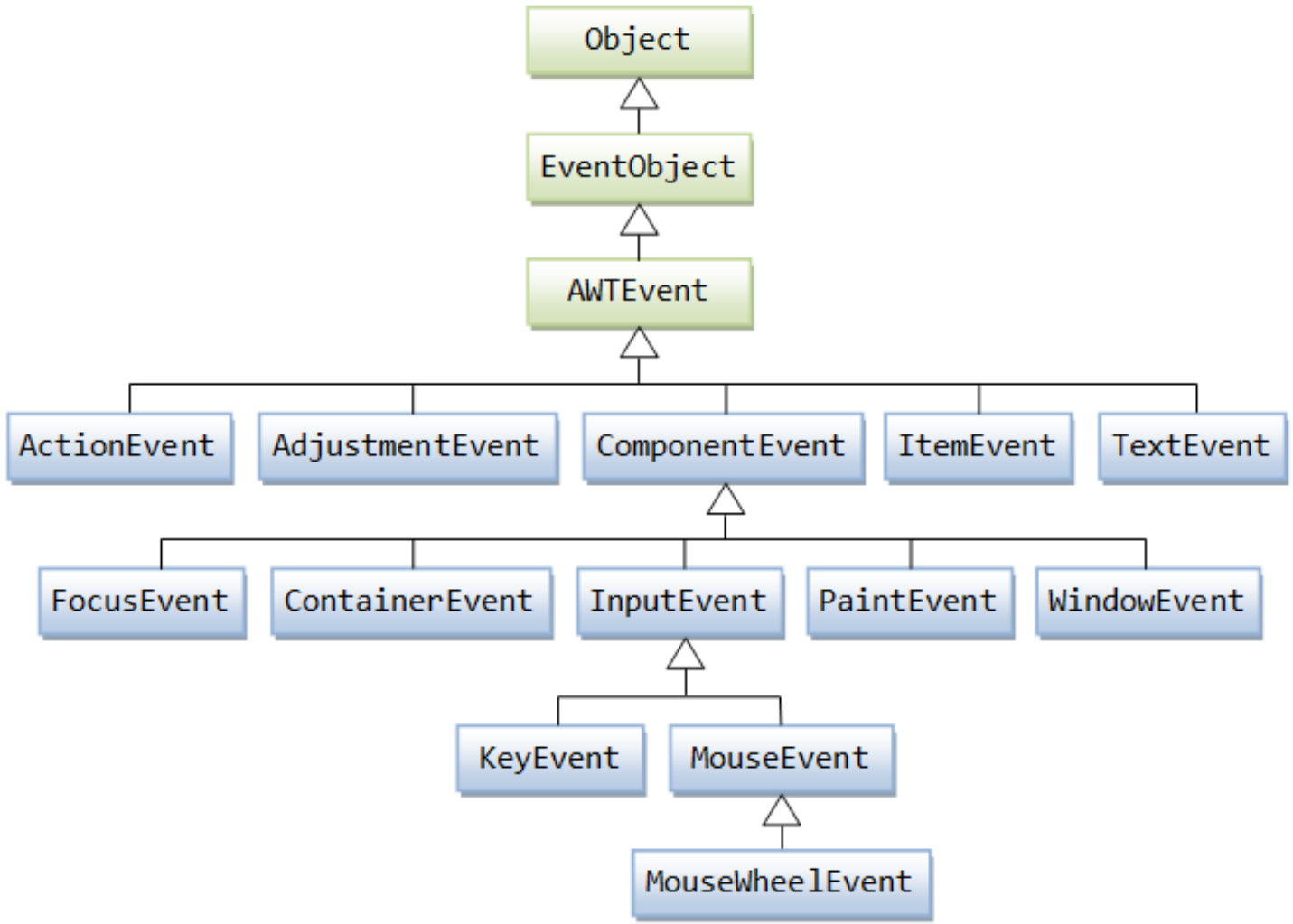
- **CardLayout :** Élargit chaque composant à la taille du conteneur et range comme une pile de cartes, il définit des objets qui ne sont pas visibles simultanément mais consécutivement.
  - Le conteneur **JTabbedPane** est généralement plus pratique pour l'utilisateur
- **Constructeurs**
  - CardLayout () ;
  - CardLayout (int , int) : Permet de préciser l'espace horizontal et vertical du tour du composant.
- Plusieurs méthodes permettent de modifier le composant actuellement affiché
  - first () Affiche le premier composant
  - last () Affiche le dernier composant
  - previous () Affiche le composant précédent
  - next () Affiche le composant suivant

Layout null

- Le layout null est utilisé pour définir des tailles et position de composants personnalisés et fixes.
- Bien qu'il existe, il n'est **pas recommandé parce que l'interface est plus difficile à** réaliser et à maintenir. De plus, elle est statique et ne tient pas compte des redimensionnements de la fenêtre.

```
pane.setLayout(null);
JButton bt = new JButton("cliquez-moi");
pane.add(bt);
bt.setLocation(50,100); // position en pixels
bt.setSize(150, 200); // taille en pixels
// ou bien bt.setBounds(50,100,150,200);
```

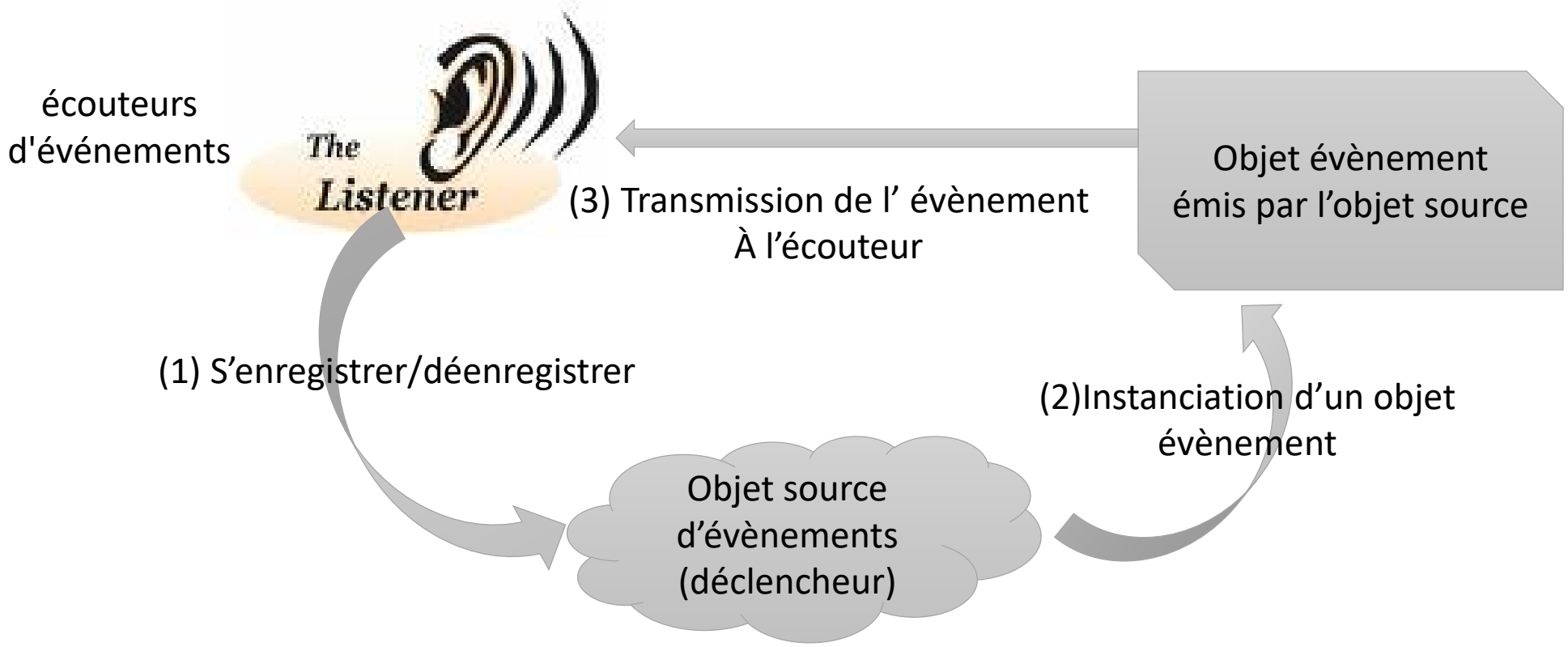
Gestionnaire d'évènements


Object
EventObject
AWTEvent
ActionEvent
AdjustmentEvent
ComponentEvent
ItemEvent
TextEvent
FocusEvent
ContainerEvent
InputEvent
PaintEvent
WindowEvent
KeyEvent
MouseEvent
MouseWheelEvent

Gestion d'évènements

- Une interface graphique doit interagir avec l'utilisateur et donc réagir à certains événements.
- La gestion des évènements obéit au modèle **Event Délégation Model :**
  - Les objets graphiques délèguent le contrôle des évènements utilisateur à des entités externes dites **Listeners (Ecouteurs)**
  - Ce modèle est requis pour des soucis d'amélioration de performances
- Les **écouteurs** d'évènements sont des **interfaces** regroupées par familles d'évènements et isolées dans ***java.awt.event.***
  - On distingue ***ActionListener, MouseListener, MouseMotionListener, WindowListener, …***

Gestion d'évènements: Mécanisme



Gestion d'évènements: Mécanisme

- Des objets **sources d'événements** (bouton, barre de défilement, …) transmettent les événements à des objets **écouteurs d'événements**.
  - Un objet **écouteur d'événements** est une instance d'une classe qui implémente une interface spéciale appelée **interface écouteur**.
  - Un objet **source d'événements** permet de recenser les objets écouteurs et de leur envoyer des **objets événements**.
  - Lorsqu'un événement se produit, la source d'événement envoie l'objet événement à tous les écouteurs recensés.

Gestion d'évènements: Mécanisme

- En général et par convention, **l'écouteur d'événement** d'un composant d'interface, est souvent **le conteneur de ce composant**.
  - Un objet écouteur (par exemple un JPanel) contient un à plusieurs objets source d'événements (par exemple des JButton)
  - L'écouteur s'enregistre auprès des sources d'événements afin de pouvoir les écouter (Le JPanel doit implémenter l'interface **ActionListener** possédant une seule méthode **actionPerformed()**)
  - Lorsqu'un évènement se produit (clic de souris), l'écouteur reçoit un objet **ActionEvent**.
  - La méthode **getSource()** (héritée de **EventObject** superclasse des classes événements) permet alors de déterminer quel est l'objet source de l'événement, ou la méthode **getActionCommand()** renvoie la chaîne de commande associée à l'action.

Programmation d'évènements sur un composant

- Pour programmer les évènements sur un composant
  - importer le groupe de classe java.awt.event :

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

  - la classe doit déclarer qu'elle utilisera une ou plusieurs interfaces d'écoute :

```
public class ApplAction extends JFrame implements ActionListener {
```

  - Pour déclarer plusieurs interfaces, il suffit de les séparer par des virgule

```
public class ApplAction extends JFrame implements ActionListener, MouseListener{
```

Programmation d'évènements sur un composant

- Choisir le type d'écouteur, et l'enregistrer sur le composant avec la méthode **addXYZListener()**

```
JButton b = new JButton( "bouton");
b.addActionListener(this);/* this indique que la classe elle même recevra et gérera
l'évènement utilisateur */
```

- Implémenter le listener XYZ en définissant **chacune** de ses méthodes.
- Pour identifier le composant qui a généré l'evénement il faut utiliser les méthodes ***getActionCommand()*** ou ***getSource()*** héritée de ***EventObject.***

```
String composant = evt.getActionCommand();// ou bien
Object source = evt.getSource();
```

Programmation d'évènements : Exemple(1)

```
import java.awt.*;
import java.awt.event.*;
public class MonApplication_ActionListener_v1 {
    public static void main(String[] args) {
        Frame f = new Frame ("Ma fenetre a moi "); //Le container
        Button b1 =new Button("D'accord ..."); //Un composant
        b1.addActionListener(new Ecouteur_b1()); //Enregistrement de l'écouteur
        Button b2 =new Button("Annuler ..."); //Un autre composant
        b2.addActionListener(new Ecouteur_b2());//Enregistrement de l'écouteur
        f.setSize(200,200);
        f.setLayout (new FlowLayout());
        f.add(b1);
        f.add(b2);
        f.setVisible(true);
}}
class Ecouteur_b1 implements ActionListener {
    public void actionPerformed(ActionEvent e){
        System.out.println(" Vous êtes d'accord ... Merci");
        }}
class Ecouteur_b2 implements ActionListener{
    public void actionPerformed(ActionEvent e){
        System.out.println(" Vous avez choisi d'annuler ... au revoir");//
        System.exit(-1);
    }
}
```

Programmation d'évènements : Exemple(2) avec 1 seul écouteur

```
import java.awt.*;
import java.awt.event.*;
public class MonApplication_ActionListener_V2 {
public static void main(String[] args) {
Frame f = new Frame ("Ma fenetre a moi "); //Le container

Button b1 =new Button("D'accord ..."); //Un composant
b1.setActionCommand("Label1");
b1.addActionListener(new Ecouteur_Bouton()); //Enregistrement de l'écouteur

Button b2 =new Button("Annuler ..."); //Un autre composant
b2.setActionCommand("Label2");
b2.addActionListener(new Ecouteur_Bouton());//Enregistrement de l'écouteur

}}
class Ecouteur_Bouton implements ActionListener {
public void actionPerformed(ActionEvent e)
{ if (e.getActionCommand() == "Label1")
System.out.println(" Vous êtes d'accord ... Merci");
else
System.out.println(" Vous avez choisi d'annuler ... Au revoir");
}}
```

les interfaces écouteurs d'événements

- Une classe qui désire recevoir des événements doit implémenter une interface écouteur. Elle se recense auprès de la source d'événement, puis elle reçoit les événements souhaités et les traite grâce aux méthodes de l'interface écouteur.
- Il y a plusieurs interfaces écouteurs dans le package java.awt.event dont voici quelques unes :

L'interface ActionListener

- ***Evènement :*** *clic, touche entrée, sélection d'un élément*
- ***Méthode*** : actionPerformed(actionEvent e)
- **Evénements générés par** : **AbstractButton**, **Button, JComboBox, JFileChooser, JTextField, List, MenuItem, TextField** avec la méthode **addActionListener**

ActionListener: Exemples

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

public class AppAction extends Frame implements ActionListener{

public  AppAction()
{
     super();
     setTitle(" Titre de la Fenetre ");
     setSize(300, 150);
     setLayout(new FlowLayout());

     Button b1 = new Button("boutton 1");
     b1.addActionListener(this);
     add(b1);

     Button b2 = new Button("boutton 2");
     b2.addActionListener(this);
     add(b2);

     Button b3 = new Button("Quitter");
     b3.addActionListener(this);
     add(b3);

     show();
}

//on doit alors implementer la methode actionPerformed
public void actionPerformed(ActionEvent evt)
{
     String composant = evt.getActionCommand();
     System.out.println("Action sur le composant : " + composant);

     if(composant.equals("Quitter")) System.exit(0);
}


public static void main(String[] args) {
AppAction a = new AppAction();
}


}
```

L'interface ItemListener

- ***Evènement :*** sélection de cases à cocher et de liste d'options.
- ***Méthode*** : Pour déterminer si une case à cocher est sélectionnée ou inactive, utiliser la méthode **getStateChange()** avec les constantes **ItemEvent.SELECTED** ou **ItemEvent.DESELECTED**
- **Evénements générés par** : **AbstractButton**, **Checkbox, CheckboxMenuItem, Choice, JComboBox, List** avec la méthode **addItemListener().**

ItemListener : Exemple(1)

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

public class AppItem extends Frame implements ItemListener{

public AppItem() {
    super();
    setTitle(" Titre de la Fenetre ");
    setSize(300, 150);
    setLayout(new FlowLayout());
    Checkbox cb = new Checkbox("choix 1", true);
    cb.addItemListener(this);
    add(cb);
    show();
}

public void itemStateChanged(ItemEvent item) {
    int status = item.getStateChange();
    if (status == ItemEvent.SELECTED)
    System.out.println("choix selectionne");
    else
    System.out.println("choix non selectionne");
}

public static void main(String[] args) {
AppItem a = new AppItem();
}

}
```

ItemListener

- Pour connaître l'objet qui a généré l'événement, il faut utiliser la méthode getItem().
- Pour déterminer la valeur sélectionnée dans une combo box, il faut utiliser la méthode getItem() et convertir la valeur en chaîne de caractères.

ItemListener : Exemple(2)

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

public class AppItemChoice extends Frame implements ItemListener{

public AppItemChoice() {
    super();
    setTitle(" Titre de la Fenetre ");
    setSize(300, 150);
    setLayout(new FlowLayout());
    Choice c = new Choice();
    c.add("choix 1");
    c.add("choix 2");
    c.add("choix 3");
    c.addItemListener(this);
    add(c);
    show();
}

public void itemStateChanged(ItemEvent item) {
    Object obj = item.getItem();
    String selection = (String)obj;
    System.out.println("choix : "+selection);
}

public static void main(String[] args) {
AppItemChoice a = new AppItemChoice();
}


}
```

L'interface WindowListener

- ***Evènement :*** *fenêtre activée, désactivée, réduite, fermée, …*
- ***Méthode*** :
  - **windowActivated**(WindowEvent e)
  - **windowClosed**(WindowEvent e)
  - **windowClosing**(WindowEvent e)
  - **windowDeactivated**(WindowEvent e)
  - **windowDeiconified**(WindowEvent e)
  - **windowIconified**(WindowEvent e)
  - **windowOpened**(WindowEvent e)
- **Evénements générés par** : **Window** avec la méthode **addWindowListener**

WindowListener : Exemple

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

public class AppWindow extends Frame implements WindowListene

public AppWindow() {
     super();
     setTitle(" Titre de la Fenetre ");
     setSize(300, 150);
     addWindowListener(this);
     show();
}

public void windowClosing(WindowEvent e)
{
System.exit(1);
}
public void windowActivated(WindowEvent e)
{
}
public void windowClosed(WindowEvent e)
{
}
public void windowDeactivated(WindowEvent e)
{
}
public void windowDeiconified(WindowEvent e)
{
}
public void windowIconified(WindowEvent e)
{
}
public void windowOpened(WindowEvent e)
{
}


public static void main(String[] args) {
AppWindow a = new AppWindow();
}

}
```

L'interface TextListener

- ***Evènement :*** modification de zone de saisie ou de texte.
- **Méthode : textValueChanged(TextEvent e)**
- **Evénements générés par** : **TextComponent** avec la méthode **addTextListener**

TextListener : Exemple

```
import java.awt.*;
import java.awt.event.*;
public class AppText  extends Frame implements TextListener{
public AppText() {
super ("Titre de la fenêtre" );
TextField t = new TextField("");
t.addTextListener(this);
add(t);
setVisible(true);
}
public void textValueChanged(TextEvent txt) {
Object source = txt.getSource();
System.out.println("saisi = "+((TextField)source).getText());
}
public static void main(String[]args){
  AppText at=new AppText();
}
}
```

L'interface **MouseListener**

- ***Evènement :*** *clic sur bouton, déplacement du pointeur*
- **Méthode :**
  - **mouseClicked**(MouseEvent e)
  - **mouseEntered**(MouseEvent e)
  - **mouseExited**(MouseEvent e)
  - **mousePressed**(MouseEvent e)
  - **mouseReleased**(MouseEvent e)
- Evénements générés par : **Component** avec la méthode **addMouseListener**

L'interface **MouseMotionListener**

- ***Evènement :*** *événements de glisser-déplacé*
- **Méthode :**
  - **mouseDragged**(MouseEvent e)
  - **mouseMoved**(MouseEvent e)

- Evénements générés par : **Component** avec la méthode **addMouseMotionListener**

L'interface AdjustementListener

- ***Evènement :*** *déplacement du curseur d'une barre de défilement*
- **Méthode : adjustmentValueChanged(AdjustmentEvent e)**
- Evénements générés par : **Scrollbar, JScrollbar** avec la méthode **addAdjustementListener**

ComponentListener

- ***Evènement :*** *déplacement, affichage, masquage ou modification de taille de composants*
- **Méthode :**
  - **componentHidden**(ComponentEvent
  - **componentMoved(ComponentEvent e)**
  - **componentResized(ComponentEvent e)**
  - **componentShown(ComponentEvent e)**
- Evénements générés par : **Component** avec la méthode **addComponentListener**

ContainerListener

- ***Evènement :*** *ajout ou suppression d'un composant dans un conteneur*
- **Méthode :**
  - **componentAdded**(ContainerEvent e)
  - **componentRemoved**(ContainerEvent e)
- Evénements générés par : **Container** avec la méthode **addContainerListener**

FocusListener

- ***Evènement :*** *obtention ou perte du focus par un composant*
- **Méthode :**
  - **focusGained**(FocusEvent e)
  - **focusLost**(FocusEvent e)

- Evénements générés par : **Component** avec la méthode **addFocusListener**

Les Adaptateurs

- L'implémentation d'une interface nécessite de développer toutes ses méthodes, ce qui peut être considéré onéreux.
- Afin de réconforter les développeurs, **java.awt.event** a prévu un ensemble de classes, dites adaptateurs, qui implémentent d'ores et déjà, toutes les méthodes des interfaces avec un comportement vide.
- Ainsi, au lieu d'implémenter des interfaces, la programmation consistera à étendre les adaptateurs et à ne redéfinir que les méthodes souhaitées.
- L'interface **ActionListener** n'a pas d'adaptateur étant donné qu'elle ne contient qu'une unique méthode **actionPerformed()**

Les Adaptateurs

- Ces adaptateurs sont les suivants :
  - ComponentAdapter
  - ContainerAdapter
  - FocusAdapter
  - KeyAdapter
  - MouseAdapter
  - MouseMotionAdapter
  - WindowAdapter

Exemple avec WindowAdapter

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

public class MonApplication_windwoListener_V2 {
 public static void main(String[] args) {
   Frame f = new Frame ("Ma fenetre a moi "); //Le container
  f.addWindowListener(new Ecouteur_Fenetre()); //Enregistrement de l'écouteur
   ...

 }}
class Ecouteur_Fenetre extends WindowAdapter
{
   // On ne redéfint que la méthode de la fermeture de la fenêtre
   public void windowClosing(WindowEvent e) {
       System.out.println(" Au revoir !!!");
       System.exit(-1);
   }
}
```

Conteneurs secondaires

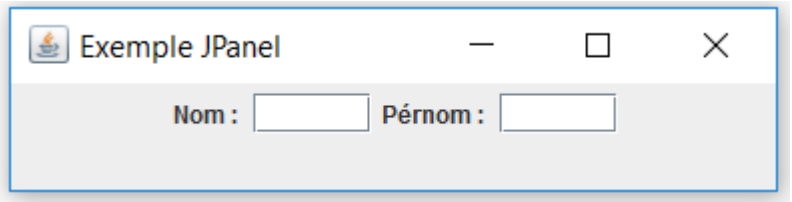
Le panneau : JPanel

- Le conteneur léger le plus simple de Swing et le panneau (JPanel),
- Il permet de grouper des composants selon le gestionnaire de position FlowLayout,
- Pour ajouter un composant à un panneau, on utilise la méthode add.
- **constructeurs** :
  - Panel() crée un JPanel
  - Panel(LayoutManager manager) crée un JPanel avec le layout manager spécifié

JPanel : Exemple

```
public class PanelExemple extends JFrame {

private JTextField nom;
private JLabel labelNom ;
private JTextField prenom ;
private JLabel labelPrenom;
JPanel pan;

public PanelExemple(){
super ("Exemple JPanel ");
Container c=this.getContentPane();
pan=new JPanel();

labelNom = new JLabel (" Nom : ");
labelPrenom = new JLabel ("Pérnom : ");
nom = new JTextField(5);
prenom = new JTextField(5);

pan.add(labelNom);
pan.add(nom);
pan.add(labelPrenom);
pan.add(prenom);
```

```
c.add(pan);
this. setSize(400,100);
this. setVisible( true);
}
public static void main(String[] args) {
PanelExemple cle=new PanelExemple();

}}
```



Le panneau de défilement : JScrollPane

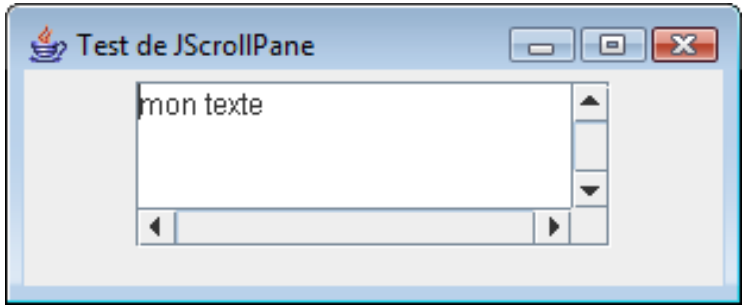
- JScrollPane est un conteneur permettant de munir un composant de barres de défilement.
- Les *JScrollBars* sont munies d'une stratégie d'affichage qui peut être :
  - *VERTICAL_SCROLLBAR_AS_NEEDED* la *ScrollBar* verticale n'est visible que si elle est nécessaire.
  - *VERTICAL_SCROLLBAR_NEVER* la *ScrollBar* verticale n'est jamais visible
  - *VERTICAL_SCROLLBAR_ALWAYS* la *ScrollBar* verticale est toujours visible
  - *HORIZONTAL_SCROLLBAR_AS_NEEDED* la *ScrollBar* horizontale n'est visible que si elle est nécessaire.
  - *HORIZONTAL_SCROLLBAR_NEVER* la *ScrollBar* horizontale n'est jamais visible
  - *HORIZONTAL_SCROLLBAR_ALWAYS* la *ScrollBar* horizontale est toujours visible

JScrollPane : Constructeurs

- Constructeurs
  - JScrollPane() Crée un JScrollPane sans composant vue, avec des politique horizontale et verticale AS_NEEDED
  - JScrollPane(Component vue) Crée un JScrollPane avec composant vue, avec des politique horizontale et verticale AS_NEEDED
  - JScrollPane(Component vue, int vPolitique, int hPolitique) Crée un JScrollPane avec composant vue, une politique horizontale et une politique verticale
  - JScrollPane(int vPolitique, int hPolitique) Crée un JScrollPane sans composant vue, mais avec une politique horizontale et une politique verticale

JScrollPane : Exemple

```
public class MonScrollPane {

public static void main(String[] args) {
  JFrame f = new JFrame("Test de JScrollPane");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();
    JTextArea  textArea1 = new JTextArea ("mon texte");

 JScrollPane scrollPane = new JScrollPane(textArea1,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
    scrollPane.setPreferredSize(new Dimension(200,70));

    pannel.add(scrollPane);
    f.getContentPane().add(pannel);
    f.setVisible(true);
}
}
```



Le panneau divisé : JSplitPane

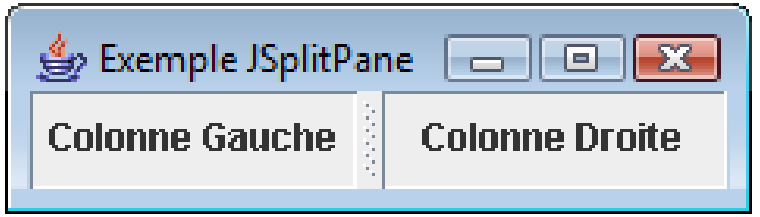
- Un JSplitPane permet d'afficher deux composants séparés verticalement ou horizontalement.
- La barre de division qui apparait entre les deux composants peut être déplacée.
- L'orientation du JSplitPane peut être :
  - JSplitPane.HORIZONTAL_SPLIT : les deux composants sont alignés horizontalement
  - jSplitPane.VERTICAL_SPLIT : les deux composants sont alignés verticalement

JSplitPane : Méthodes

- **Constructeurs**
  - JSplitPane() Crée un JSplitPane horizontal et à affichage non continu.
  - JSplitPane(int orientation) Crée un JSplitPane orienté suivant orientation et à affichage non continu.
  - JSplitPane(int orientation, Component cGauche, Component cDroit) Crée un JSplitPane orienté suivant orientation et à affichage non continu. Les deux composants sont cGauche et cDroit.
- **Méthodes** :
  - void setOrientation(int o) Affecte l'orientation du JSplitPane

JSplitPane : Exemple

```
public class MonSplitPane{

public static void main(String[] args) {
JFrame f = new JFrame("Exemple JSplitPane");
JPanel p1=new JPanel();
JPanel p2=new JPanel();
p1.add(new JLabel("Colonne Gauche"));
p2.add(new JLabel("Colonne Droite"));

JSplitPane jsp=new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT,p1,p2);
f.getContentPane().add(jsp);
    f.pack();
    f.setVisible(true);
}
}
```



Le panneau à onglets : JTabbedPane

- Le composant JTabbedPane permet de construire des interfaces en utilisant des onglets.
- Les composants peuvent ainsi être regroupes de manière thématique pour obtenir des interfaces allégées.
- Chaque onglet est constitué d'un titre, d'un composant et éventuellement d'une image.
- Pour utiliser ce composant, il faut :
  - instancier un objet de type JTabbedPane
  - créer le composant de chaque onglet
  - ajouter chaque onglet à l'objet ***JTabbedPane*** en utilisant la méthode ***addTab()***

JTabbedPane : Constructeurs

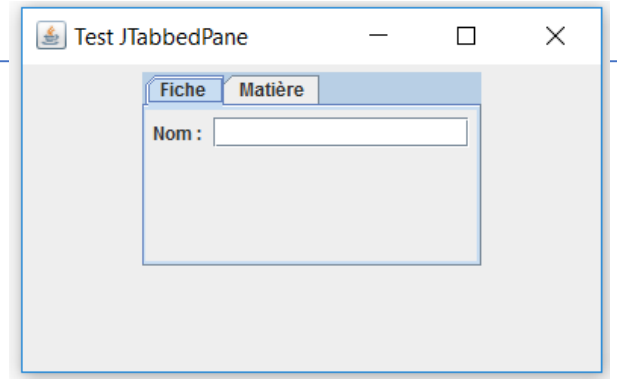
- Constructeurs :
  - JTabbedPane() Crée un panneau à onglets, les onglets sont placés en haut.
  - JTabbedPane(**int** tabPlacement) Crée un panneau à onglets, les onglets sont placés :
    - en haut si *tabPlacement* vaut *JTabbedPane.TOP*
    - en bas si *tabPlacement* vaut *JTabbedPane.BOTTOM*
    - à gauche si *tabPlacement* vaut *JTabbedPane.LEFT*
    - à droite si *tabPlacement* vaut *JTabbedPane.RIGHT*
  - JTabbedPane(**int** tabPlacement, **int** tabLayoutPolicy) La stratégie de placement (quand ils ne tiennent pas sur une ligne) des onglets est :
    - *JTabbedPane.WRAP_TAB_LAYOUT* : Les onglets passent à la ligne.
    - *JTabbedPane.SCROLL_TAB_LAYOUT* : une barre de défilement apparait.

JTabbedPane : méthodes

- addTab(String, Component) Permet d'ajouter un nouvel onglet dont le titre et le composant sont fournis en paramètres. Cette méthode possède plusieurs surcharges qui permettent de préciser une icône et une bulle d'aide.
- insertTab(String, Icon, Component, String, index) Permet d'insérer un onglet dont la position est précisée dans le dernier paramètre
- remove(int) Permet de supprimer l'onglet dont l'index est fourni en paramètre
- setTabPlacement Permet de préciser le positionnement des onglets dans le composant JTabbedPane. Les valeurs possibles sont les constantes TOP, BOTTOM, LEFT et RIGHT définies dans la classe JTabbedPane.

JTabbedPane : Exemple

```
public class JTabbedPaneExemple {

public static void main(String[] args) {
JFrame f = new JFrame("Test JTabbedPane");
    f.setSize(400, 250);
    JPanel pannel = new JPanel();

JTabbedPane onglets = new
JTabbedPane(JTabbedPane.TOP);

JPanel p1 = new JPanel();
p1.add(new JLabel("Nom : "));
p1.add(new JTextField(15));
    onglets.addTab("Fiche", p1);
```




```
JPanel p2 = new JPanel();
    p2.add(new JLabel("Liste des matières"));
    String [] tabMat={" Math " , " Info " , " Droit " ,
"Français " , " Sciences"};
    JList l = new JList<>(tabMat);
    p2.add(l);

    onglets.addTab("Matière", p2);

    onglets.setOpaque(true);
    pannel.add(onglets);
    f.getContentPane().add(pannel);
    f.setVisible(true);  }
}
```