Chapitre 4 : Les composants

- Composants Swing
- Principaux composants: boutons, menus, barre de menus, composants textes, tables et listes
- Les afficheurs associés aux conteneurs

Les composants Swing

`JComponent` hérite de `Component` et de `Container`.

La classe fournit en outre un support pour :

- ✓ *A pluggable look and feel* (plaf).
- ✓ des infos bulles ou *tooltips* (classe `JTooltip`).
- ✓ le scrolling automatique des listes, tables, arbres (via l'interface `Scrollable`).
- ✓ le débugage des graphiques (avec la classe `DebugGraphics`).

Les composants Swing

Nouvelles caractéristiques des `JComponent` :

>*Double buffering*.

>Accessibilité (aux handicapés).

>Bordures : il y a 8 classes de bords.

>Gestion des touches clavier.

>Propriétés des composants (paires de nom/valeur (=objet)) : peuvent être associées à n'importe quel composant Swing; - >permettent d'associer un composant swing avec un autre objet sans étendre la classe.

Les composants Swing

> Composants de base: *des boutons*, *champs de textes, menus,* `JComboBox, JList, JSlider`.

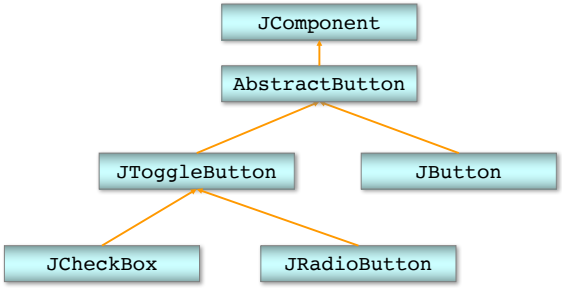
> Composants non modifiables : `JLabel, JProgressBar, JToolTip.`

> Composants modifiables : *plusieurs éditeurs de textes,* `JColorChooser, JFileChooser, JTree`.

Les composants Swing

> Cadres de haut niveau: `JFrame, JApplet, JDialog.`
> Conteneurs généraux: `JPanel, JScrollPane, JSplitPane, JTabbedPane, JToolBar.`
> Conteneurs spécialisés: `JInternalFrame, JLayeredPane, JRootPane.`

Boutons

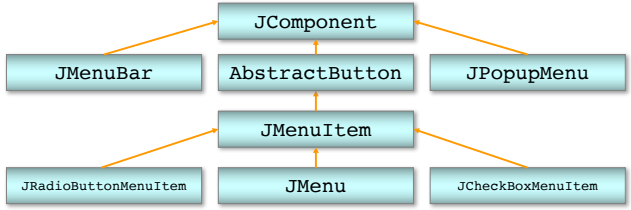


Boutons

> `JButton` définit un bouton standard.
> `JToggleButton` définit un bouton à deux états. Il existe en deux versions : `JCheckbox` et `JRadioButton` .
> Les *toggle buttons* peuvent être regroupés dans un même `ButtonGroup` de sorte qu'un seul élément du groupe puisse être simultanément sélectionné.
> La classe `JToolBar` permet de créer une barre d'outils contenant divers boutons.

Menus

Swing supporte des menus contextuels (menus pop-up) et des barres de menus



Menus

> `JMenuBar` : barre de menus d'une fenêtre d'application.
> Un objet `JMenu` est une option de niveau supérieur dans une barre de menus qui fait dérouler une liste d'options quand on clique dessus. Les options de tels menus sont définies par la classe `JMenuItem.`

Menus

> `JPopupMenu` : c'est un menu contextuel (habituellement implémenté pour apparaître à la position courante du curseur quand le bouton droit de la souris est enfoncé).
> `JCheckBoxMenuItem` : option de menu dotée d'une case à cocher quand l'option est sélectionnée.

Ajout de menus

> `JMenuBar:` implémente la barre de menus placée en haut d'une fenêtre
> On ajoute des objets `JMenu` ou `JMenuItem` dans une `JMenuBar` avec la méthode `add` et ces objets apparaîtront dans la barre de menus

Menu et item de menu

> Un objet `JMenu` possède un label, et quand on clique dessus, il déroule un menu d'items.
> Un objet `JMenuItem` est un simple item de menu visualisé par un label. Quand on clique dessus, il peut invoquer un programme. Un `JMenuItem` peut également afficher un icône en sus de son label.

Menu et item de menu

> Les items d'un `JMenu` (menu déroulant d'une barre de menu) peuvent être de simples `JMenuItem`, `JCheckBoxMenuItem`, `JRadioButtonMenuItem`, ou eux-mêmes des `JMenu` (sous-menus).

> On les ajoute au `JMenu` avec la méthode `add`.

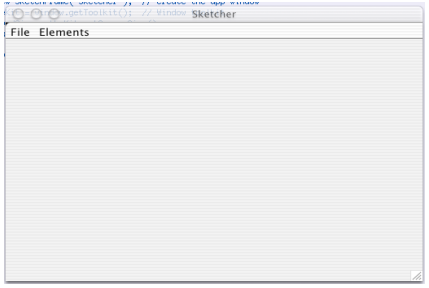
Créer une barre de menus

```
import java.awt.*;      // extrait du livre de Ivor Norton
                        // "Maitrisez Java 2"
Public class Sketcher {
  static SketcherFrame window;     /la fenêtre cadre d'application
  public static void main(String[] args) {
        window = new SketcherFrame(«Sketcher»);
        Toolkit leKit = window.getToolkit();
        Dimension wndSize = leKit.getScreenSize();
        window.setBounds(wndSize.width/4, wndSize.height/4,
                wndSize.width/2, wndSize.height/2);
        window.setVisible(true);
  }
}
```


Créer une barre de menus

```
import javax.swing.*;
Public class SketcherFrame extends JFrame {
   private JMenuBar menuBar = new JMenuBar();
   public SketcherFrame (String titre) {
        setTitle(titre);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setJMenuBar (menuBar);
        JMenu fileMenu = new JMenu(«File»);
        JMenu elementMenu = new JMenu («Elements»);
        menuBar.add(fileMenu);
        menuBar.add(elementMenu);
   }
}
```


Créer une barre de menus



Ajouter un raccourci

> Un raccourci (ou mnémonique) est une combinaison de clés utilisée pour sélectionner un menu dans la barre de menus. *Sous Windows: ALT+une lettre du label du menu.*

```
fileMenu.setMnemonic('F');
```

La lettre F du titre File apparait alors soulignée.

> Un accélérateur est une combinaison de touches utilisée pour sélectionner une option dans un menu. *Sous Windows: Ctrl+une lettre.*

Ajouter un accélérateur

```
lineItem.setAccelerator(KeyStroke.getKeyStroke
                        ('L',Event.CTRL_MASK));
```

La classe `KeyStroke` définit une combinaison de touches. `getKeyStroke` retourne l'objet correspondant (il combine ses arguments).

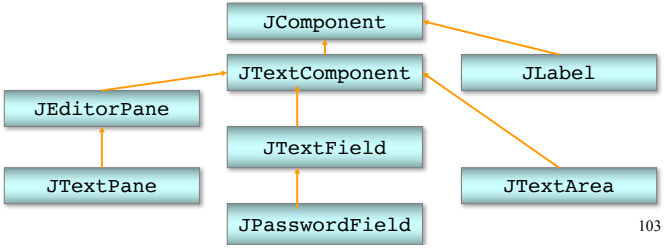
La classe `Event` définit les constantes `SHIFT_MASK`, `ALT_MASK` et `CTRL_MASK`.

Pour combiner les touches Alt et Ctrl, on peut utiliser

```
Event.ALT_MASK + Event.CTRL_MASK
```


Composants textes

Il existe une grande variété de composants textes dans Swing



Composants textes

> `JLabel` : le plus simple. Complètement passif, non éditable. je suis un label

> `JTextField` : une seul champ de texte, éditable.

> `JTextArea` : plusieurs lignes de texte, éditable :

bla bla bla
bla bla
bla bla

Les scrollbars peuvent être gérées automatiquement grâce à un JScrollPane

Composants textes

> `JPasswordField` permet de lire un mot de passe (qui sera masqué pour l'utilisateur).
> `JEditorPane` et `JTextPane` : composants plus complexes permettant d'implanter des fonctions d'édition élaborées comme celles prévues pour les formats html ou rtf.
> `JTextPane` permet d'inclure des images au sein des éléments de textes.

Autres composants utiles

> `JList` : une liste d'éléments. Sert à sélectionner des éléments. On peut préciser le mode de sélection.
> `JTable` : gère un tableau d'éléments en lignes et colonnes. On peut sélectionner une ligne, une colonne ou un élément.
> `JPanel` : (conteneur) sert à regrouper des composants, par exemple des boutons. On en utilise fréquemment plusieurs.

Conteneurs et afficheurs associés

Exemples de conteneurs

L'exemple type de conteneur est le `JPanel`. C'est un simple panneau, auquel on peut ajouter des éléments.
Un autre exemple est la `Box`, qui permettra d'ajouter des éléments horizontalement ou verticalement. A chaque conteneur correspond un afficheur. L'afficheur par défaut associé à une `Box` est un `BoxLayout`. Celui associé à un `JPanel` est un `FlowLayout`.

Rappel

La disposition des objets inclus dans un conteneur est assurée par un autre objet qui implémente l'interface

LayoutManager

Cet objet gère l'affichage des objets du conteneur. C'est un

Afficheur

Accès aux composants d'un conteneur

Les composants ajoutés à un conteneur sont stockés dans un tableau `Component` associé au conteneur. On peut y accéder avec :

- `int getComponentCount()`
- `Component getComponent(int index)`
- `Component[] getComponents()`

Exemple

Supposons que l'on ait un objet de type `Container` (par exemple le panneau d'un `JFrame`), nommé `content`. On peut écrire :

```
Component aComponent = null;

int numComponents =

content.getComponentCount();

for(int i = 0; i < numComponents; i++ ) {
      aComponent = content.getComponent(i);
// faire quelque chose avec ce composant...
}
```


Ajouts d'éléments

Pour ajouter un composant au conteneur, on utilise une version de `add qui peut varier selon l'afficheur associé`

- `Component add (Component c)`
- `Component add(Component c, int index)` (si l'index vaut -1, ajoute c à la fin)
- `void add(Component c, int constante)` (par exemple, ajout à un conteneur géré par un BorderLayout)
- `void add(Component c, Object constraints, int index)`, ici ajout d'un composant à un conteneur géré par un GridBagLayout

Ajouts d'éléments

Pour ajouter un composant au conteneur, on utilise une version de `add`. Ici, quand le conteneur est géré par un GridBagLayout on aura un argument de type GridBagConstraints

> `void add(Component c, Object constraints)` (ajoute c à la fin du tableau, et sa disposition sera soumise aux contraintes spécifiées par l'objet constraints pour l'afficheur associé)
> `void add(Component c, Object constraints, int index)`

interface LayoutManager

> Un afficheur est un objet qui implémente l'interface `LayoutManager.`
> Il détermine la mise en page (position et taille) des composants à l'intérieur du conteneur, lors de l'affichage ou lors d'un changement de taille.
> Il y a plusieurs afficheurs dans awt et swing, et on peut en définir d'autres.
> Il y en a 6 principaux.

Afficheurs

> `FlowLayout` : place les objets sur des lignes successives comme le ferait un traitement de texte. Souvent utilisé pour des boutons. C'est l'afficheur par défaut associé aux `JPanel.`
> `BorderLayout` : place les objets contre les bords ou au centre. C'est l'afficheur par défaut du panneau de contenu des `JFrame,JDialog` et `JApplet.`
> `CardLayout` : place les objets les uns sur les autres. Seul le sommet de pile est visible à un moment donné.

Afficheurs

> `GridLayout` : place les objets sur une grille dont on peut spécifier le nombre de lignes et de colonnes.
> `GridBagLayout` : place également les éléments sur une grille, mais la hauteur des lignes et la largeur des colonnes peut varier. Assez complexe d'utilisation, il offre cependant une grande souplesse de contrôle.
> `BoxLayout` : dispose les composants soit en ligne, soit en colonne. C'est l'afficheur associé à la classe Box.

Afficheurs

> `BoxLayout` [et `Box`] sont définis dans swing. Les autres dans awt.

> Pour modifier l'afficheur associé à un conteneur, on utilise `SetLayout()`.

Exemple: pour changer la mise en page du panneau de contenu d'un `JFrame`

```
JFrame myWindow = new JFrame("Ma fenêtre");
FlowLayout flow = new FlowLayout();
myWindow.getContentPane().setLayout(flow);
```


Afficheur `FlowLayout`

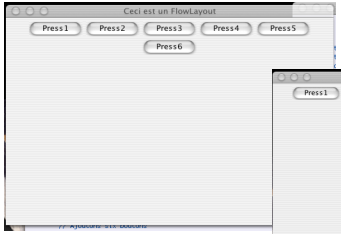
```
import javax.swing.*;
import java.awt.*;

Public class TryFlowLayout {
  static JFrame awindow= new JFrame("Ceci est un FlowLayout");
  public static void main(String[] args) {

        Toolkit leKit = awindow.getToolkit();
        Dimension wndSize = leKit.getScreenSize();
        awindow.setBounds(wndSize.width/4, wndSize.height/4,
                                  wndSize.width/2, wndSize.height/2);
        awindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```


Afficheur `FlowLayout`

```
        FlowLayout flow = new FlowLayout();
        Container content = awindow.getContentPane();
        content.setLayout(flow);

        // Ajoutons six boutons
        for ( int i=1; i<= 6 ; i++ )
                content.add( new JButton ( " Press " + i );

        awindow.setVisible(true);
  }
}
```


Afficheur `FlowLayout`

Centré par défaut :



Avec changement de la taille de la fenêtre :

Afficheur `FlowLayout`

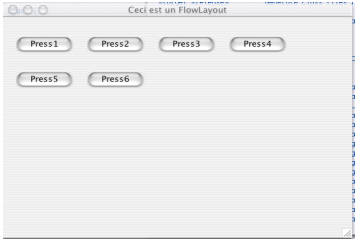
avec

```
FlowLayout flow = new FlowLayout(FlowLayout.LEFT);
```

Afficheur `FlowLayout`

Par défaut, espacement de 5 pixels entre les composants d'une ligne, ou colonne. Pour le changer :

```
FlowLayout flow = new FlowLayout(FlowLayout.LEFT,20,30);
```



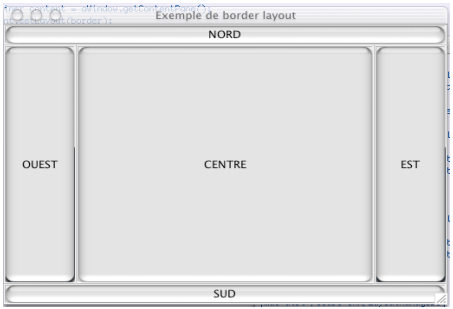
Essai avec une applet

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

public class TryApplet extends JApplet {

    public void init() {
        Container content = getContentPane();
        content.setLayout( new FlowLayout(FlowLayout.RIGHT));

        JButton button;
        Font[] fonts = { new Font("Arial", Font.ITALIC, 10),
                new Font("PlayBill", Font.PLAIN, 14)
        };

        BevelBorder edge = new BevelBorder(BevelBorder.RAISED);  // Bevelled border
        // Add the buttons using alternate fonts

        for( int i = 1; i <= 6; i++ ) {
            content.add( button = new JButton("Press "+i));   // Ajouter le bouton
                        button.setFont(fonts[i%2]);      // Alterner les fontes
                        button.setBorder(edge);      // bord du bouton
        }

    }
}
```

Afficheur `BorderLayout`

```
import javax.swing.border.* ;          // pour les bords des boutons
// …le reste identique à ce qui précède …
BorderLayout border = new BorderLayout();
Container content = awindow.getContentPane();
content.setLayout(border);
EtchedBorder edge = new EtchedBorder(EtchedBorder.RAISED);
// Ajoutons cinq boutons: un à chaque bord + un au centre
JButton button;
content.add (button = new JButton("EST") , BorderLayout.EAST);
button.setBorder(edge);         // bord du bouton
content.add (button = new JButton("OUEST"), BorderLayout.WEST);
button.setBorder(edge); // etc. avec NORTH, SOUTH et CENTER
```

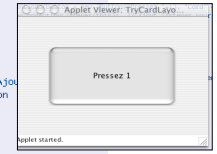
Afficheur BorderLayout



Afficheur CardLayout

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;   //classes pour gerer les evenements

public class TryCardLayout extends JApplet implements ActionListener {

    CardLayout card = new CardLayout(50,50);

    public void init() {

        Container content = getContentPane();
        content.setLayout(card);
        JButton button;
        for(int i = 1; i <= 6; i++ ) {
            content.add(button = new JButton("Pressez "+i), "Card" +i); // Ajou
            button.addActionListener(this);  // Ajout listener pour le bouton

        }
    }

    // Handle button events

    public void actionPerformed(ActionEvent e) { card.next(getContentPane()); }
}
```

espaces



Afficheur CardLayout

> L'afficheur `CardLayout` gère une pile de composants, comme un paquet de cartes. Le premier élément ajouté est mis en haut, et le dernier en bas. Seul le sommet est rendu visible.

> On peut préciser des espaces (*gaps*) au constructeur du CardLayout : distances entre le bord du composant et le bord du conteneur.

Afficheur CardLayout

> Remarquez les deux arguments passés à la méthode `add` : le second identifie le composant par un `Object`, ici une `String`.

```
> void show(Container parent,
                String name)
// permet alors de sélectionner le composant
// par la chaîne passée en argument à add.
> void previous (Container parent)
  // sélectionne le précédent
```

> même chose avec `last, next, first`

Afficheur `GridLayout`

Il dispose les objets sur une grille rectangulaire. Il y a trois constructeurs :

> `GridLayout()` // 1 seule ligne

> `GridLayout( int nrows, int ncols)`

> `GridLayout(int nrows, int ncols, int hgap, int vgap)`

// précise les espacements (*gaps*)
// entre les lignes et les colonnes

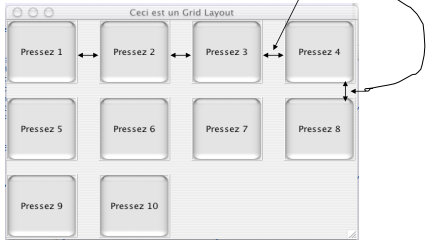
Afficheur `GridLayout`

Avec les deux derniers constructeurs, on peut mettre le nombre de lignes ou le nombre de colonnes à 0 (ou exclusif).

Si l'un des deux est nul, le `GridLayout` fournit le nombre de lignes ou de colonnes nécessaire selon ce qui est ajouté.

Afficheur `GridLayout`

Avec
GridLayout grid = New GridLayout(3, 4, 30, 20) ;



Afficheur `BoxLayout`

Il organise les composants sur une seule ligne ou une seule colonne (ex: `JMenuBar` utilise un `BoxLayout`. C'est aussi l'afficheur de la classe Box).

> Les composants sont ajoutés de gauche à droite pour une ligne, de haut en bas pour une colonne.

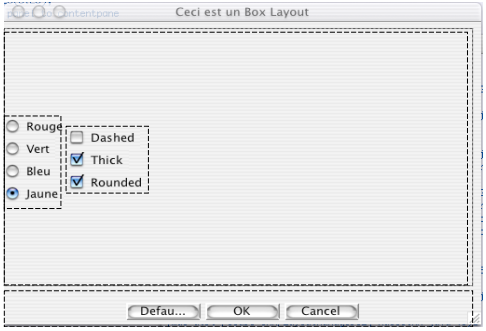
> Lorsque la ligne ou la colonne est pleine, on ne déborde pas, mais il y a réduction ou masquage.

Afficheur `BoxLayout`

- Le gestionnaire `BoxLayout` essaie de donner la même hauteur aux composants d'une ligne, et la même largeur aux composants d'une colonne.
- La classe `Box` (sous-classe de `Container`) a un `BoxLayout` pour afficheur. C'est une classe très utile : elle est d'un maniement plus simple que `JPanel` et elle possède des constructeurs intéressants (en particulier pour gérer les espaces).


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

public class TryBoxLayout {
    static JFrame aWindow = new JFrame("Ceci est un Box Layout");

    public static void main(String[] args) {
        Toolkit theKit = aWindow.getToolkit();
        Dimension wndSize = theKit.getScreenSize();
        aWindow.setBounds(wndSize.width/4, wndSize.height/4,
                          wndSize.width/2, wndSize.height/2);
        aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Creation d'une colonne de radioboutons a gauche
        Box left = Box.createVerticalBox();
        ButtonGroup radioGroup = new ButtonGroup();
        JRadioButton rbutton;
        radioGroup.add( rbutton = new JRadioButton("Rouge"));
        left.add(rbutton);
        radioGroup.add( rbutton = new JRadioButton("Vert"));
        left.add(rbutton);
        radioGroup.add( rbutton = new JRadioButton("Bleu"));
        left.add(rbutton);
        radioGroup.add( rbutton = new JRadioButton("Jaune"));
        left.add(rbutton);
```

La classe `Box`




```
// Creation de la colonne de droite de checkboxes
Box right = Box.createVerticalBox();
right.add(new JCheckBox("Dashed"));
right.add(new JCheckBox("Thick"));
right.add(new JCheckBox("Rounded"));
// Creer la ligne du haut contenant left et right
Box top = Box.createHorizontalBox();
top.add(left);
top.add(right);
// Creation d'une ligne de boutons en bas
JPanel bottomPanel = new JPanel();
Border edge = BorderFactory.createRaisedBevelBorder();
JButton button;
Dimension size = new Dimension(80,20);
bottomPanel.add(button = new JButton("Defaults"));
button.setBorder(edge);
button.setPreferredSize(size);
bottomPanel.add(button = new JButton("OK"));
button.setBorder(edge);
button.setPreferredSize(size);
bottomPanel.add(button = new JButton("Cancel"));
button.setBorder(edge);
button.setPreferredSize(size);
// Add top and bottom panel to contentpane
Container content = aWindow.getContentPane();
content.setLayout(new BorderLayout());
content.add(top, BorderLayout.CENTER);
content.add(bottomPanel, BorderLayout.SOUTH);
```

La classe `Box`

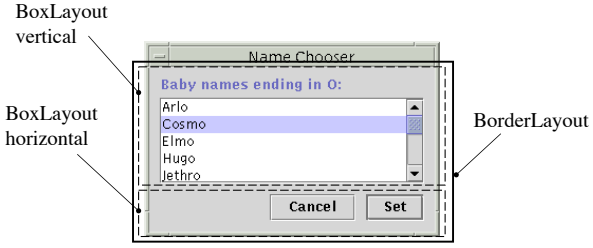
Pour améliorer la disposition des éléments on peut :

- ✓ régler les espaces entre les éléments : composants `Strut` et `Glue` (pour avoir un menu Aide à droite dans une barre de menus par exemple), `Filler` et `Rigid Area.`
- ✓ enjoliver les cadres : utiliser des `JPanel` pour contenir les `Box,` et leur mettre des bords (avec titres ou autres).

Exemple avec 2 BoxLayout

```
JScrollPane listScroller = new JScrollPane(list);
listScroller.setPreferredSize(new Dimension(250, 80));
listScroller.setMinimumSize(new Dimension(250, 80));
listScroller.setAlignmentX(LEFT_ALIGNMENT);
  ...
JPanel listPane = new JPanel();          // avec BoxLayout vertical
listPane.setLayout(new BoxLayout(listPane,
                                 BoxLayout.Y_AXIS));
JLabel label = new JLabel(labelText);
listPane.add(label);
listPane.add(Box.createRigidArea(new Dimension(0,5)));
listPane.add(listScroller);
listPane.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
```


Exemple avec 2 BoxLayout



Exemple avec 2 BoxLayout

```
JPanel buttonPane = new JPanel();  // BoxLayout horizontal
buttonPane.setLayout(new BoxLayout(buttonPane,
                                   BoxLayout.X_AXIS));
buttonPane.setBorder(BorderFactory.createEmptyBorder
                                        (0, 10, 10, 10));
buttonPane.add(Box.createHorizontalGlue());
buttonPane.add(cancelButton);
buttonPane.add(Box.createRigidArea
                                  (new Dimension(10, 0)));
buttonPane.add(setButton);
Container contentPane = getContentPane();
contentPane.add(listPane, BorderLayout.CENTER);
contentPane.add(buttonPane, BorderLayout.SOUTH);
```

Afficheur `GridBagLayout`

Dispose les composants sur une grille dont la taille des cellules peut varier.

> Un composant est placé sur une cellule dans la grille (sa position) mais il peut occuper plusieurs cellules en hauteur ou en largeur.

> Chaque composant possède un jeu de contraintes précisant la mise en page. On peut les définir via un objet `GridBagConstraints` que l'on associe au composant (avec `setConstraints`) avant de l'ajouter au conteneur.

`GridBagConstraints`

`gridx, gridy` : origine (en cellules) dans la grille.

`gridwidth, gridheight` : zone allouée (en cellules).

`weightx, weighty` : comment l'espace libre en x et y est redistribué entre les composants en cas de changement de taille de la grille.

`anchor` : ancrage du composant au sein de sa zone.

`ipadx, ipady` : de combien la taille du composant doit être augmentée au-delà de sa taille minimale.

`fill` : comment le composant doit être redimensionné pour remplir la zone qui lui est allouée.

`insets` : espaces qui peuvent être ajoutés autour du composant au sein de la zone allouée.

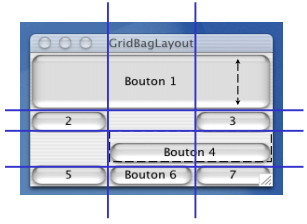
Constantes de `GridBagConstraints`

`gridx, gridy`: GridBagConstraints.RELATIVE = après le dernier composant ajouté.

`gridwidth, gridheight`: GridBagConstraints.RELATIVE = après le dernier dans cette rangée (ou colonne). GridBagConstraints.REMAINDER = le dernier de cette colonne ou rangée.

`fill`: GridBagConstraints.NONE, GridBagConstraints.VERTICAL, GridBagConstraints.HORIZONTAL, GridBagConstraints.BOTH.

Exemple: `GridBagLayout`



Pas d'afficheur: positionnement absolu

```
public class NoneWindow extends JFrame {
        private JButton b1, b2, b3;
    public NoneWindow() {
        Container contentPane = getContentPane();
        contentPane.setLayout(null);

        b1 = new JButton( "un");
        contentPane.add(b1);
        b2 = new JButton( "deux");
        contentPane.add(b2);
        b3 = new JButton( "trois");
        contentPane.add(b3);
```


Positionnement absolu

```
Insets insets = contentPane.getInsets();

b1.setBounds(25 + insets.left, 5 + insets.top, 75, 20);
b2.setBounds(55 + insets.left, 35 + insets.top, 75, 20);
b3.setBounds(150 + insets.left, 15 + insets.top, 75, 30);
        . . .
    }
    . . .
}
```