

Gestion de la répartition

- Introduction
- Compléments sur quelques LayoutManager
 - FlowLayout
 - GridLayout
 - BorderLayout
 - CardLayout
- GridBagLayout
- BoxLayout
- OverlayLayout
- GroupLayout
- SpringLayout
- Layout perso

Gestion de la répartition des composants

- Spécifiée par les interfaces `LayoutManager` et `LayoutManager2`

```
public interface LayoutManager {  
    void addLayoutComponent(String name, Component comp)  
    void layoutContainer(Container parent)  
    Dimension minimumLayoutSize(Container parent)  
    Dimension preferredLayoutSize(Container parent)  
    void removeLayoutComponent(Component comp)  
}
```

→ `FlowLayout`
`GridLayout`

```
public interface LayoutManager2 extends LayoutManager {  
    void addLayoutComponent(Component comp, Object constraints)  
    float getLayoutAlignmentX(Container target)  
    float getLayoutAlignmentY(Container target)  
    void invalidateLayout(Container target)  
    Dimension maximumLayoutSize(Container target)  
}
```

→ gestion à l'aide
de contraintes
`BorderLayout`
...

- Création (avant affichage)

- composants non valides

composant valide
=
sa configuration est à jour
(taille + position)

- Appel de `JFrame.pack`

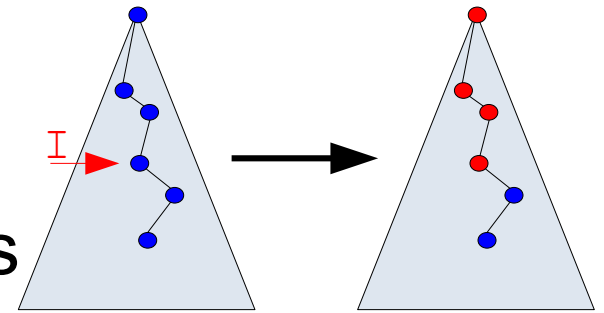
- 1^{re} application de la politique de répartition par le gestionnaire du `ContentPane`
- tous les sous-composants sont validés

- Au cours de l'exécution de l'application

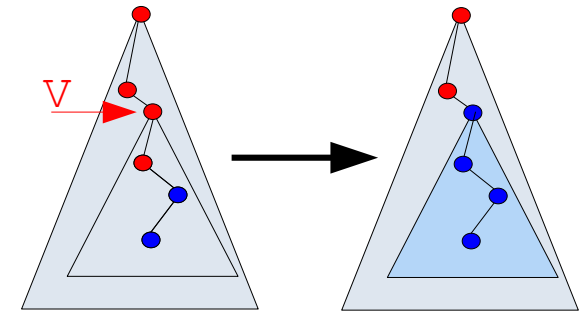
- retailage de la fenêtre
- appel de `Component.validate`
- appel de `JComponent.revalidate`

} appel de
`layoutContainer`
du `LayoutManager`

- `Component.invalidate()`
 - le composant et sa super-hiérarchie doivent être reconfigurés

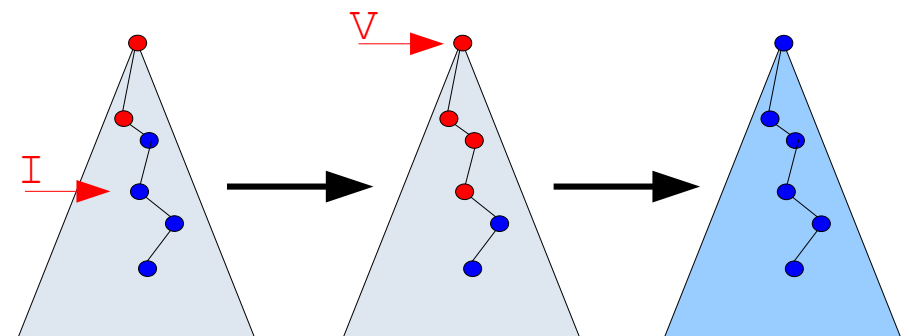


- `Component.validate()`
 - à utiliser sur un conteneur
 - active récursivement tous les gestionnaires de répartition à partir de ce conteneur



- `JComponent.revalidate()` ← composants Swing

- invalide le composant et sa super hiérarchie
- jusqu'à `x` tel que `isValidateRoot()`
- valide `x` sur EDT
- après** tout événement antérieur



Méthodes utilisées par un gestionnaire pour placer les composants

- Espace nécessaire au composant :
 - `Dimension get{Minimum|Preferred|Maximum}Size()`
- Fixer la taille du composant :
 - `void setSize(Dimension)`
- Fixer la position du composant :
 - `void setLocation(Point)`
- Fixer à la fois la taille et la position :
 - `void setBounds(int x, int y, int w, int h)`

Méthodes utilisables par vous pour préparer vos composants

- Récupérer la taille du composant :
 - `Dimension getSize()`
 - `int get{Height|Width}()`
- Récupérer la position du composant :
 - `Point getLocation()`
 - `int get{X|Y}()`
- Fixer la taille du composant :
 - `void setMinimumSize(Dimension)`
 - `void setPreferredSize(Dimension)`
 - `void setMaximumSize(Dimension)`

Configuration initiale de certains composants Swing

– JComponent

- taille minimum : (0, 0)
- taille préférée : (0, 0)
- taille maximale : (Short.MAX_VALUE, Short.MAX_VALUE)

– JButton et JLabel

- taille minimum : icône + texte + marges
- taille préférée : icône + texte + marges
- taille maximale : icône + texte + marges

– JTextField

- taille minimum : marges
- taille préférée : texte + marges
- taille maximale : (Integer.MAX_VALUE, Integer.MAX_VALUE)

java.awt.FlowLayout

- *Voir cours de POO2*
- Constantes d'alignement :
 - Absolues : LEFT, CENTER, RIGHT
 - Relatives : LEADING, TRAILING
 - pour une orientation des composants gauche-droite
 - LEADING \Leftrightarrow LEFT, TRAILING \Leftrightarrow RIGHT
 - pour une orientation des composants droite-gauche
 - LEADING \Leftrightarrow RIGHT, TRAILING \Leftrightarrow LEFT

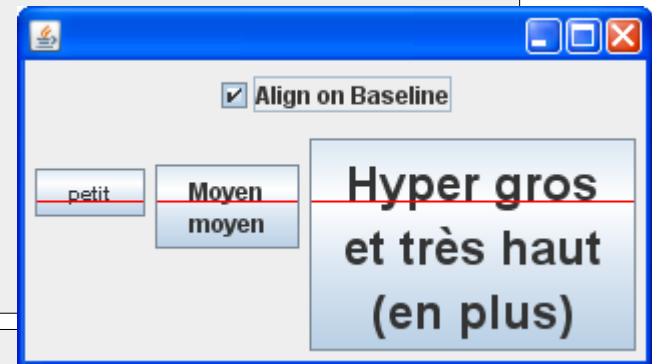
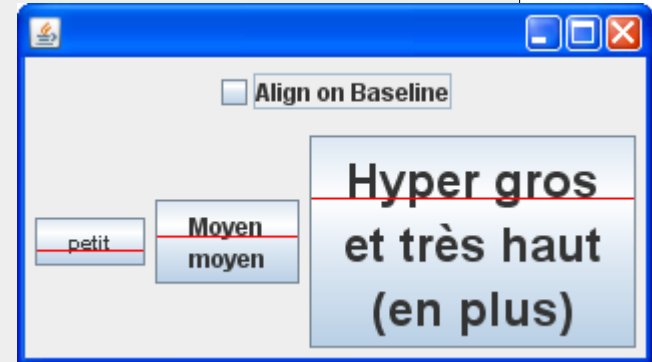
```
Component :  
    ComponentOrientation getComponentOrientation()  
  
ComponentOrientation :  
    boolean isHorizontal()  
    boolean isLeftToRight()
```


Alignement sur la ligne de base

```
final JPanel p = new JPanel(new FlowLayout());
{ //--
    String t1 = "<html><body><p><font size=\"-2\">...</font></p></body></html>" ;
    String t2 = "<html>...</html>" ;
    String t3 = "<html>...</html>" ;
    p.add(new MyButton(t1));
    p.add(new MyButton(t2));
    p.add(new MyButton(t3));
} //--

JPanel q = new JPanel();
{ //--
    JCheckBox cb = new JCheckBox("Align on Baseline");
    cb.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JCheckBox cb = (JCheckBox) e.getSource();
            FlowLayout fl = (FlowLayout) p.getLayout();
            fl.setAlignOnBaseline(cb.isSelected());
            p.revalidate();
        }
    });
    q.add(cb);
} //--
```

```
class MyButton extends JButton {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        int baseline = getBaseline(getWidth(), getHeight());
        if (baseline >= 0) {
            g.setColor(Color.RED);
            g.drawLine(0, baseline, getWidth(), baseline);
        }
    }
}
```



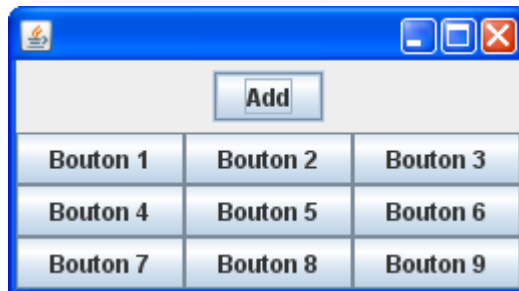
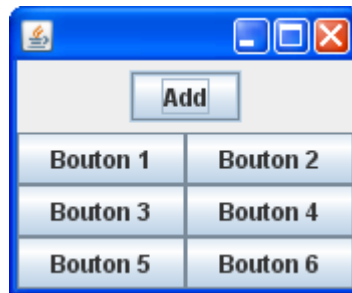
java.awt.GridLayout

- *Voir cours de POO2*
- Algorithme de calcul de `rows` et `cols` :

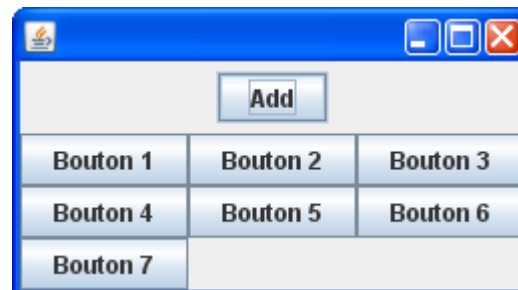
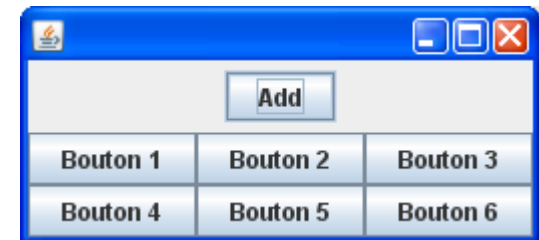
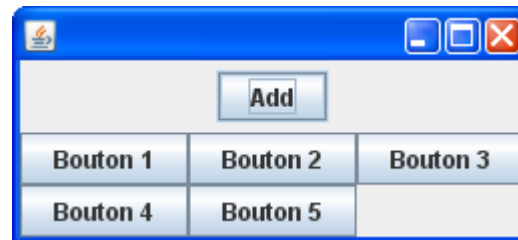
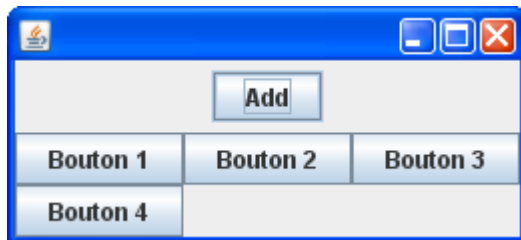
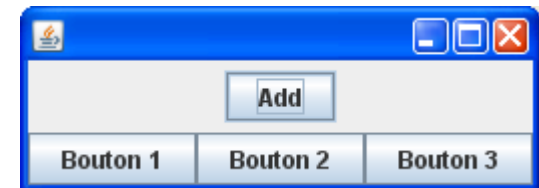
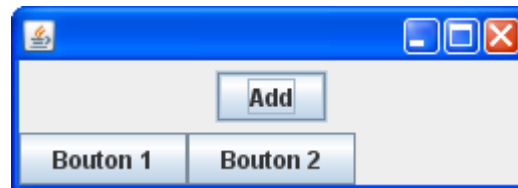
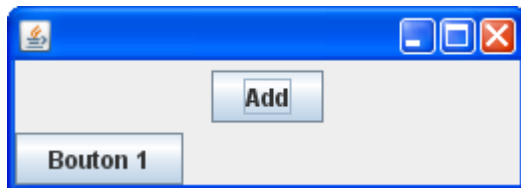
```
Si rows > 0 Alors
    cols ← (nComponents + rows - 1) / rows
Sinon
    rows ← (nComponents + cols - 1) / cols
FinSi
```

- Remplissage gauche/droite puis haut/bas
- On peut indiquer 0 ligne ou (xor) 0 colonne
 - 0 ligne : le nombre de colonnes ne varie pas, le nombre de lignes est ajusté
 - 0 colonne : le nombre de lignes ne varie pas, le nombre de colonnes est ajusté

GridLayout (3, 0) ou (3, n)



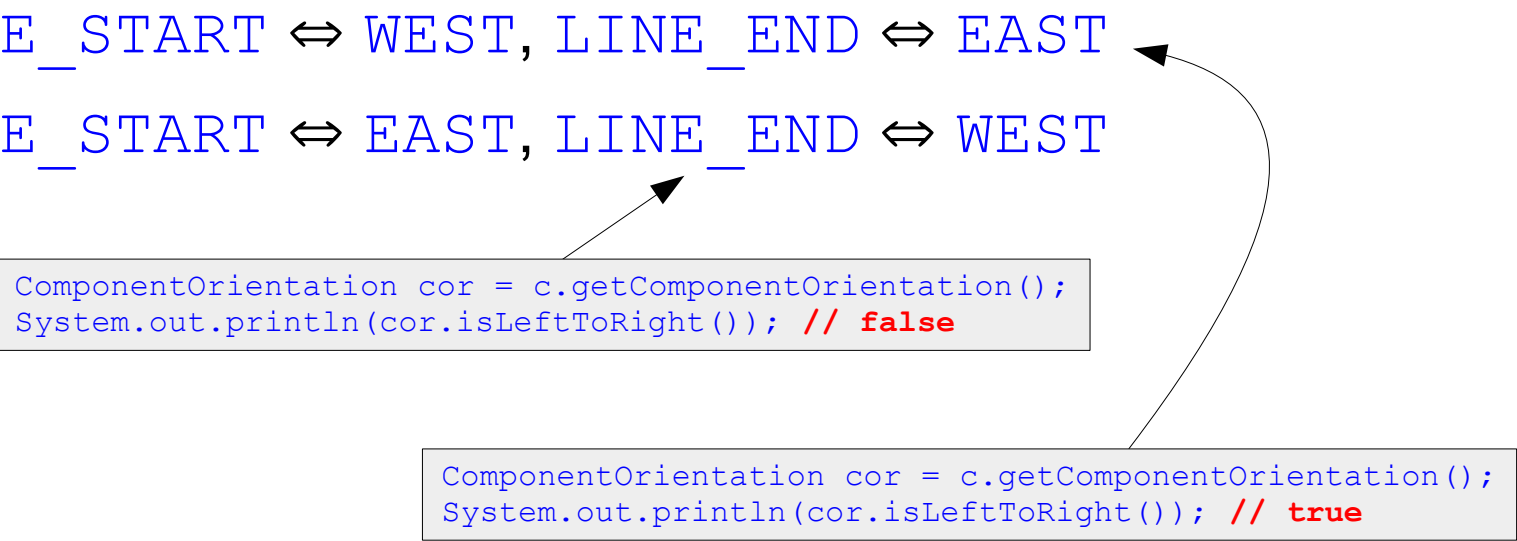
GridLayout (0, 3)



java.awt.BorderLayout

- *Voir cours MPOO2*
- Contraintes de placement :
 - Absolues : NORTH, SOUTH, WEST, EAST
 - Relatives : PAGE_START, PAGE_END, LINE_START, LINE_END
 - LINE_START \Leftrightarrow WEST, LINE_END \Leftrightarrow EAST
 - LINE_START \Leftrightarrow EAST, LINE_END \Leftrightarrow WEST

```
ComponentOrientation cor = c.getComponentOrientation();  
System.out.println(cor.isLeftToRight()); // false
```



```
ComponentOrientation cor = c.getComponentOrientation();  
System.out.println(cor.isLeftToRight()); // true
```

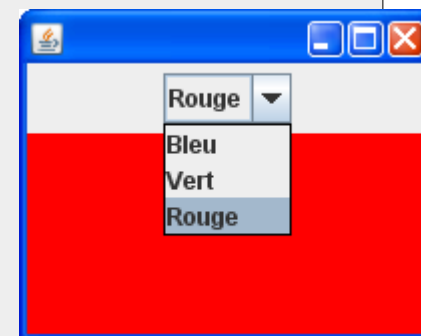
java.awt.CardLayout

- Affiche un seul composant à la fois, mais on peut choisir celui que l'on veut afficher
 - **void** first(Container parent)
 - affiche le premier composant du conteneur
 - **void** next(Container parent)
 - affiche le composant suivant du conteneur (cyclique)
 - **void** previous(Container parent)
 - affiche le composant précédant du conteneur (cyclique)
 - **void** last(Container parent)
 - affiche le dernier composant du conteneur
 - **void** show(Container parent, String name)
 - affiche le composant de nom `name` du conteneur

Exemple d'utilisation de CardLayout

```
HashMap<String, JPanel> map = new HashMap<String, JPanel>();
// map : nomCouleur -> JPanel de cette couleur, de taille 200x100
-----
final JPanel p = new JPanel(new CardLayout());
{ //--
    for (String n : map.keySet()) {
        p.add(map.get(n), n);
    }
} //--
frame.add(p, BorderLayout.CENTER);
-----
JPanel q = new JPanel();
{ //--
    JComboBox cb = new JComboBox();
    for (String n : map.keySet()) {
        cb.addItem(n);
    }
    cb.addItemListener(new ItemListener() {
        public void itemStateChanged(ItemEvent e) {
            CardLayout cl = (CardLayout) (p.getLayout());
            cl.show(p, (String) e.getItem());
        }
    });
    q.add(cb);
} //--
frame.add(q, BorderLayout.NORTH);
```

ajoute un composant

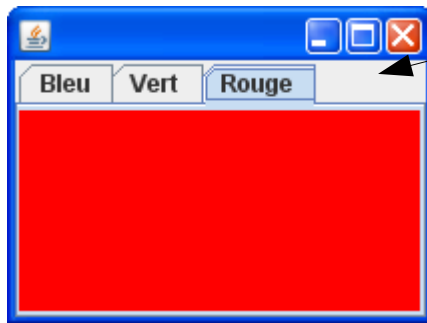


affiche un composant

- Utilisation fastidieuse
 - il faut gérer un mécanisme pour la sélection du composant à afficher
- JTabbedPane
 - fournit un service équivalent
 - plus simple à utiliser

Exemple d'utilisation de JTabbedPane

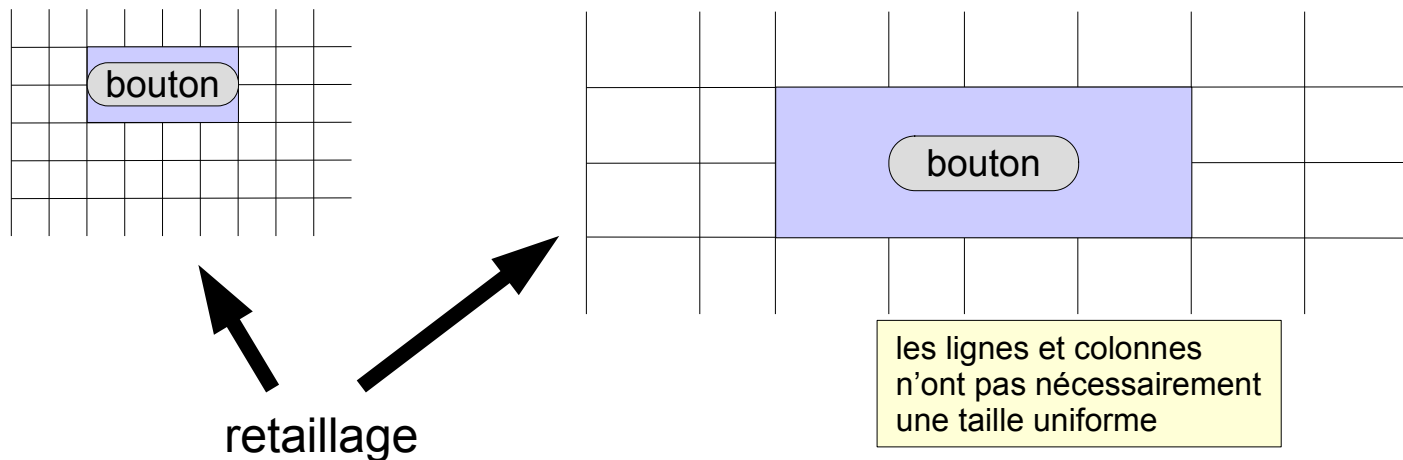
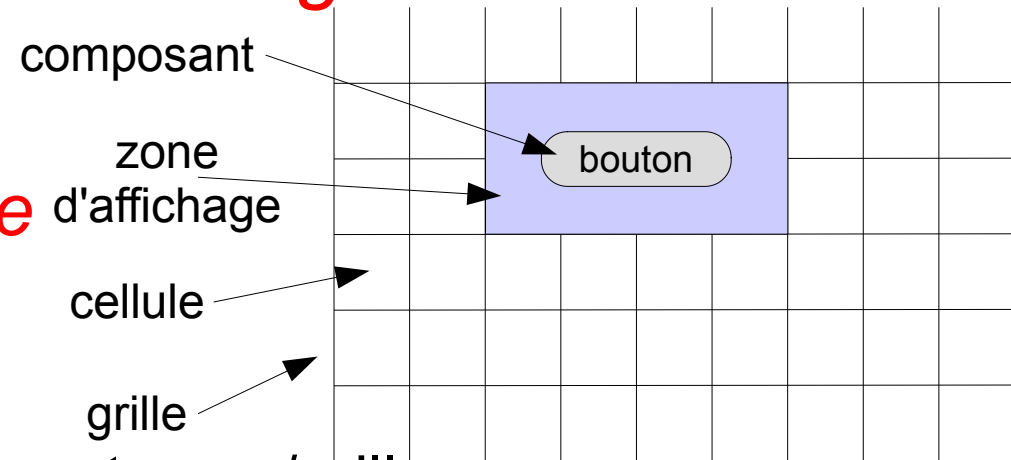
```
HashMap<String, JPanel> map = new HashMap<String, JPanel>();  
// map : nomCouleur -> JPanel de cette couleur, de taille 200×100  
-----  
JTabbedPane p = new JTabbedPane(); {  
    for (String n : map.keySet()) {  
        p.add(map.get(n), n);  
    }  
}  
frame.add(p, BorderLayout.CENTER);
```



Onglets : plus besoin de JComboBox
pour piloter le CardLayout

java.awt.GridBagLayout

- Conteneur découpé selon une *grille de cellules*
- Chaque composant
 - sur une *zone d'affichage*
 - plusieurs *cellules*
 - avec une *contrainte*
 - rapports composant/zone et zone/grille



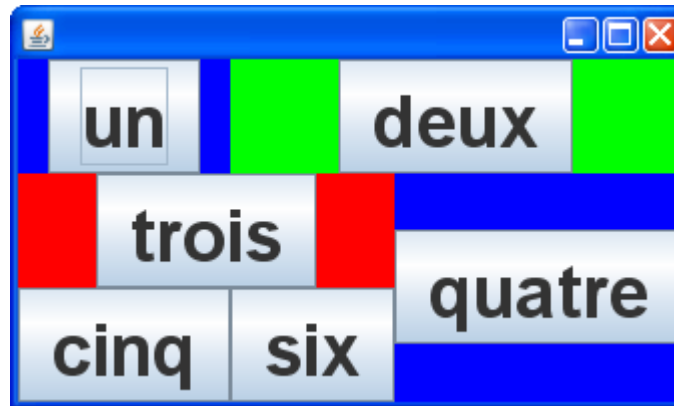
Mise en place

- Création : un seul constructeur
 - `GridBagLayout()`
- Ajout des composants
 - Association conteneur - gestionnaire
 - `JPanel p = new JPanel(new GridBagLayout());`
 - Puis ajout des composants **avec** leur contrainte (stockées dans le `GridBagLayout` !)
 - `p.add(Component, Object);`
 - contrainte de type `GridBagConstraints` (ou mieux : `GBC`)

java.awt.GridBagConstraints

- Contraintes appliquées au composant par le gestionnaire :
 - Position de la zone sur la grille
 - `gridx`, `gridy`
 - Taille de la zone
 - `gridwidth`, `gridheight`
 - Marges et disposition du composant dans sa zone
 - `insets`, `anchor`
 - Marges internes du composant
 - `ipadx`, `ipady`
 - Distribution de l'espace supplémentaire
 - `fill`, `weightx`, `weighty`

Exemple



```
JPanel p = new JPanel(new GridBagLayout());
JButton b1 = new JButton("un");
...
JButton b6 = new JButton("six");

p.add(b1, new GridBagConstraints(...));
p.add(b2, new GridBagConstraints(...));
p.add(b3, new GridBagConstraints(...));
p.add(b4, new GridBagConstraints(...));
p.add(b5, new GridBagConstraints(...));
p.add(b6, new GridBagConstraints(...));
```

GridBagConstraints

gridx
gridy
gridwidth
gridheight
insets
ipadx
ipady
anchor
fill
weightx
weighty

Positionnement de la zone d'affichage

- **int** `gridx` : (RELATIVE, ≥ 0)

- numéro de colonne (début en 0) de la cellule qui contient le csg de la zone d'affichage

~~– RELATIVE
= à la suite du précédent~~

	0	1	2
0	un		deux
1		trois	
2	cinq	six	quatre

- **int** `gridy` : (RELATIVE, ≥ 0)

- numéro de ligne (début en 0) de la cellule qui contient le csg de la zone d'affichage

~~– RELATIVE
= à la suite du précédent~~

```
b1 : gridx=0, gridy=0  
b2 : gridx=1, gridy=0  
b3 : gridx=0, gridy=1  
b4 : gridx=2, gridy=1  
b5 : gridx=0, gridy=2  
b6 : gridx=1, gridy=2
```

Taille de la zone d'affichage

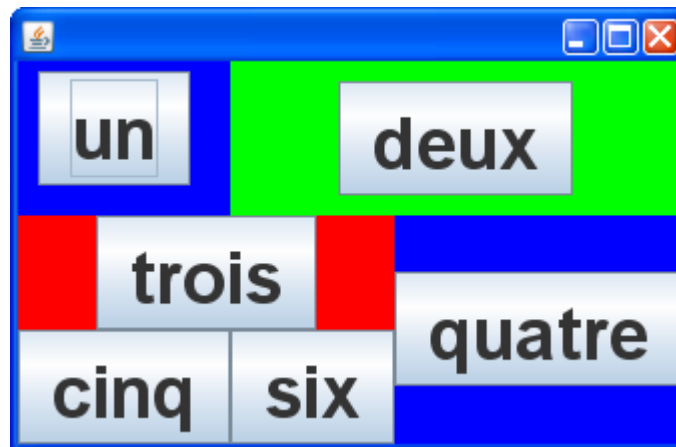
- **int** `gridwidth` : $(1, \geq 0)$
 - nombre de cellules occupées par la zone horizontalement
- **int** `gridheight` : $(1, \geq 0)$
 - nombre de cellules occupées par la zone verticalement

	0	1	2
0	un		deux
1		trois	
2	cinq	six	quatre

```
b1 : gridwidth=1, gridheight=1  
b2 : gridwidth=2, gridheight=1  
b3 : gridwidth=2, gridheight=1  
b4 : gridwidth=1, gridheight=2  
b5 : gridwidth=1, gridheight=1  
b6 : gridwidth=1, gridheight=1
```

Marge entre le composant et sa zone d'affichage

- `Insets insets : (new Insets(0, 0, 0, 0))`
 - espace à rajouter à l'intérieur de la zone, autour du composant
 - dans l'ordre : nord, ouest, sud, est

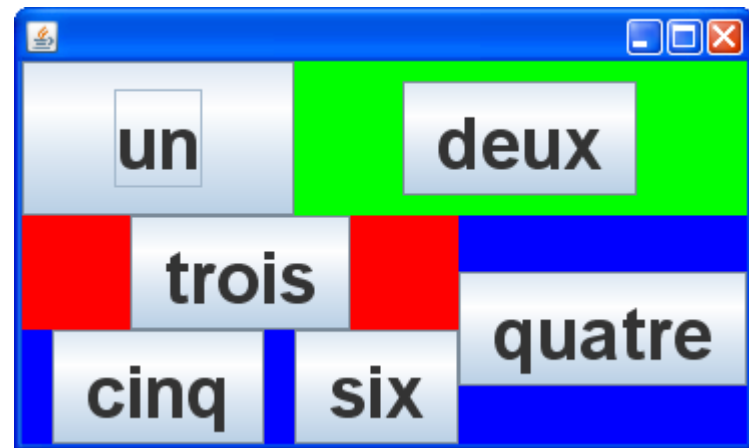


```
b1 : insets(top=5, left=10, bottom=15, right=20)
```


Marges internes au composant

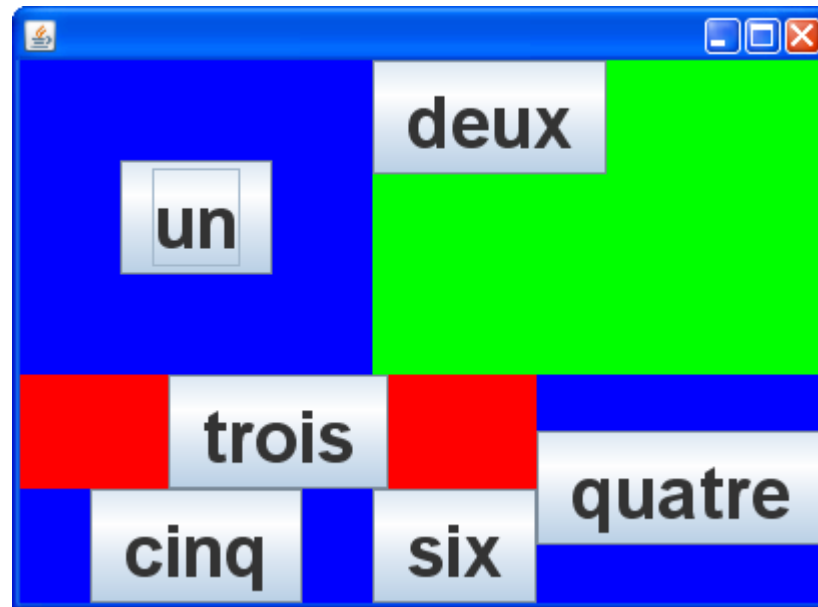
- **int** `ipadx` : (0)
 - espace à rajouter à la taille minimum du composant, répartie horizontalement de chaque côté (en parts égales)
- **int** `ipady` : (0)
 - espace à rajouter à la taille minimum du composant, répartie verticalement de chaque côté (en parts égales)

```
b1 : ipadx=60, ipady=20
```



Ancre du composant dans sa zone d'affichage

- **int** `anchor` : (`CENTER`)
 - où placer le composant si sa zone d'affichage est trop grande ?
 - emplacement du composant dans sa zone
 - absolu
 - `NORTHWEST`, `NORTH`, `NORTHEAST`, `WEST`, `CENTER`, `EAST`, `SOUTHWEST`, `SOUTH`, `SOUTHEAST`
 - par rapport à l'orientation
 - `FIRST_LINE_START`, `PAGE_START`, `FIRST_LINE_END`, `LINE_START`, `CENTER`, `LINE_END`, `LAST_LINE_START`, ...
 - par rapport à la ligne de base
 - `ABOVE_BASELINE_LEADING`, `ABOVE_BASELINE`, `ABOVE_BASELINE_TRAILING`, `BASELINE_LEADING`, `BASELINE`, `BASELINE_TRAILING`, ...



```
b1 : insets(top=50, left=50, bottom=50, right=50)  
b2 : anchor=FIRST_LINE_START
```

Étirement du composant sur sa zone d'affichage

- **int** *fill* : (NONE)
 - comment étirer le composant si sa zone d'affichage est trop grande ?
 - étirement du composant dans sa zone
 - NONE : pas d'étirement
 - HORIZONTAL : étirement horizontal seulement
 - VERTICAL : étirement vertical seulement
 - BOTH : étirement dans les deux sens

NONE



BOTH



HORIZONTAL

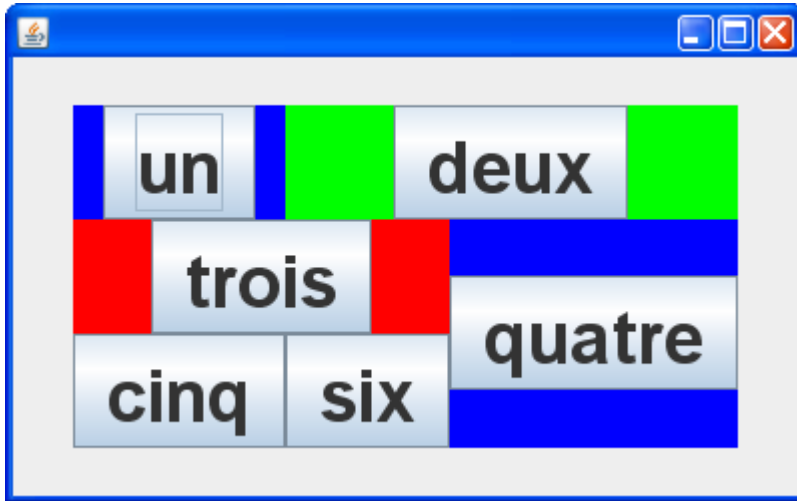


VERTICAL



Attribution à la zone d'affichage de l'espace en trop alloué au conteneur

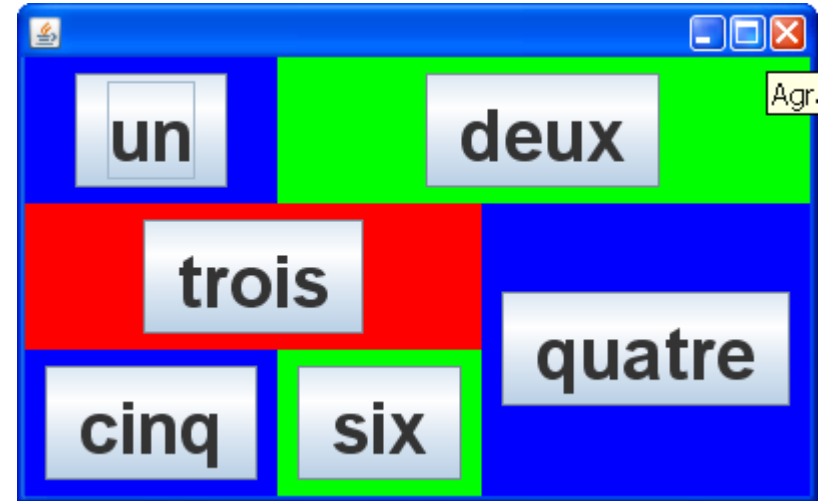
- **double** `weightx` : $(0, \geq 0)$
 - comment répartir l'espace supplémentaire horizontal entre les différentes colonnes ?
- **double** `weighty` : $(0, \geq 0)$
 - comment répartir l'espace supplémentaire vertical entre les différentes lignes ?



```

b1 : weightx=0, weighty=0
b2 : weightx=0, weighty=0
b3 : weightx=0, weighty=0
b4 : weightx=0, weighty=0
b5 : weightx=0, weighty=0
b6 : weightx=0, weighty=0

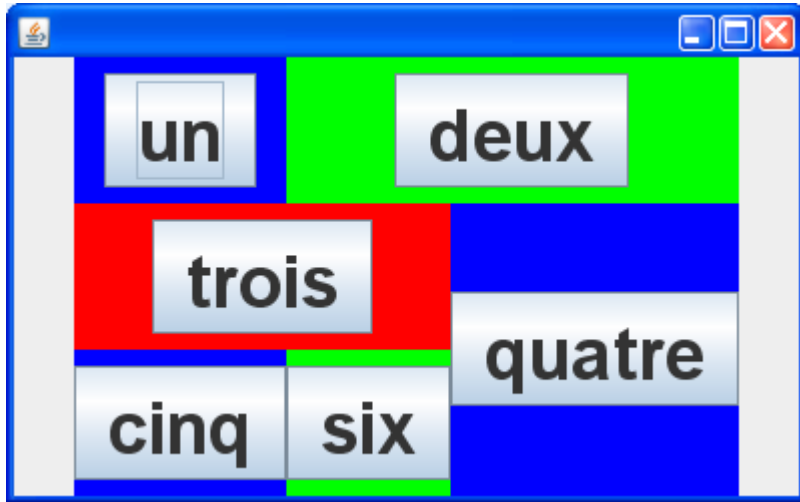
```



```

b1 : weightx=1, weighty=1
b2 : weightx=1, weighty=1
b3 : weightx=1, weighty=1
b4 : weightx=1, weighty=1
b5 : weightx=1, weighty=1
b6 : weightx=1, weighty=1

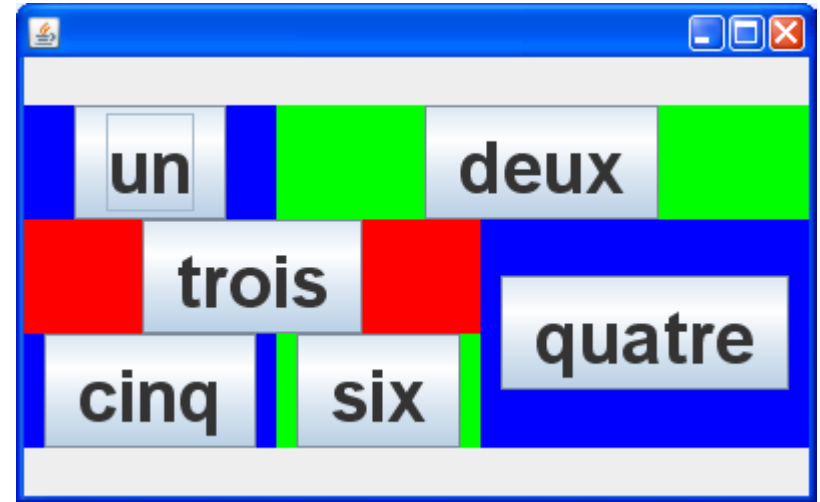
```



```

b1 : weightx=0, weighty=1
b2 : weightx=0, weighty=1
b3 : weightx=0, weighty=1
b4 : weightx=0, weighty=1
b5 : weightx=0, weighty=1
b6 : weightx=0, weighty=1

```



```

b1 : weightx=1, weighty=0
b2 : weightx=1, weighty=0
b3 : weightx=1, weighty=0
b4 : weightx=1, weighty=0
b5 : weightx=1, weighty=0
b6 : weightx=1, weighty=0

```