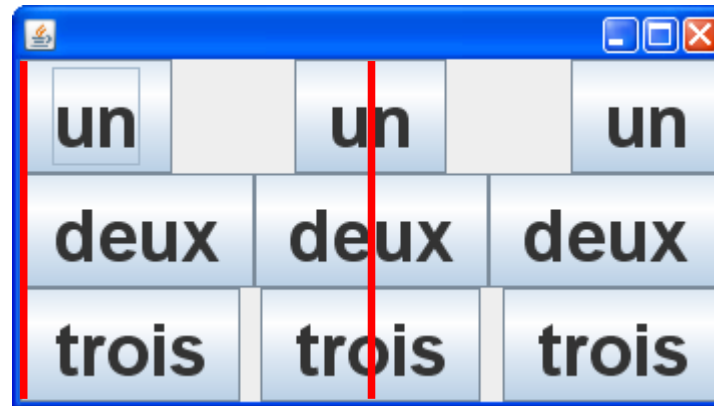


# javax.swing.BoxLayout

- Empile les composants horizontalement ou verticalement
  - respect des contraintes d'alignement
  - prise en compte de la taille max
- Contraintes stockées dans les composants (méthodes de `Component`) :
  - **void** `setAlignment{X|Y}(float)` ←
  - **void** `setMaximumSize(Dimension)`

# Exemple

- Trois `JPanel` dotés de `BoxLayout` verticaux



`b[i].setAlignmentX(0);`

`b[i].setAlignmentX(0.5f);`

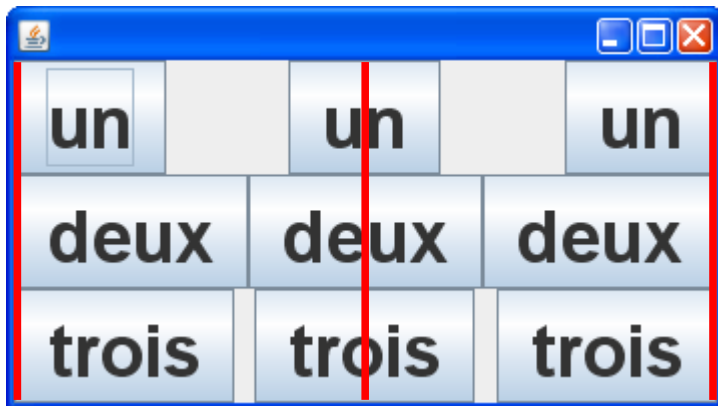
`b[i].setAlignmentX(1);`

- `setAlignmentX` → proportion de surface du composant, visible à gauche de l'axe rouge

- Dans un conteneur à `BoxLayout`

- chaque composant est étiré autant que nécessaire\*, sans dépasser sa largeur maximale

\* : pour occuper l'espace disponible



Comportement des boutons  
par défaut = pas d'étirement  
→ (max = preferred)



```
b[i].setMaximumSize(new Dimension(  
    Integer.MAX_VALUE,  
    b[i].getPreferredSize().height  
));
```

# Montées et descentes

- Valeur d'alignement d'un composant :

- `{get|set}Alignment{X|Y}`

- entre `0.0f` et `1.0f`

- `0.5f` par défaut pour `JComponent`

```
pour JButton et JLabel :  
getAlignmentX() == 0.0f  
getAlignmentY() == 0.5f
```

- **Montée** = taille préférée  $\times$  alignement

- Ex. : montée horizontale

- `Dimension d = cmp.getPreferredSize();`  
`int hAscent = (int) (d.width * cmp.getAlignmentX());`

- **Descente** = taille préférée – montée

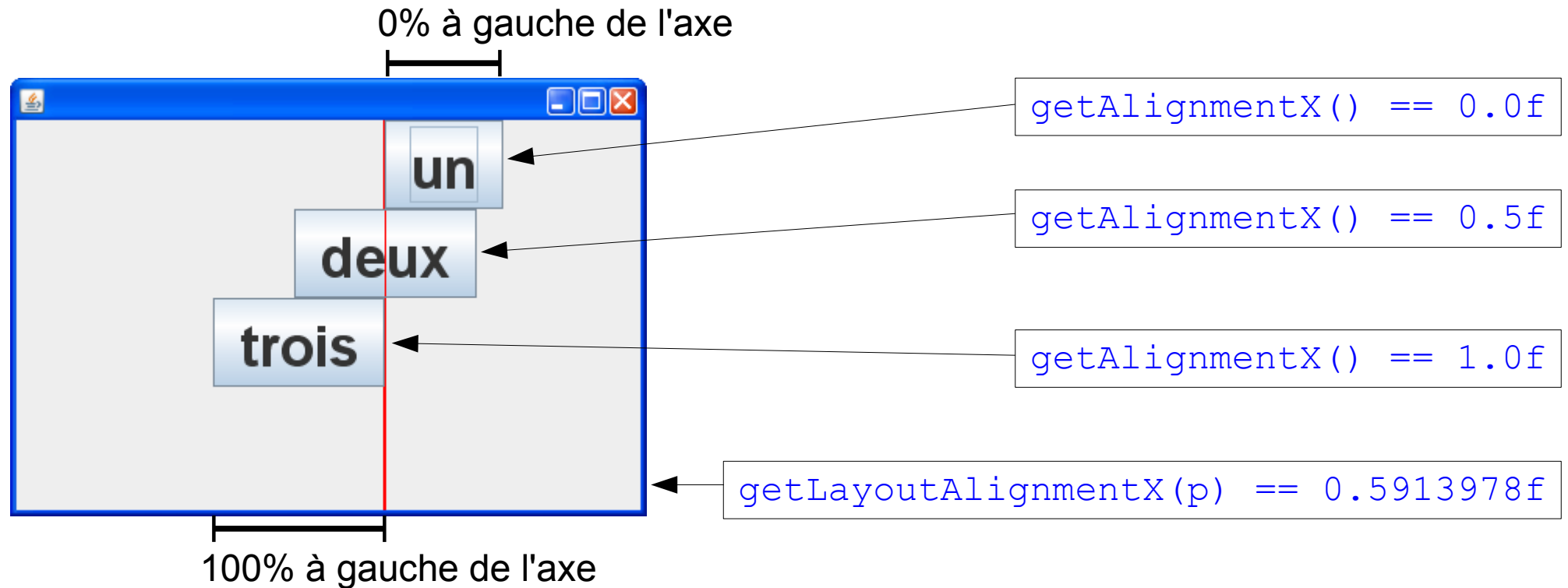
- Ex. : descente horizontale

- `int hDescent = d.width - hAscent;`

# Définition de l'axe d'un conteneur

- Ligne fictive d'alignement des sous-composants
- Ex. position de l'axe vertical d'un conteneur à `BoxLayout` vertical :
  - dépend de l'alignement en X des composants du conteneur
  - $width \times \max(hAscent_i) / [\max(hAscent_i) + \max(hDescent_i)]$
- Cette valeur est calculée par appel à `LayoutManager2.getLayoutAlignmentX`

# Position de l'axe d'alignement

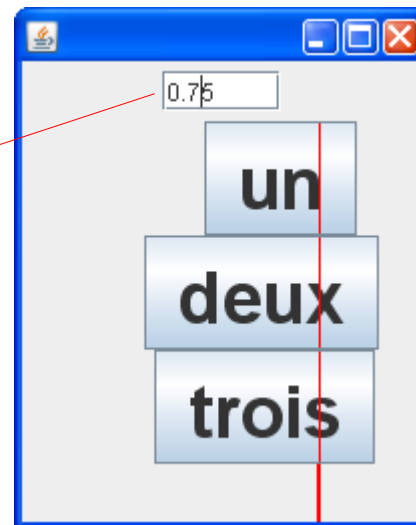
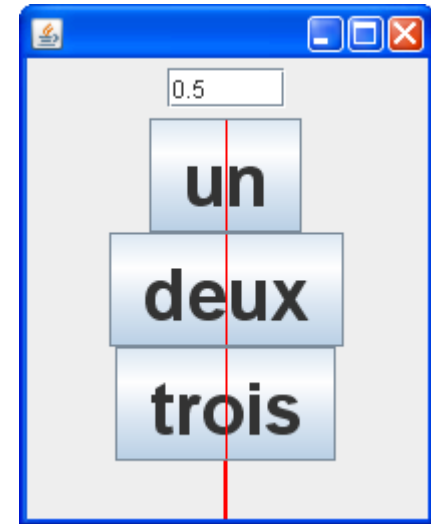
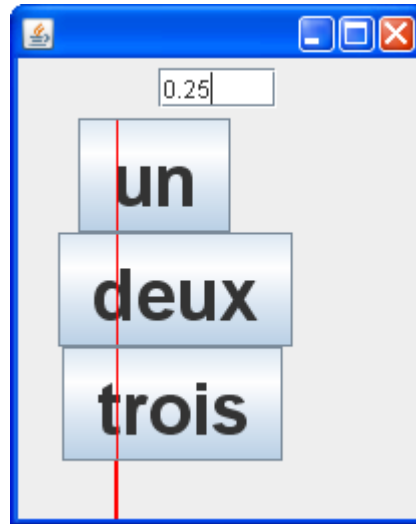


|  | "un"                                | "deux"           | "trois"           |                          |
|--|-------------------------------------|------------------|-------------------|--------------------------|
| <code>max(ascent<sub>i</sub>) ==</code>  | <code>max(0,</code>                 | <code>58,</code> | <code>110)</code> | pixels à gauche de l'axe |
| <code>max(descent<sub>i</sub>) ==</code> | <code>max(76,</code>                | <code>58,</code> | <code>0)</code>   | pixels à droite de l'axe |
| <code>getLayoutAlignmentX(p) ==</code>   | <code>110 / 186 == 0.5913978</code> |                  |                   |                          |

# Exemples de `BoxLayout` verticaux avec largeur maximale bornée

```
b[i].getMaximumSize().equals(b[i].getPreferredSize())
```

Par défaut  
pour les  
`JButtons`



```
b[i].setAlignmentX(0.75f)
```

# Exemples de `BoxLayout` verticaux avec largeur maximale non bornée

```
b[i].getMaximumSize().width == Integer.MAX_VALUE
```

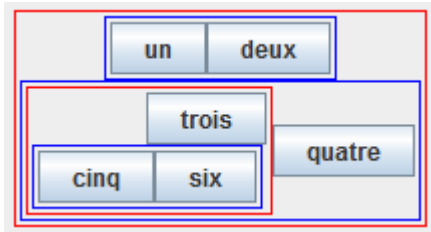


`getAlignmentX() == 0.0f`

`getAlignmentX() == 0.5f`

`getAlignmentX() == 1.0f`





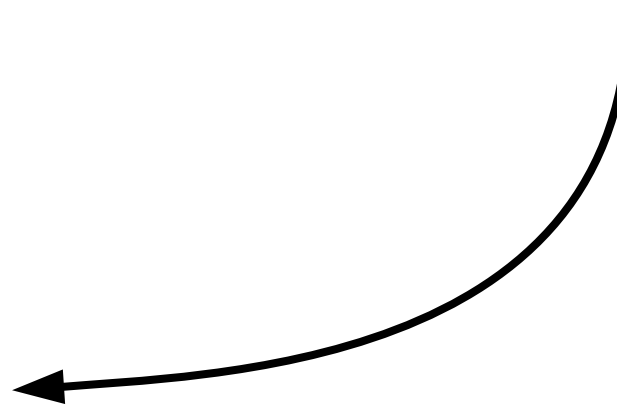
Comportement  
par défaut



```
b[i].setMaximumSize(  
    new Dimension(  
        Integer.MAX_VALUE,  
        Integer.MAX_VALUE  
    )  
);
```



```
b[i].setMaximumSize(  
    new Dimension(  
        Integer.MAX_VALUE,  
        Integer.MAX_VALUE  
    )  
);  
b[2].setAlignmentX(0.5f);
```



# Mise en place

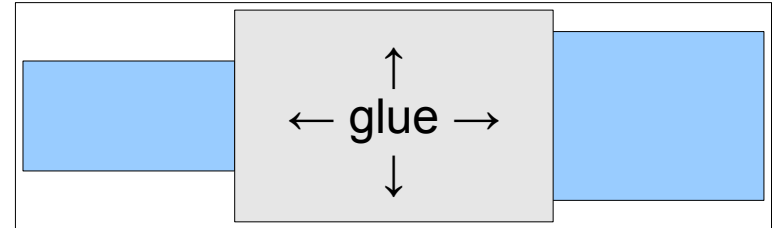
- Création : un seul constructeur
  - `BoxLayout(Container target, int axis)`
    - Axes absolus :
      - `X_AXIS, Y_AXIS`
    - Axes relatifs (à la propriété `componentOrientation`)
      - `LINE_AXIS, PAGE_AXIS`
- Association conteneur / gestionnaire
  - `JPanel p = new JPanel(null);`  
`p.setLayout(new BoxLayout(p, X_AXIS));`
- Puis ajout des composants sans contrainte supplémentaire
  - `Container.add(Component);`

# javax.swing.Box

- `Box` = conteneur transparent  
doté d'un `BoxLayout`

```
Box b = new Box(BoxLayout.X_AXIS);  
ou encore plus simplement  
Box b = Box.createHorizontalBox();
```

- *glue* = composant invisible,  
extensible, de tailles



- min, pref : `Dimension(0, 0)`
- `Component Box.createGlue()`
  - max : `Dimension(Short.MAX_VALUE, Short.MAX_VALUE)`
- `Component Box.createHorizontalGlue()`
  - max : `Dimension(Short.MAX_VALUE, 0)`
- `Component Box.createVerticalGlue()`
  - max : `Dimension(0, Short.MAX_VALUE)`

- **strut** = composant invisible, extensible dans une seule direction, de tailles

- `Box.createHorizontalStrut(int width)`

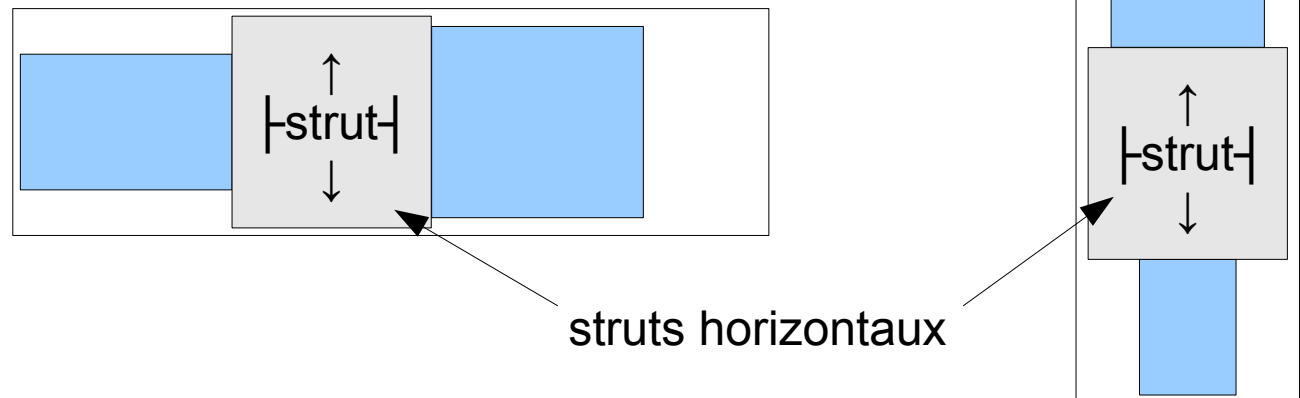
- `min, pref : Dimension(width, 0)`

- `max : Dimension(width, Short.MAX_VALUE)`

- `Box.createVerticalStrut(int height)`

- `min, pref : Dimension(0, height)`

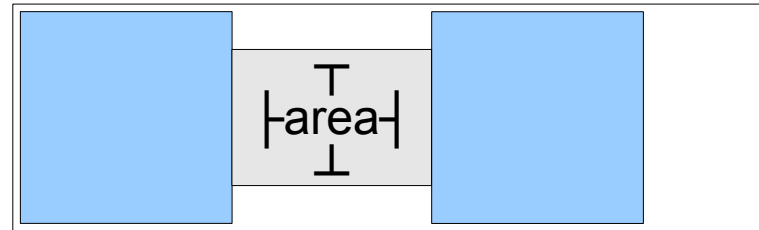
- `max : Dimension(Short.MAX_VALUE, height)`



- *aire rigide* = composant invisible de taille fixe

- `Box.createRigidArea(Dimension d)`

- min, pref, max : d



- *remplisseur* = composant invisible totalement configurable

- `new Box.Filler(Dimension, Dimension, Dimension)`

↑  
min

↑  
pref

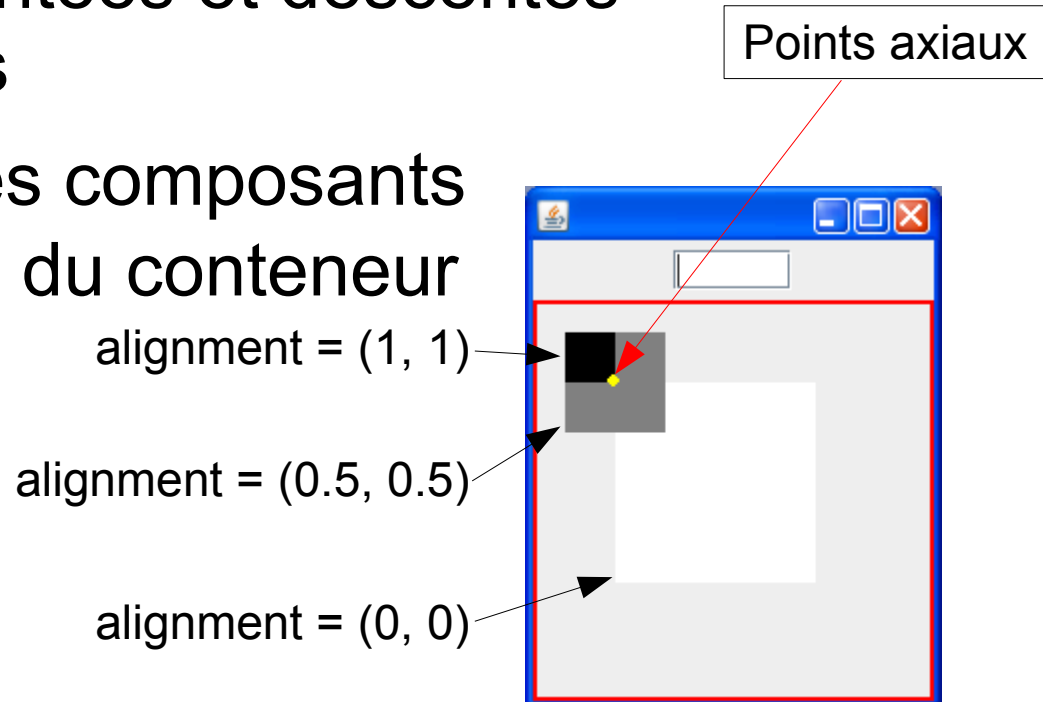
↑  
max

# javax.swing.OverlayLayout

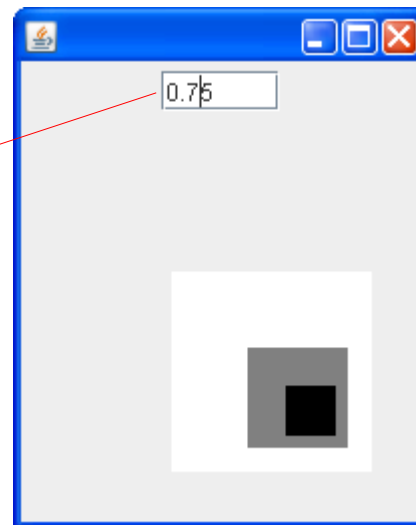
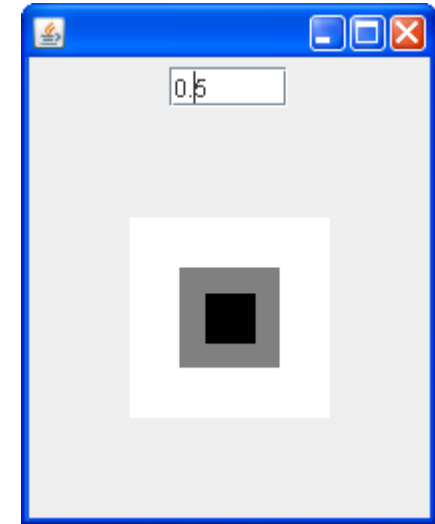
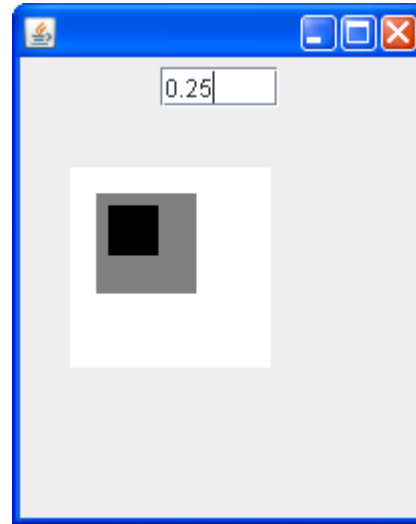
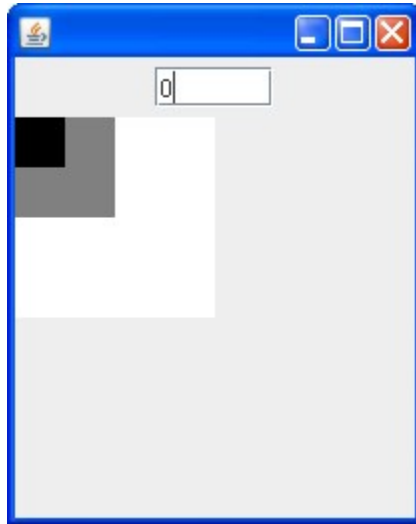
- Empile des composants selon l'axe Z
  - respect des contraintes d'alignement
  - prise en compte de la taille max
- Contraintes stockées dans les composants
  - **void** setAlignment{X|Y}(**float**)
  - **void** setMaximumSize(Dimension)

# Définition du point axial

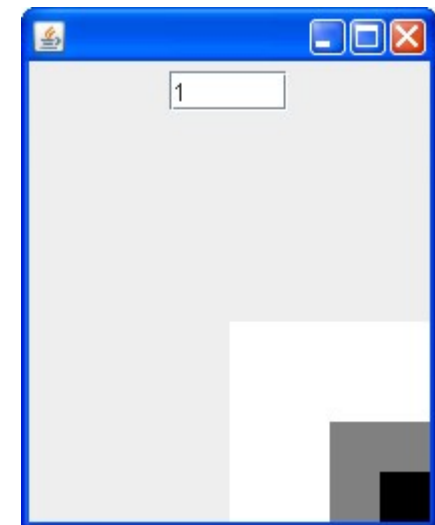
- *Point axial* d'un composant =  
Point(montée en X, montée en Y)
- *Point axial* du conteneur
  - fonction des montées et descentes des composants
  - points axiaux des composants alignés sur celui du conteneur



# Examples



```
lb[i].setAlignmentX(0.75f);  
lb[i].setAlignmentY(0.75f);
```





# Mise en place

- Création : un seul constructeur
  - `OverlayLayout(Container target)`
- Association conteneur - gestionnaire
  - `JPanel p = new JPanel(null);`  
`p.setLayout(new OverlayLayout(p));`
- Puis ajout des composants sans contrainte supplémentaire
  - `Container.add(Component)`

# javax.swing.GroupLayout

- Présentation d'après le tutoriel de Sun :
  - <https://docs.oracle.com/javase/tutorial/uiswing/layout/group.html>
- Création de formulaires
- Chaque composant
  - est enregistré deux fois dans le *gestionnaire*
    - une fois pour sa disposition horizontale
    - une fois pour sa disposition verticale
  - !! pas enregistré dans le *conteneur* !

# Structurer la répartition

- *groupe* = ensemble de composants, de groupes et d'écarts (*gaps*) reliés entre eux de manière séquentielle ou parallèle
- permet de répartir des éléments
  - *en séquence*
  - *en parallèle*

description horizontale  
groupe parallèle



elt1

elt2

elt3

description  
verticale  
groupe  
séquentiel

elt1

elt2

elt3

description horizontale  
groupe séquentiel



elt1

elt2

elt3

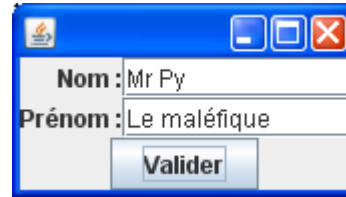
description  
verticale  
groupe  
parallèle

elt1

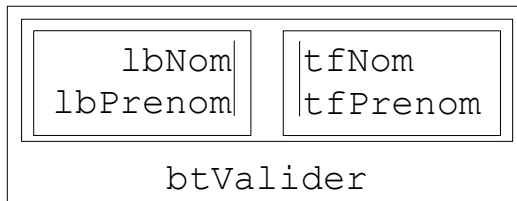
elt2

elt3

# Exemple (1/3)

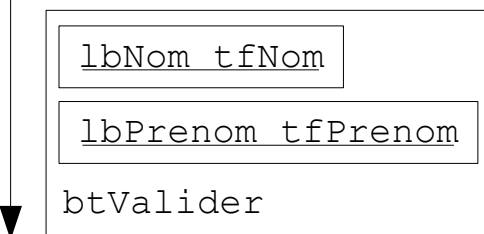


description horizontale



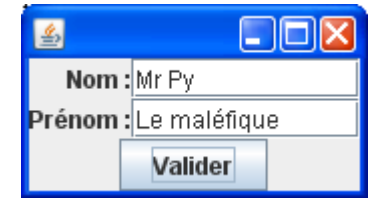
```
horizontal layout =  
    parallel group (CENTER) {  
        sequential group {  
            parallel group (TRAILING) {lbNom, lbPrenom},  
            parallel group (LEADING) {tfNom, tfPrenom},  
        },  
        btValider  
    }
```

description verticale



```
vertical layout =  
    sequential group {  
        parallel group (BASELINE) {lbNom, tfNom},  
        parallel group (BASELINE) {lbPrenom, tfPrenom},  
        btValider  
    }
```

# Exemple (2/3)



A small Java Swing window with a blue title bar. It contains two text input fields: 'Nom : Mr Py' and 'Prénom : Le maléfique'. Below the fields is a button labeled 'Valider'.

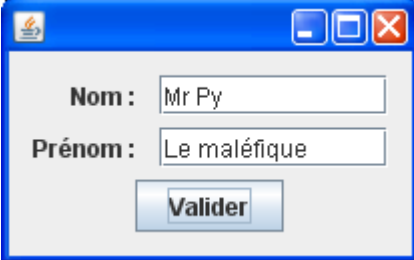
```
GridLayout lmp = new GridLayout(p);
lmp.setHorizontalGroup(
    lmp.createParallelGroup(GroupLayout.Alignment.CENTER)
        .addGroup(lmp.createSequentialGroup()
            .addGroup(lmp.createParallelGroup(GroupLayout.Alignment.TRAILING)
                .addComponent(lbNom)
                .addComponent(lbPrenom)
            )
            .addGroup(lmp.createParallelGroup(GroupLayout.Alignment.LEADING)
                .addComponent(tfNom)
                .addComponent(tfPrenom)
            )
        )
        .addComponent(btValider)
);
lmp.setVerticalGroup(
    lmp.createSequentialGroup()
        .addGroup(lmp.createParallelGroup(GroupLayout.Alignment.BASELINE)
            .addComponent(lbNom)
            .addComponent(tfNom)
        )
        .addGroup(lmp.createParallelGroup(GroupLayout.Alignment.BASELINE)
            .addComponent(lbPrenom)
            .addComponent(tfPrenom)
        )
        .addComponent(btValider)
);
```

```
horizontal layout =
    parallel group (CENTER) {
        sequential group {
            parallel group (TRAILING) {lbNom, lbPrenom},
            parallel group (LEADING) {tfNom, tfPrenom},
        },
        btValider
    }
```

```
vertical layout =
    sequential group {
        parallel group (BASELINE) {lbNom, tfNom},
        parallel group (BASELINE) {lbPrenom, tfPrenom},
        btValider
    }
```

# Exemple (3/3)

```
GridLayout lmp = new GridLayout(p);  
lmp.setAutoCreateGaps(true);  
lmp.setAutoCreateContainerGaps(true);  
lmp.setHorizontalGroup(  
    ...  
);  
lmp.setVerticalGroup(  
    ...  
);
```



A screenshot of a Java Swing window. The window has a blue title bar with standard window controls (minimize, maximize, close). The main content area is light gray. It contains two text input fields. The first field is labeled 'Nom :' and contains the text 'Mr Py'. The second field is labeled 'Prénom :' and contains the text 'Le maléfique'. Below these fields is a button labeled 'Valider'.

# Mise en place

- Création : un seul constructeur
  - `GroupLayout(Container host)`
- Association conteneur - gestionnaire
  - `JPanel p = new JPanel(null);`  
`GroupLayout lmp = new GroupLayout(p);`  
`p.setLayout(lmp);`
- Puis ajout des composants deux fois *sur le gestionnaire*, pas sur le conteneur
  - `GroupLayout.setHorizontalGroup(GroupLayout.Group)`
  - `GroupLayout.setVerticalGroup(GroupLayout.Group)`
  - `GroupLayout.Group.addComponent(Component)`

# Écarts

- *écart* = composant invisible
  - de taille fixe
  - intercalé entre
    - deux composants voisins,
    - ou entre un composant et le bord du conteneur
- Définition des écarts :
  - `GroupLayout.Group.addGap`
  - `GroupLayout.SequentialGroup.addPreferredGap`
  - `GroupLayout.SequentialGroup.addContainerGap`
- Écarts prédéfinis :
  - `GroupLayout.setAutoCreateGaps(boolean auto)`
  - `GroupLayout.setAutoCreateContainerGaps(boolean auto)`



# Retaillage des composants

- `GroupLayout` respecte les tailles min/pref/max des composants
- Ajout de contraintes sur la taille
  - `GroupLayout.Group.addComponent (`  
    `Component c, int min, int pref, int max`  
    `)`
  - constantes prédéfinies
    - `GroupLayout.DEFAULT_SIZE`
      - utiliser la taille correspondante du composant
    - `GroupLayout.PREFERRED_SIZE`
      - utiliser la taille préférée du composant

# Exemples

```
g.addComponent(c, 0, DEFAULT, Short.MAX_VALUE)  
g.addComponent(c, 0, PREFERRED, Short.MAX_VALUE)
```

composant retaillable entre 0 et l'infini,  
de taille préférée la taille préférée du composant

```
g.addComponent(c, DEFAULT, DEFAULT, Short.MAX_VALUE)
```

composant retaillable entre sa taille min et l'infini,  
de taille préférée la taille préférée du composant

```
g.addComponent(c, PREFERRED, DEFAULT, Short.MAX_VALUE)
```

composant retaillable entre sa taille préférée et l'infini,  
de taille préférée la taille préférée du composant

```
g.addComponent(c, PREFERRED, DEFAULT, PREFERRED)
```

composant non retaillable, de taille sa taille préférée

# Lier la taille de plusieurs éléments

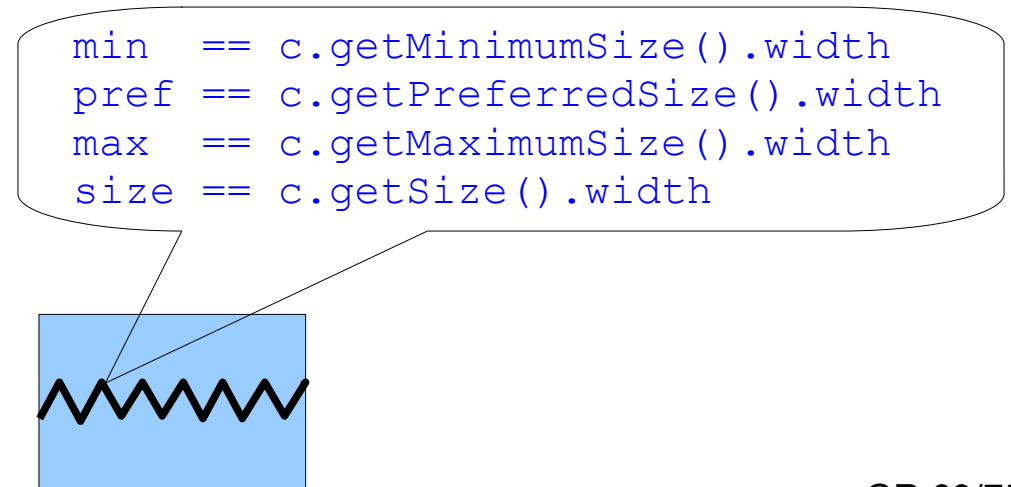
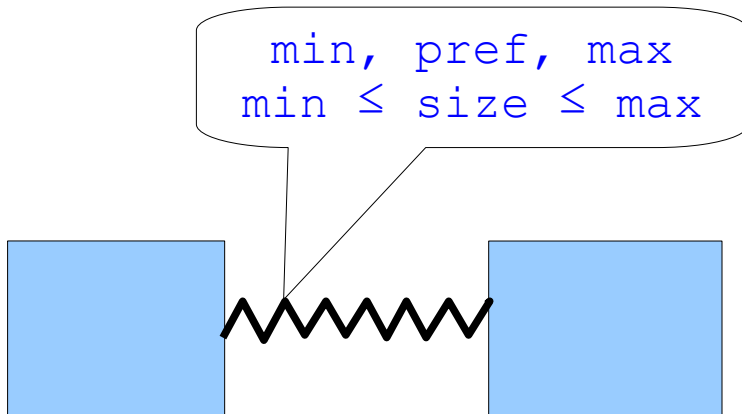
- À l'intérieur d'un même groupe parallèle
  - reporter à l'infini la taille maximale des éléments
  - possibilité de configurer la "retaillabilité" du groupe
    - `GroupLayout.createParallelGroup(  
    int align, boolean resizable  
)`
- Entre deux groupes indépendants
  - `GroupLayout.linkSize(  
    [int axis,] Component... c  
)`

# Modification de la composition du conteneur à l'exécution

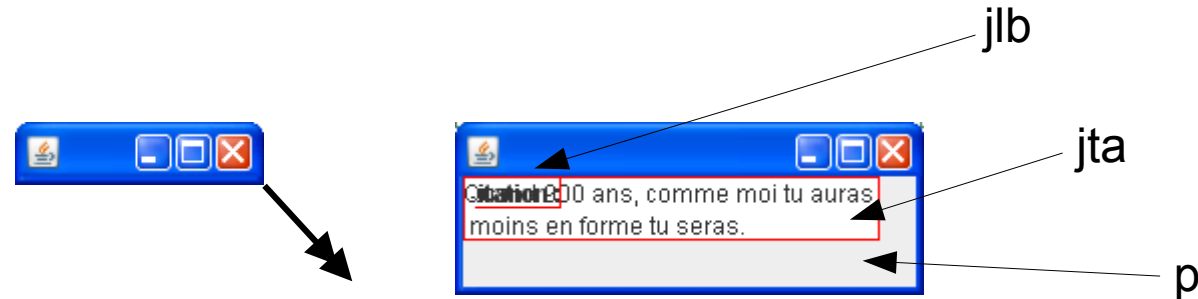
- `GroupLayout.replace(  
    Component oldComp, Component newComp  
)`
  - remplace `oldComp` par `newComp`
- `GroupLayout.setHonorsVisibility(boolean)`
  - l'espace occupé par les composants devenus invisibles doit-il être pris en compte ?
  - possible de raffiner pour chaque composant avec `setHonorsVisibility(Component, boolean)`

# javax.swing.SpringLayout

- Définition de la position et de la taille des composants à l'aide de ressorts (*spring*)
- **ressort** = contrainte entre deux arêtes
  - d'un même composant,
  - ou de deux composants d'un même conteneur,
  - ou d'un composant et de son conteneur

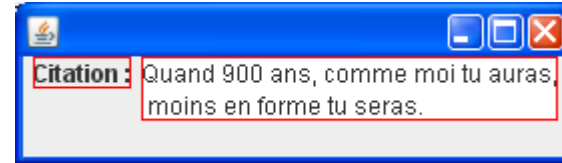
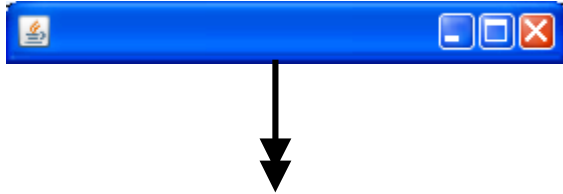


# Exemple



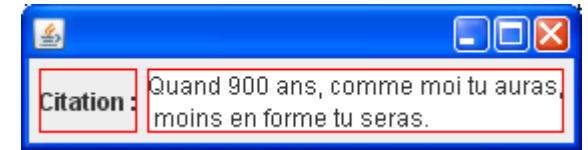
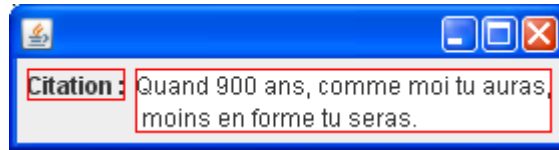
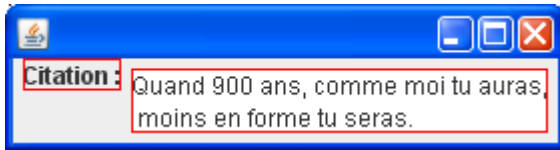
```
JLabel jlb = new JLabel("Citation :");
JTextArea jta = new JTextArea(
    "Quand 900 ans, comme moi tu auras,\n"
    + " moins en forme tu seras."
);
...
SpringLayout slm = new SpringLayout();
JPanel p = new JPanel(slm);
{ //--
    p.add(jlb);
    p.add(jta);
} //--
```

# Répartition horizontale



```
slm.putConstraint(  
    SpringLayout.EAST, p,  
    Spring.constant(5, 5, Short.MAX_VALUE),  
    SpringLayout.EAST, jta  
);  
slm.putConstraint(  
    SpringLayout.EAST, jta,  
    jta.getPreferredSize().width,  
    SpringLayout.WEST, jta  
);  
slm.putConstraint(  
    SpringLayout.WEST, jta,  
    5,  
    SpringLayout.EAST, jlb  
);  
slm.putConstraint(  
    SpringLayout.WEST, jlb,  
    Spring.constant(5, 5, Short.MAX_VALUE),  
    SpringLayout.WEST, p  
);
```

# Répartition verticale

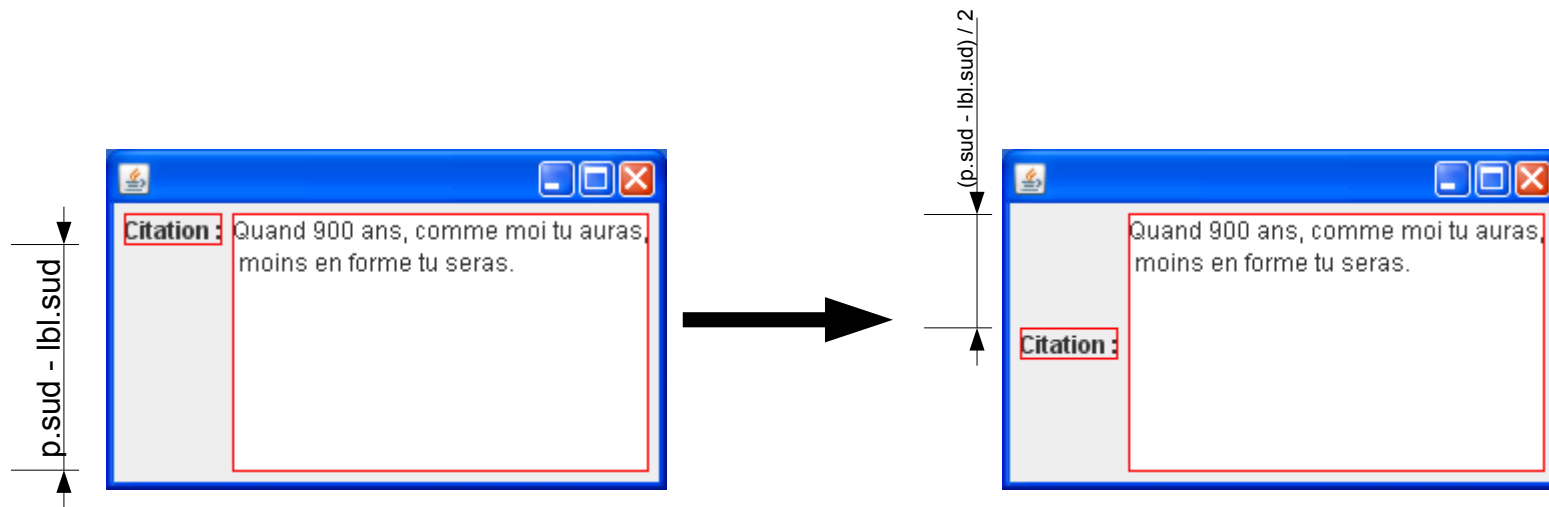


```
slm.putConstraint(  
    SpringLayout.NORTH, jta,  
    5,  
    SpringLayout.NORTH, p  
);  
slm.putConstraint(  
    SpringLayout.SOUTH, p,  
    5,  
    SpringLayout.SOUTH, jta  
);
```

```
slm.putConstraint(  
    SpringLayout.NORTH, jlb,  
    0,  
    SpringLayout.NORTH, jta  
);
```

```
slm.putConstraint(  
    SpringLayout.SOUTH, jlb,  
    0,  
    SpringLayout.SOUTH, jta  
);
```





```

SpringLayout.Constraints cons = slm.getConstraints(jlb);
cons.setY(
    Spring.scale(
        Spring.sum(
            slm.getConstraints(p).getConstraint(SpringLayout.SOUTH),
            Spring.minus(cons.getConstraint(SpringLayout.SOUTH))
        ),
        0.5f
    )
);

```

# Pas de gestionnaire (?)

- Il faut empêcher l'association d'un gestionnaire au conteneur :
  - `JPanel p = new JPanel(null) ;`
  - `p.setLayout(null) ;`
- Il faut donner des positions et des tailles aux composants :
  - `Component.setSize`
  - `Component.setLocation`
  - `Component.setBounds`
- Il faut appeler `repaint` sur chaque composant
- À éviter si la fenêtre est redimensionnable !

# Gestionnaire personnalisé

- Allez voir :
  - <https://docs.oracle.com/javase/tutorial/uiswing/layout/custom.html>

