IUT Lyon 1
l'excellence technologique

R2-02-1. Programmation IHM - Cours 1 (suite)

Développement interfaces utilisateurs en Java

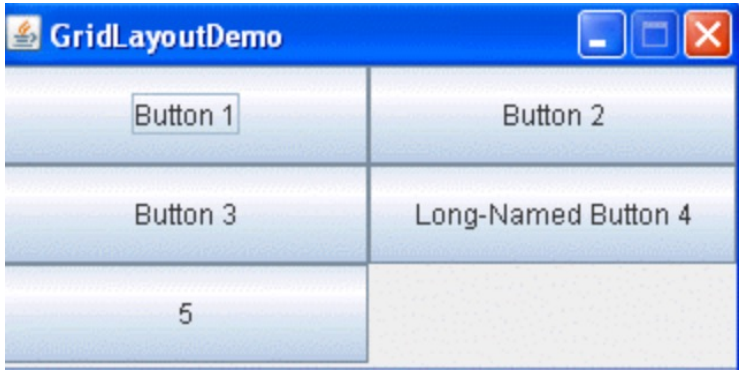
V. DESLANDRES

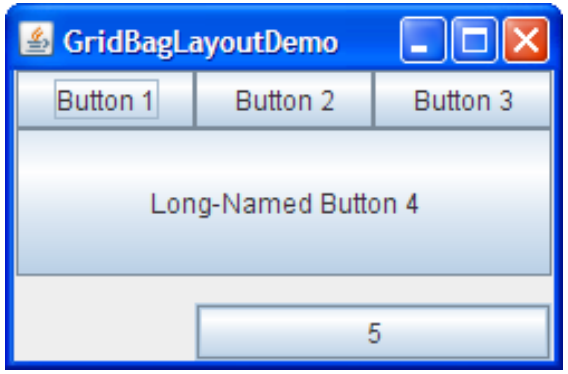
veronique.deslandres@univ-lyon1.fr

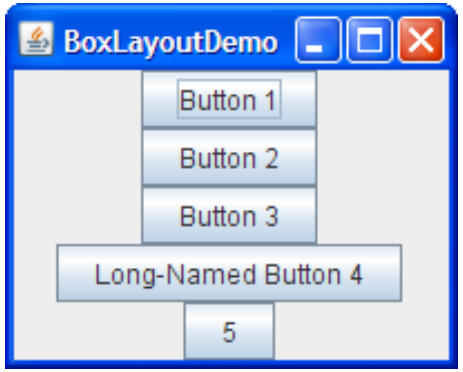
2022/2023

Sommaire de ce cours

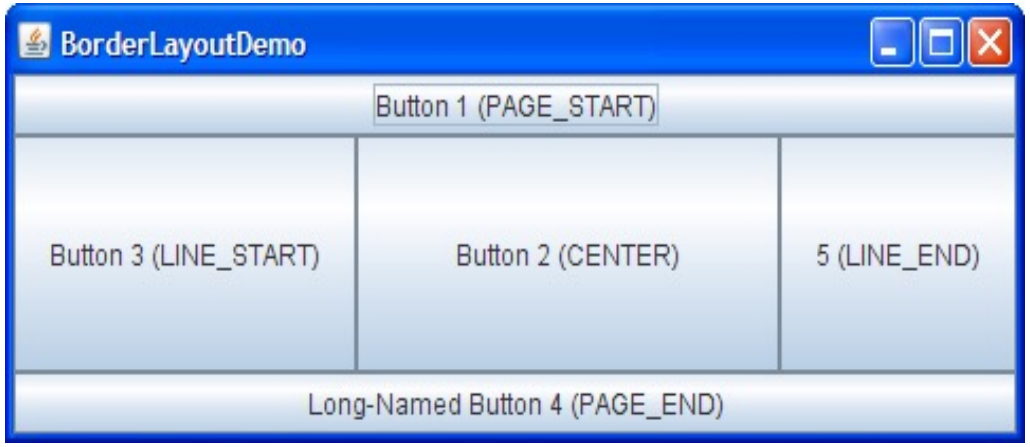
Quelques *layouts* de Swing Java





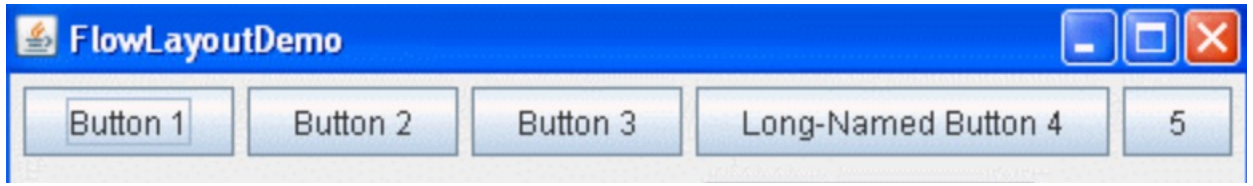


(disposition par défaut dans une `JFrame`)



Les composants sont placés dans des zones d'emplacement prédéfini

(disposition par défaut dans un `JPanel`)



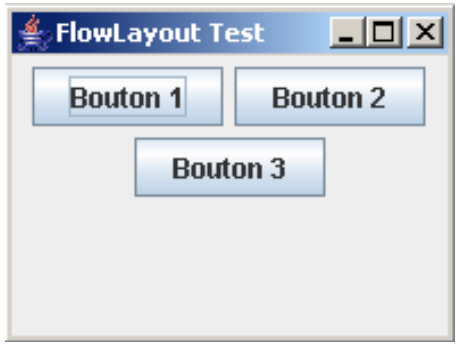
Les composants sont placés les uns à la suite des autres, centrés

Lequel choisir ?

- Tout dépend de l'affichage souhaité
- Souvent on **imbrique des panneaux dans des panneaux**, chacun ayant son propre layout.

- Nota : les composants de Swing ont chacun une taille 'propre' adaptée à leur contenu (texte, icônes, etc.)
- Toutefois certains gestionnaires de disposition (ex. `BorderLayout, GridLayout`) se permettent de modifier cette taille par défaut pour afficher les composants en fonction d'autres critères (comme la hauteur de ligne pour `GridLayout`).

Boutons en `FlowLayout` :



Boutons en `GridLayout` :

Layout : répartir les composants dans la fenêtre

L'intérêt principal des *layouts* permet de bien gérer les **redimensionnements de la fenêtre**.

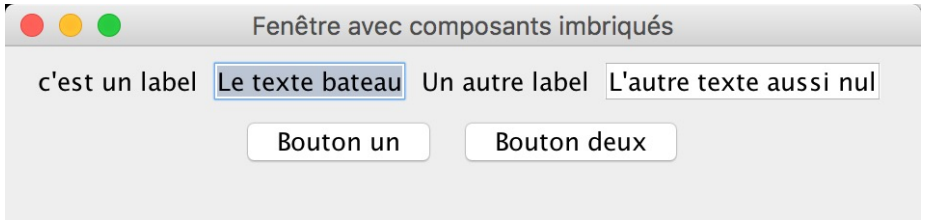
Généralités :

- Les composants sont **toujours placés dans** un conteneur
  - Soit **celui de la fenêtre**, sinon dans un **panneau (`JPanel`)**
- On peut définir *n* panneaux sur une même fenêtre, comme dans l'exemple Fenêtre Etudiant :
  - Panneau des nom/prénom
  - Panneau des boutons, etc.
- NOTA : on peut rendre un panneau **visible** ou pas **dynamiquement**
- Tous les containers de haut niveau (`JFrame, JWindow,…`) ont un *contentPane* qui contiendra les éléments :
  - `cp = frame.getContentPane(); cp.add(leBouton);`
- On peut **convertir** le conteneur de la fenêtre en **panneau** par :
  - `JPanel c =(JPanel)getContentPane();`

par défaut, retourne un `Container (AWT)`

FlowLayout (1/3)

- Place les composants *les uns à la suite des autres*, de façon centrée
  - ligne par ligne, en passant à la ligne suivante si nécessaire
  - C'est le layout par défaut des `JPanel`

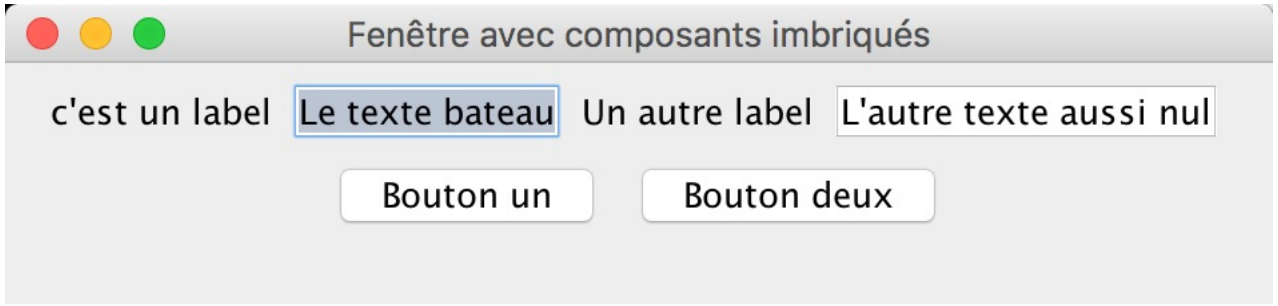


- C'est une classe, avec 3 constructeurs :
  - `FlowLayout()`
  - `FlowLayout(int align)`
    - Paramètre d'alignement (`FlowLayout.LEFT` ou `RIGHT` ou `CENTER`) *(CENTER par défaut)*
  - `FlowLayout(int align, int hgap, int vgap)`
    - hgap est l'espacement horizontal (d'une colonne à une autre)
    - vgap est l'espacement vertical (d'une ligne à une autre)

- On définit le mode de disposition du panneau avec `setLayout()` :
  - `panel.setLayout(new FlowLayout());`
- Puis on transmet le **nom** du composant à ajouter à la méthode `add` :
  - `panel.add(bouton1);`

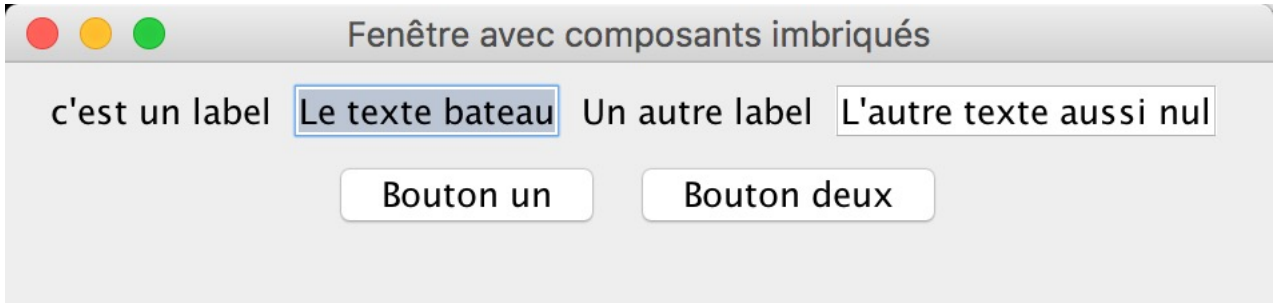
FlowLayout (2/3)

```
public class FenFlowLayout1 extends JFrame {

    private JLabel label1, label2;
    private JTextField texte1, texte2;
    private JButton bouton1, bouton2;

    // Constructeur
    public FenFlowLayout1() {
        initComponents();
        setTitle("Fenêtre avec composants imbriqués");
        //this.setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```



FlowLayout (3/3)

```
private void initComponents() {

    // Initialisation des composants
    label1 = new JLabel("c'est un label");
    label2 = new JLabel("Un autre label");
    bouton1 = new javax.swing.JButton("Bouton un");
    bouton2 = new javax.swing.JButton("Bouton deux");
    texte1 = new JTextField("Le texte bateau");
    texte2 = new JTextField("L'autre texte aussi nul");

    // on récupère le contentPane de la fenêtre
    JPanel cp = (JPanel) this.getContentPane();
    cp.setLayout( new FlowLayout());
    // necessaire car cast de JPanel ==> pas de layout par defaut

    cp.add(label1);
    cp.add(texte1);
    cp.add(label2);
    cp.add(texte2);
    cp.add(bouton1);
    cp.add(bouton2);

}
```



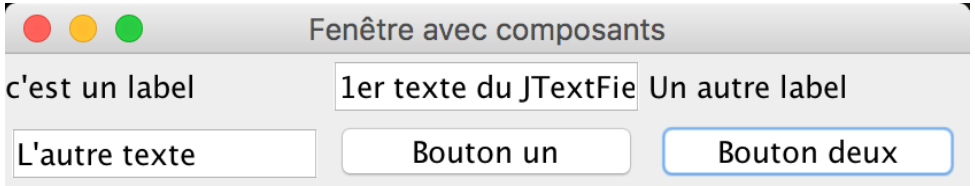
Appel au constructeur FlowLayout() avec des paramètres :

```
// on récupère le contentPane de la fenêtre
JPanel cp = (JPanel) this.getContentPane();
cp.setLayout( new FlowLayout(2, 20, 5));
```

2 ou FlowLayout.RIGHT

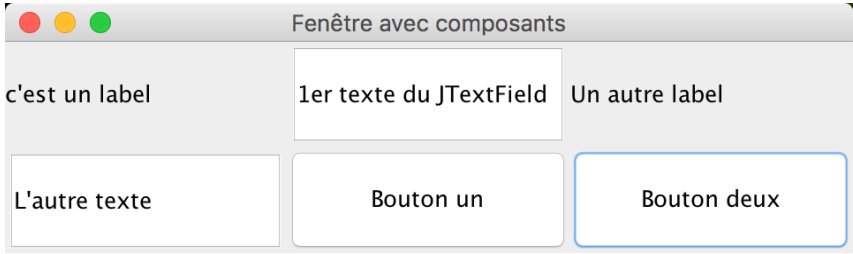
20 pixels : espacement horizontal
5 : espacement vertical

GridLayout (1/2)



- Place les composants dans une grille
  - Chaque composant occupe un espace (cellule) de **même dimension**
- Possède 3 constructeurs :
  - `GridLayout()` : crée une colonne par composant sur **une seule** ligne
  - `GridLayout(int rows, int col)`
    - *rows* = nombre de lignes et *col* = nombre de colonnes
    - *Un des 2 paramètres à 0* : adapte automatiquement en fonction de l'autre param.
  - `GridLayout(int rows, int col, int hgap, int vgap)`

En cas de redimensionnement :

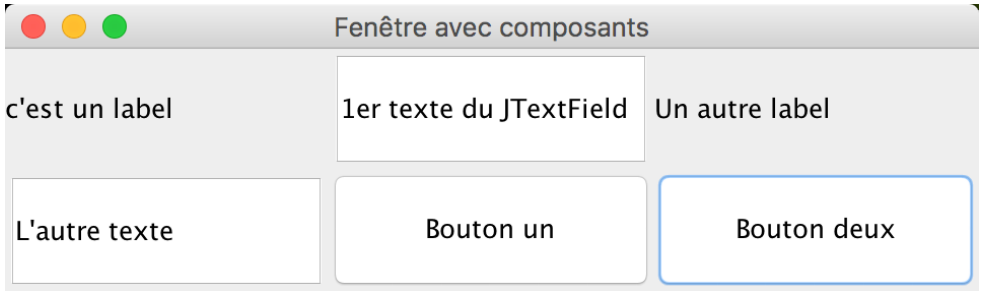


Toutes les cellules occupent le même espace.

- On définit le mode de disposition et on transmet le nom du composant à ajouter:
  - `panel.setLayout(new GridLayout(2,3));`
  - `panel.add(bouton1);` ou `panel.add(bouton1,1,1);` // ligne 1, colonne 1

GridLayout(2/2)

```
private void initComponents() {

    // Initialisation des composants
    label1 = new JLabel("c'est un label");
    label2 = new JLabel("Un autre label");
    bouton1 = new javax.swing.JButton("Bouton un");
    bouton2 = new javax.swing.JButton("Bouton deux");
    texte1 = new JTextField("1er texte du JTextField");
    texte2 = new JTextField("L'autre texte");

    // on récupère le contentPane de la fenêtre
    JPanel cp = (JPanel) this.getContentPane();
    cp.setLayout( new GridLayout(2, 3));

                    Ou bien :   cp.setLayout( new GridLayout(2, 3, 20, 10));

    cp.add(label1);
    cp.add(texte1);
    cp.add(label2);
    cp.add(texte2);
    cp.add(bouton1);
    cp.add(bouton2);
}
```



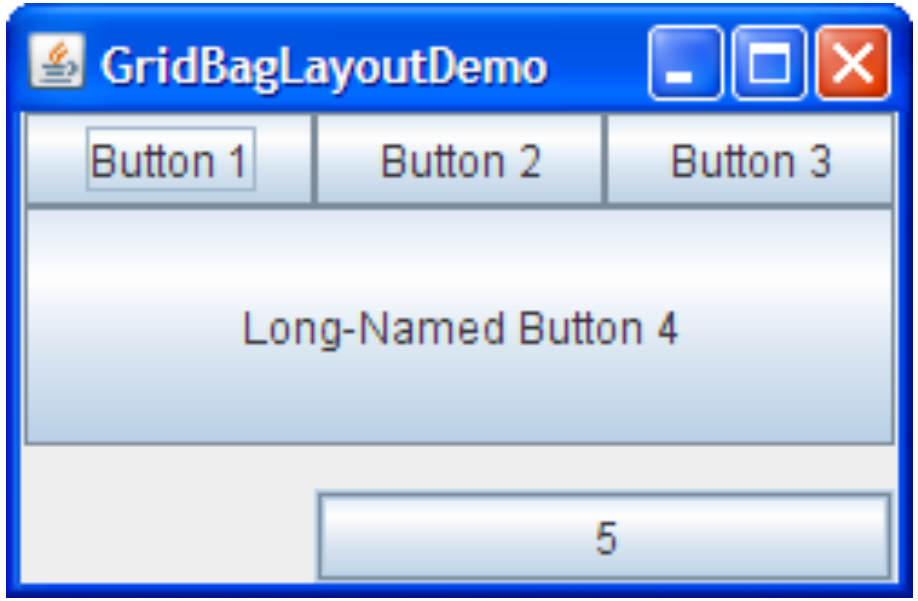
2 lignes et 3 colonnes

*2 lignes et 3 colonnes espacées de 20 pixels entre les colonnes (**hgap**) et de 10 pixels entre les lignes (**vgap**)*

GridBagLayout

- Il existe un autre *layout* qui permet d'avoir des composants qui occupent plusieurs cellules de la grille (le *spanning*)
- Classes `GridBagLayout` et `GridBagConstraints` : on définit les contraintes sur chaque composant à placer

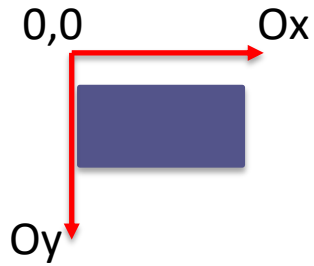
https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html



Absolute layout (1/3)

- Pour placer les composants à un endroit précis, il faut d'abord indiquer **qu'on n'utilise pas** de *LayoutManager* :
  - `ObjetConteneur.setLayout(null);`

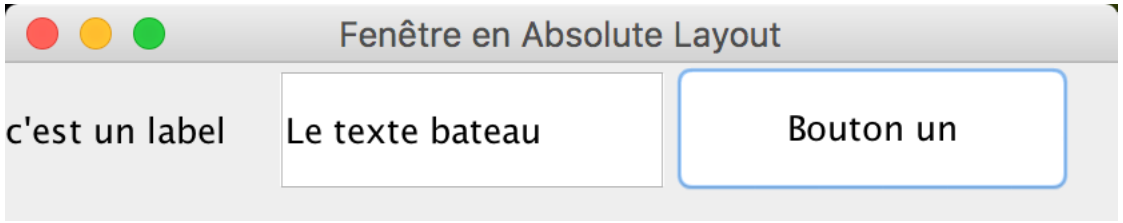
- On indique alors les coordonnées et la taille de chaque composant :
  - `Composant.setBounds(x,y,larg,haut);`
    - `x` et `y` sont les coordonnées du point en haut à gauche du conteneur
    - `larg` est la largeur du composant (axe des `x`)
    - `haut` est la hauteur du composant (axe des `y`)



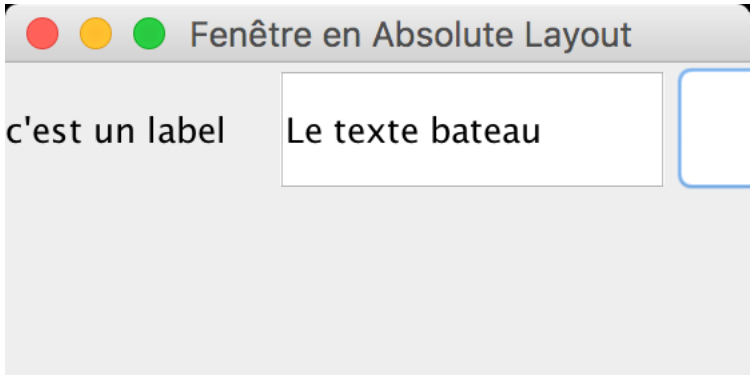
- LIMITES :
  - Les composants **ne sont pas redimensionnés** en cas de modification de la taille de la fenêtre
  - En cas de résolutions d'écran plus petites : le composant peut ne pas apparaître !

Absolute Layout (2/3)

```
private void initComponents() {

    // Initialisation des composants
    label1 = new JLabel("c'est un label");
    label2 = new JLabel("Un autre label");
    bouton1 = new javax.swing.JButton("Bouton un");
    bouton2 = new javax.swing.JButton("Bouton deux");
    texte1 = new JTextField("Le texte bateau");
    texte2 = new JTextField("L'autre texte aussi nul");

    // on récupère le contentPane de la fenêtre
    JPanel cp = (JPanel) this.getContentPane();
    cp.setLayout( null );

    label1.setBounds(0, 0, 100, 50);
    cp.add(label1);
    texte1.setBounds(100, 0, 150, 50);
    cp.add(texte1);
    bouton1.setBounds(250, 0, 150, 50);
    cp.add(bouton1);

}
```



En cas de redimensionnement :

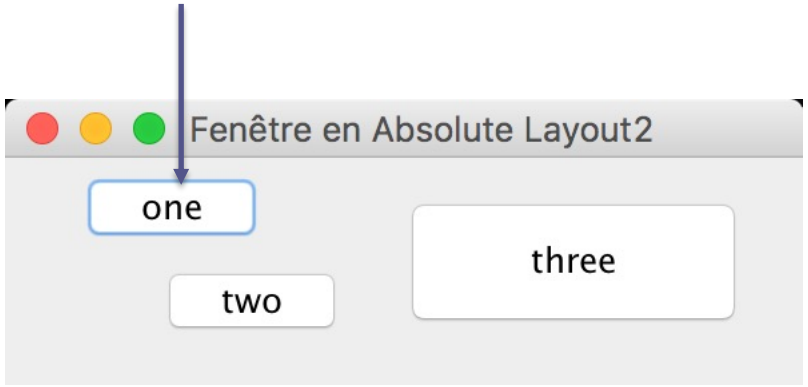


Absolute Layout (3/3)

On peut aussi exploiter la taille de la fenêtre et placer les composants par rapport aux côtés

```
private void initComponents() {
    b1 = new JButton("one");
    b2 = new JButton("two");
    b3 = new JButton("three");

    JPanel cp = (JPanel) this.getContentPane();
    cp.setLayout(null);

    cp.add(b1);
    cp.add(b2);
    cp.add(b3);

    Insets insets = cp.getInsets();

    Dimension size = b1.getPreferredSize();
    b1.setBounds(25 + insets.left, 5 + insets.top,
            size.width, size.height);

    size = b2.getPreferredSize();
    b2.setBounds(55 + insets.left, 40 + insets.top,
            size.width, size.height);

    size = b3.getPreferredSize();
    b3.setBounds(150 + insets.left, 15 + insets.top,
            size.width + 50, size.height + 20);
}
```

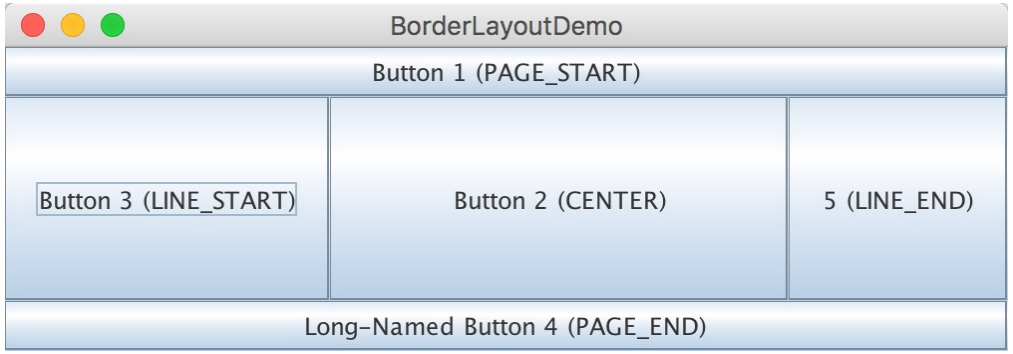
Ici b1 est placé à 25 pixels du bord G et à 5 pixels du haut de la fenêtre



BorderLayout (1/4)

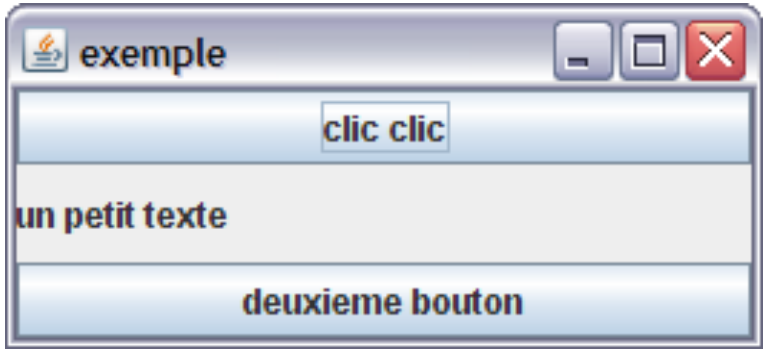
- Le *layout* par défaut d'une `JFrame`

- Découpe l'écran en 5 régions *(anciens noms)* :
  - **`PAGE_END` *(south)* `PAGE_START` *(north)***
  - **`LINE_START` *(east)* `LINE_END` *(west)***
  - **`CENTER`**



- On peut n'en utiliser que certaines, mais tout l'espace sera occupé

- Possède 2 constructeurs :
  - `BorderLayout()`
  - `BorderLayout(int hgap, int vgap)`



- Définition de la disposition en BorderLayout :

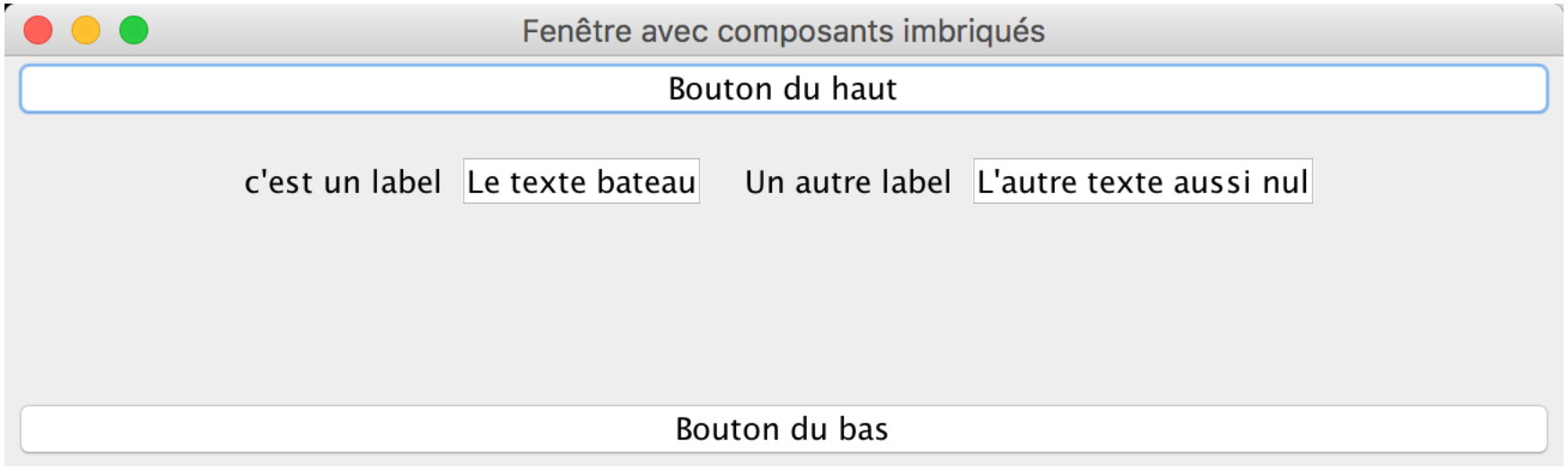
```
c.setLayout(new BorderLayout(10,5));
```

- On transmet à la méthode `add` le **nom** du composant, et **l'emplacement** souhaité :
  - `c.add(bouton1, BorderLayout.PAGE_START);`

- Il n'y a **qu'un seul composant** par région 

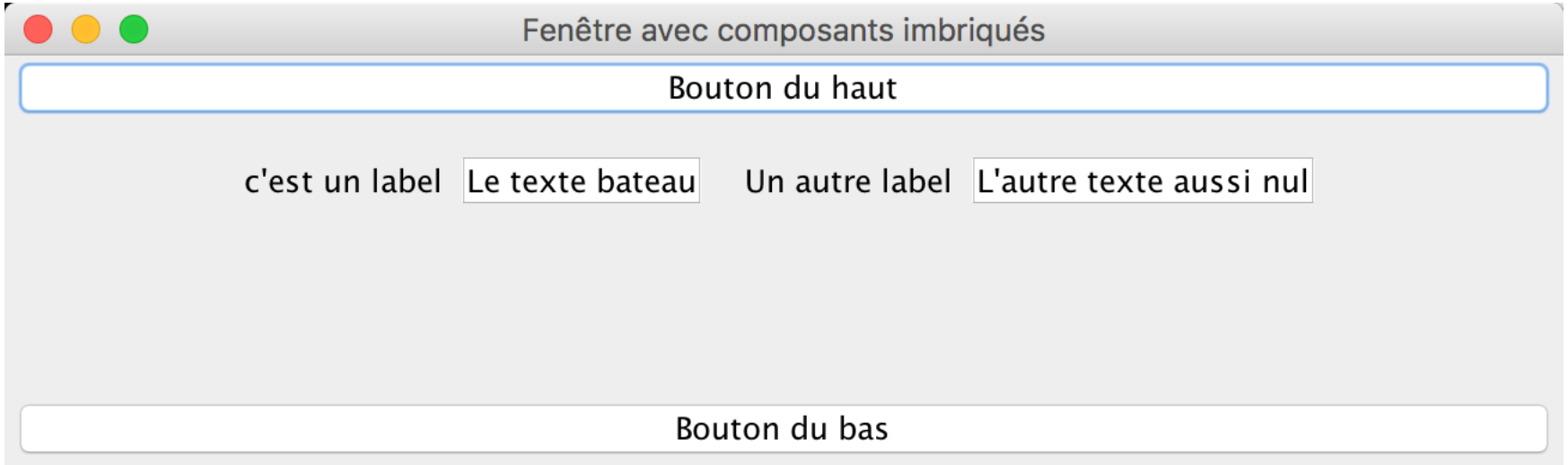
BorderLayout (2/4)

```
public class FenBorderLayout extends JFrame {

    private JLabel lblGauche, lblDroit;
    private JTextField texte1, texte2;
    private JButton enHautBouton, enBasBouton;

    // Constructeur
    public FenBorderLayout() {
        initComponents();
        setTitle("Fenêtre avec composants imbriqués");
        //this.setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     }
```



BorderLayout (3/4)

```
private void initComponents() {

    // Initialisation des composants
    lblGauche = new JLabel("c'est un label");
    lblDroit = new JLabel("Un autre label");
    enHautBouton = new javax.swing.JButton("Bouton du haut");
    enBasBouton = new javax.swing.JButton("Bouton du bas");
    texte1 = new JTextField("Le texte bateau");
    texte2 = new JTextField("L'autre texte aussi nul");

    // on récupère le contentPane de la fenêtre
    JPanel cp = (JPanel) this.getContentPane();
    cp.setLayout( new BorderLayout() );
```

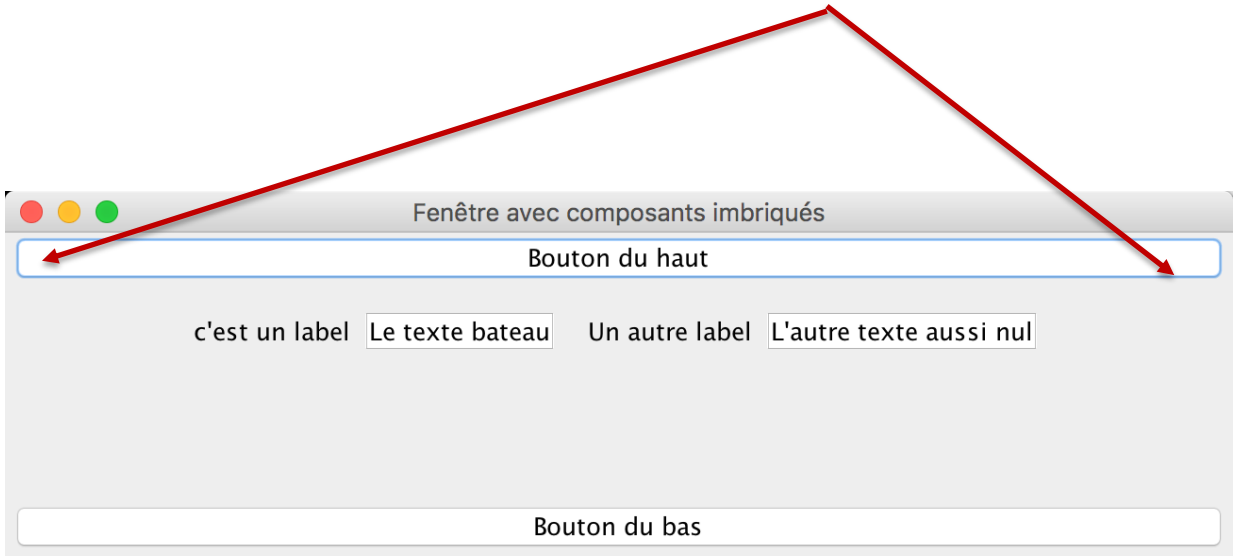


BorderLayout (4/4)

(suite de initComponents())

```
// Besoin de panneaux intermédiaires :
JPanel pannGr1, pannGr2, pannCentre;
// on définit les autres panneaux :
pannGr1 = new JPanel(); // par défaut en FlowLayout Centré
pannGr1.add(lblGauche);
pannGr1.add(texte1);

pannGr2 = new JPanel();
pannGr2.add(lblDroit);
pannGr2.add(texte2);

pannCentre = new JPanel();
pannCentre.add(pannGr1);
pannCentre.add(pannGr2);

cp.add(enHautBouton, BorderLayout.PAGE_START);
cp.add(pannCentre, BorderLayout.CENTER);
cp.add(enBasBouton, BorderLayout.PAGE_END);

}
```

NOTA : ici pas de panneaux latéraux : le CENTER occupe toute la place



Panneaux imbriqués

- Pour mettre **plus d'un composant** dans une zone, on utilise un **panneau intermédiaire**
  - placé dans la zone,
  - dans lequel on positionne les composants.

- Un *container* peut être inséré dans un autre *container*, souvent on ajoute des **JPanel**

  C'est la même méthode que pour **ajouter un composant,** on utilise `add` :

  - `unPanel.add(unAutrePanel);`

Ex.: BorderLayout imbriqués

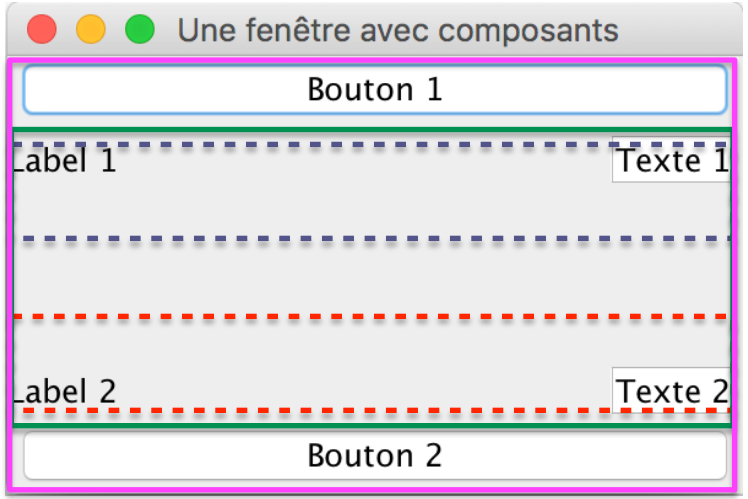
```
public class PanneauxImbriquésBorder extends JFrame{
    JLabel label1, label2;
    JTextField text1, text2;
    JButton bouton1, bouton2;
    JPanel panel1, panel2, panel3, panel4;
    public PanneauxImbriquésBorder(){
        //Création des composants
        label1=new JLabel("Label 1");
        label2=new JLabel("Label 2");
        text1=new JTextField("Texte 1");
        text2=new JTextField("Texte 2");
        bouton1=new JButton("Bouton 1");
        bouton2=new JButton("Bouton 2");
        panel1=(JPanel) getContentPane();
        panel1.setLayout(new BorderLayout());
        panel2= new JPanel(new BorderLayout());
        panel3= new JPanel(new BorderLayout());
        panel4= new JPanel(new BorderLayout());
        //Ajout des composants dans panel2
        panel2.add(label1, BorderLayout.WEST);
        panel2.add(text1, BorderLayout.EAST);
        //Ajout des composants dans panel3
        panel3.add(label2, BorderLayout.WEST);
        panel3.add(text2, BorderLayout.EAST);
        //Ajout des composants dans panel4
        panel4.add(panel2, BorderLayout.NORTH);
        panel4.add(panel3, BorderLayout.SOUTH);
        //Ajout des composants dans panel1
        panel1.add(bouton1, BorderLayout.NORTH);
        panel1.add(panel4, BorderLayout.CENTER);
        panel1.add(bouton2, BorderLayout.SOUTH);
    }
    public static void main (String[] args){
        JFrame fen=new PanneauxImbriquésBorder();
        fen.setSize(300,200);
        fen.setTitle("Une fenêtre avec composants");
        fen.setVisible(true);
    }
}
```

Panel 2

Panel 3

Panel 4

Panel 1



Afficher une image

- Pour afficher une image sur un panneau, on utilise un label
- 2 méthodes:
  - En utilisant l'objet `ImageIcon` directement dans la déclaration du label:

```
JLabel lbl_image=new JLabel(new ImageIcon("src/gif/IUT.png"));
```

  - La méthode `setIcon()` qui attend un objet `ImageIcon` :
    - `monlabel.setIcon(new ImageIcon(fichier_image));`

```
JLabel lbl_image=new JLabel();
lbl_image.setIcon(new ImageIcon("src/gif/IUT.png"));
```

    - Sous Windows : comme en Java l'antislash est un marqueur de caractère spécial :'\n', '\t', etc.
      - On peut utiliser le double antislash comme caractère de séparation

        `C:\\Users\\Anna\\workspace\\progIHM\\TP1…`

    - Sous Unix ou MacOS :
      - `monLabel.setIcon(new ImageIcon("src/gif/image1.gif"));`

Redimensionner une image

- Pour redimensionner l'image à afficher dans un label, on va utiliser une instance d'**Image** et de **ImageIcon** :

  – Récupérer l'image depuis son fichier en tant qu'**ImageIcon**, et la transformer en **Image** :

```
ImageIcon userIcon = new ImageIcon("src/main/java/gif/etu-G.jpg");
Image userImage = userIcon.getImage();
```

  – Redimensionner l'image à la bonne taille (ici en pixels) :

```
Image newImage = userImage.getScaledInstance(50, 50,
java.awt.Image.SCALE_SMOOTH);
```

- La redéfinir en ImageIcon en mode *smooth*

```
userIcon = new ImageIcon(newImage);
```

- On peut maintenant utiliser l'**imageIcon** pour le JLabel :

```
lbl_image = new JLabel(userIcon);
```

Ajout d'une image en icône

- Pour affecter une image comme icône de l'application **SOUS WINDOWS**, 3 méthodes sont utilisées:
  - En utilisant la classe `Toolkit` qui contient la méthode `getDefaultToolkit()`. L'objet `Toolkit` contient la méthode `getImage()` qui prend l'URL de l'image et rend un objet de type `Image`.

```
//Affecter une image comme icône de l'application SOUS WINDOWS
Toolkit tk=Toolkit.getDefaultToolkit();
Image im = tk.getImage("src/gif/image.gif");
setIconImage(im);
```

  - En utilisant `ImageIcon`. La méthode `getImage()` retourne une instance image de `ImageIcon`

```
setIconImage(new ImageIcon("Java_logo.png").getImage());
```

  - En utilisant `ImageIO`. La méthode `read()` de la classe `ImageIO` prend un objet `InputStream` qui pointe vers le fichier de l'image :

```
try{
    setIconImage(ImageIO.read(new FileInputStream("Java_logo.png")));
}
catch(Exception e){}
```

Taille d'une fenêtre, d'un composant ou d'un panel

- Mettre une fenêtre au centre :

```
JFrame frame=new JFrame();
frame.setLocationRelativeTo(null);
```

- Faire qu'une fenêtre occupe tout l'écran :

```
GraphicsEnvironment gEnv = GraphicsEnvironment.getLocalGraphicsEnvironment();
Rectangle bounds = gEnv.getMaximumWindowBounds();
frame.setBounds(bounds);
frame.setVisible(true);
```

- Pour redimensionner un panel ou un composant, on utilise la méthode `setPreferredSize() :`

```
lbl_image.setPreferredSize(new Dimension(500,100));

JPanel pan=(JPanel) getContentPane();
pan.setPreferredSize(new Dimension(200,300));
```

Bordure et titre d'un JPanel

- Pour ajouter une bordure à un panneau, il faut utiliser la méthode `setBorder()`

```
JPanel pan=(JPanel) getContentPane();
pan.setBorder(BorderFactory.createTitledBorder("Configure Layout"));
```

- Un exemple avec cette fenêtre :