

Les tables

- JTable affiche des données dans un tableau
- TableModel régit la gestion des données
- On peut fournir les données dans un tableau bidimensionnel d'objets : `Object[][]` et utiliser le `DefaultTableModel`, mais il vaut mieux étendre `AbstractTableModel`.
- La sélection est régie par un modèle de sélection
- De plus, il y a un modèle de colonnes.
- Un tableau est entouré d'ascenseurs, en général.

Les tables : Constructeurs

- `JTable()` modèles par défaut pour les trois modèles
- `JTable(int numRows, int numColumns)` avec autant de cellules vides
- `JTable(Object[][] rowData, Object[] columnNames)` avec les valeurs des cellules de `rowData` et noms de colonnes `columnNames`.
- `JTable(TableModel dm)` avec le modèle de données `dm`, les autres par défaut.
- `JTable(TableModel dm, TableColumnModel cm)` avec modèle de données et modèle de colonnes fournis.
- `JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm)` Les trois modèles sont fournis.
- `JTable(Vector rowData, Vector columnNames)` ici, les données sont fournies par colonne.

Les tables : Exemple

Lines 1–21 / 28

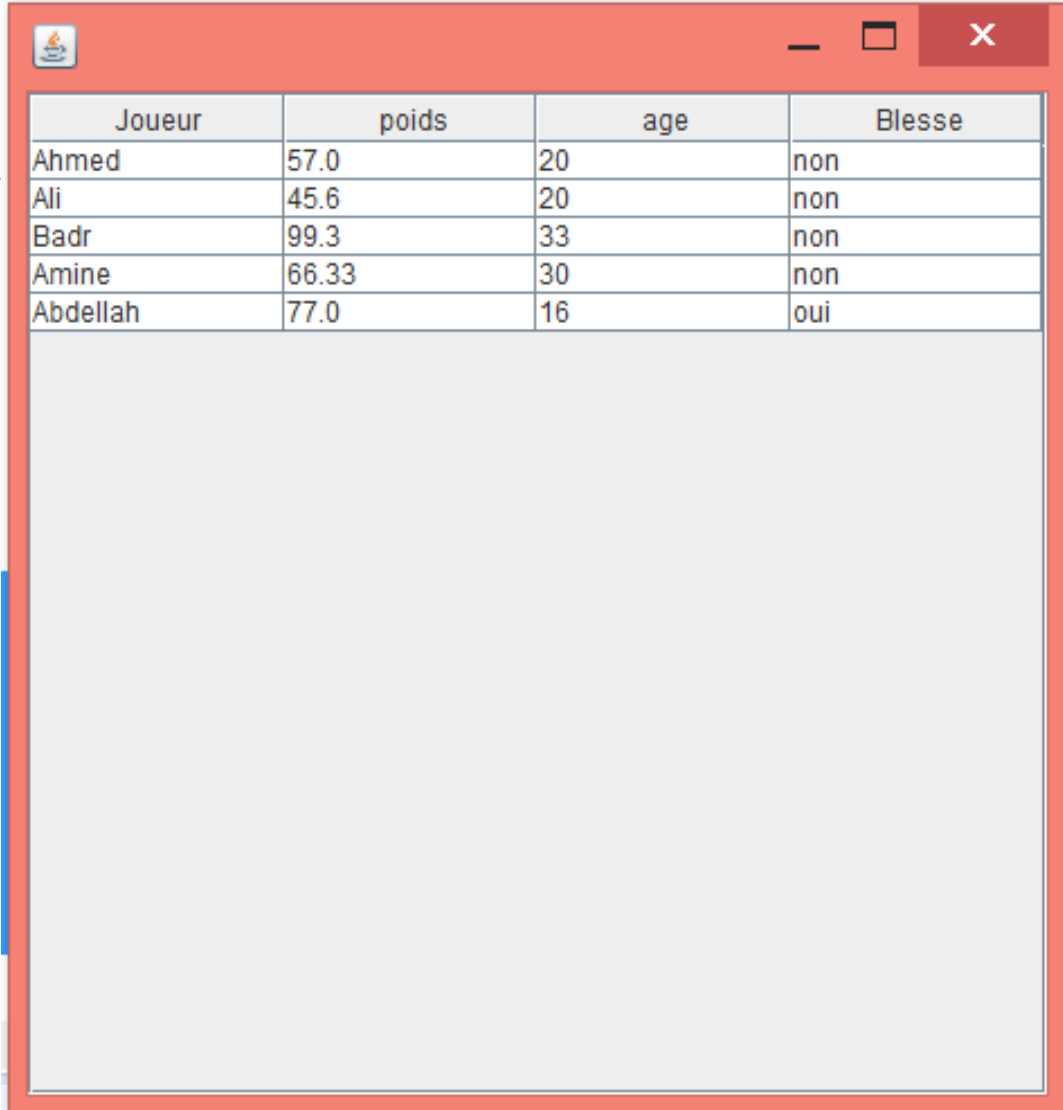
```
1 import java.awt.BorderLayout;
2 import javax.swing. ;
3 public class Table extends JPanel {
4     private Object[][] cellules = {
5         { "Ahmed", new Double(57), new Integer(20), "non"},
6         { "Ali", new Double(45.6), new Integer(20), "non"},
7         { "Badr", new Double(99.3), new Integer(33), "non"},
8         { "Amine", new Double(66.33), new Integer(30), "non"},
9         { "Abdellah", new Double(77), new Integer(16), "oui"}
10    };
11    private String[] columnNames = { "Joueur", "poids", "age", "Blesse"};
12    Table() {
13        setLayout(new BorderLayout());
14        JTable table = new JTable(cellules, columnNames); add(new
15        JScrollPane(table), BorderLayout.CENTER);
16    }
17
18    public static void main(String[] args){
19        JFrame f = new JFrame();
20        f.setContentPane(new Table());
21        f.pack();
```

Les tables : Exemple

Lines 20–40 / 28

```
1         f.setContentPane(new Table());  
2         f.pack();  
3         f.setVisible(true);  
4     }  
5 }
```

Les tables : Exemple



Joueur	poids	age	Blesse
Ahmed	57.0	20	non
Ali	45.6	20	non
Badr	99.3	33	non
Amine	66.33	30	non
Abdellah	77.0	16	oui

Les conteneurs Swing

Conteneurs généraux

➤ JPanel

- ✓ Conteneur opaque à usage générique sous-classe de JComponent
- ✓ Possède un gestionnaire de positionnement de type FlowLayout par défaut

➤ Box

- ✓ Conteneur qui n'hérite pas de JComponent
- ✓ Utilise un gestionnaire de positionnement de type BoxLayout

Gestionnaire de présentation : Objectif

- Tout conteneur qui regroupe des composants doit connaître comment les positionner
 - ✓ LayoutManager est un objet qui calcule les tailles et positions des composants enfants
 - ✓ Considère les contraintes :
 - Des tailles (préférée, min et max) des composants
 - La nature du LayoutManager

Gestionnaire de présentation : Intérêts

- L'utilisation d'un gestionnaire de présentation permet de faire face aux situations suivantes :
 - ✓ Redimensionnement de la fenêtre.
 - ✓ Changement de l'espace occupé par le composant en fonction du look and feel (apparence et comportement) utilisé.

Gestionnaire de présentation : Comment ?

- On associe un `LayoutManager` à un conteneur avec les méthodes : `setLayout/getLayout`
- Suppression d'un `LayoutManager` avec `setLayout(null)`.
- Le `LayoutManager` d'un `JPanel` est par défaut de type `FlowLayout`
- On peut aussi affecter un `LayoutManager` lors de la construction du `JPanel` :

```
1 JPanel p=new JPanel(new BorderLayout());
```

Gestionnaire de présentation : Fonctionnement

- les JComponent ont une taille nulle par défaut, il faut donc leur donner explicitement une taille (préférée ou non)
- La méthode pack() de JFrame demande à chaque composant sa taille préférée. La JFrame s'ajuste en fonction du résultat retourné.

Placement des composants

- Tout composant Swing possède :
 - ✓ une position par rapport au parent : set/getLocation, set/getX, set/getY
 - ✓ une taille : set/getWidth, set/getHeight, set/getSize
- la zone couverte peut aussi être manipulée directement avec : set/getBounds

Les LayoutManager

On distingue deux familles de gestionnaires de positionnement :

➤ LayoutManager sans contrainte :

- ✓ FlowLayout
- ✓ GridLayout
- ✓ BoxLayout

➤ LayoutManager avec contrainte :

- ✓ BorderLayout
- ✓ GridBagLayout
- ✓ GroupLayout

FlowLayout

C'est le layout manager par défaut des JPanel

affiche les composants à leur taille préférée, "de la gauche vers la droite", et reviens à la ligne si nécessaire

L'ordre des composants est celui de leur ajout dans le container

1 `FlowLayout(int align, int hgap, int vgap)`

2 `align`=alignement sur chaque ligne: `FlowLayout.LEADING`, /

3\ `FlowLayout.CENTER` ou `FlowLayout.TRAILING`

`hgap/vgap`: espace entre les composants

FlowLayout

Lines 1–21 / 28

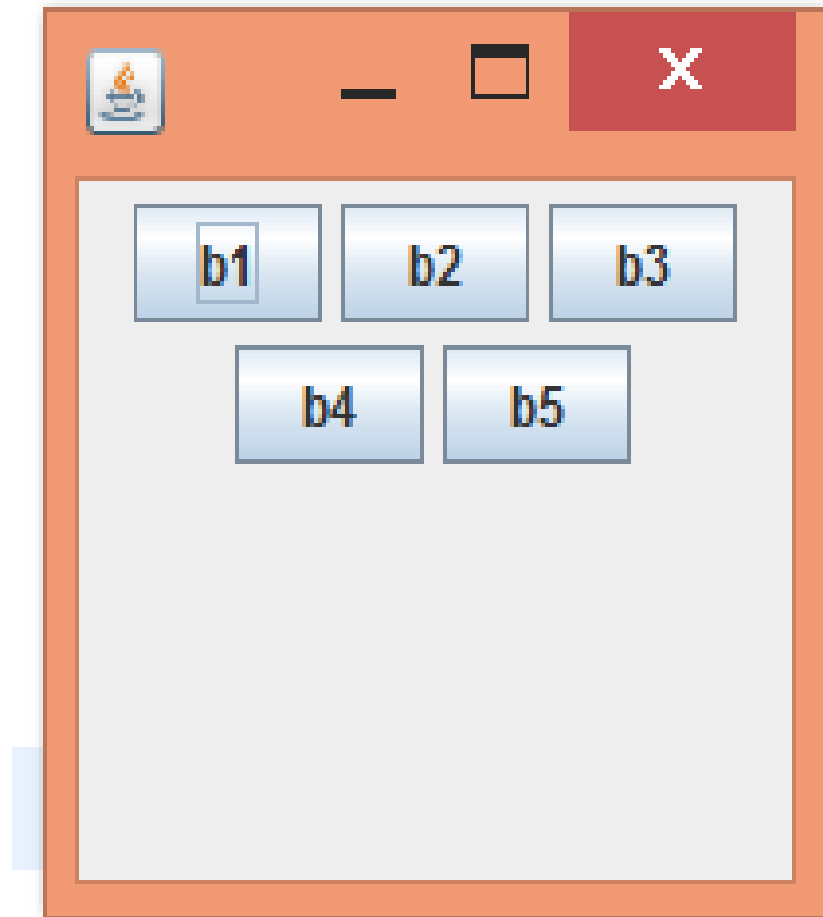
```
1 import java.awt.* ;
2
3 import javax.swing.* ;
4 public class FL extends JFrame{
5     protected JButton b1,b2,b3,b4,b5;
6     public FL() {
7         b1=new JButton("b1");
8         b2=new JButton("b2");
9         b3=new JButton("b3");
10        b4=new JButton("b4");
11        b5=new JButton("b5");
12
13        JPanel p =new JPanel(new FlowLayout());
14
15        p.add(b1);
16        p.add(b2);
17        p.add(b3);
18        p.add(b4);
19        p.add(b5);
20
21        setContentPane(p);
```

FlowLayout

Lines 20–40 / 28

```
1
2         setContentPane (p) ;
3         setSize (200,200) ;
4         setVisible (true) ;
5     }
6     public static void main (String[] args) {
7         FL f=new FL () ;
8     }
9 }
```


FlowLayout



GridLayout

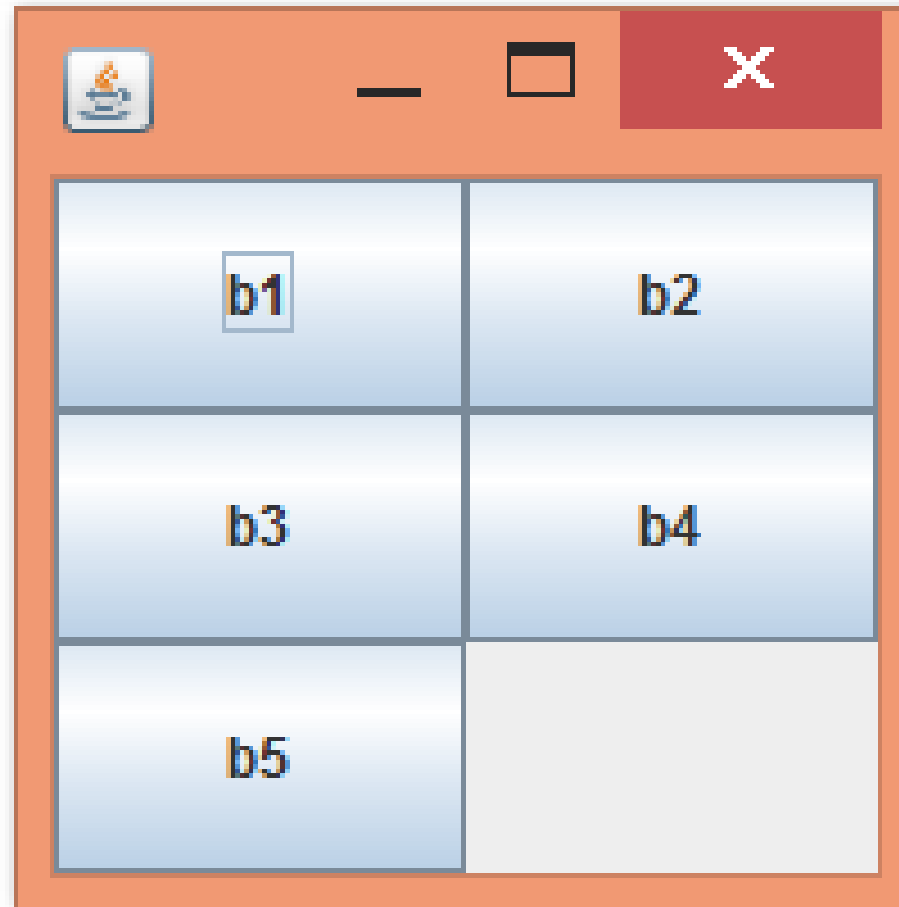
- Les composants sont distribués dans une grille conformément à leur ordre d'ajout au conteneur.
- Les composants sont affichés et ils ont tous la même taille.
- Le layout s'adapte si le nombre de composants que de cases.

1 GridLayout(`int` rows,`int` cols,`int` hgap,`int` vgap)

2 - rows/columns=lignes/colonnes

3 - hgap/vgap: espace entre les composants

GridLayout



BoxLayout

- Affiche les composants en ligne ou en colonne, selon leur taille préférée
- L'ordre d'affichage est l'ordre d'ajout dans le container

```
1 // On commence par specifier un layout null
2 JPanel p=new JPanel(null);
3 p.setLayout(new BoxLayout(p,BoxLayout.Y_AXIS));
4 c.setAlignmentX(0f);
5 / 0f alignement a gauche
6 1f alignement a droite
7 0.5f alignement au centre
8 /
```

LayoutManager avec contraintes

Ajout de composant avec une contrainte de placement :

```
1 add(java.awt.Component, Object constraint)
```

BorderLayout

- C'est le Layout par défaut du contentPane d'une JFrame
- 5 zone de placement disponibles :
 - ✓ BorderLayout.NORTH
 - ✓ BorderLayout.SOUTH
 - ✓ BorderLayout.EAST
 - ✓ BorderLayout.WEST
 - ✓ BorderLayout.CENTER (valeur par défaut)
- 1 seul composant par zone de placement

BorderLayout

Lines 1–21 / 92

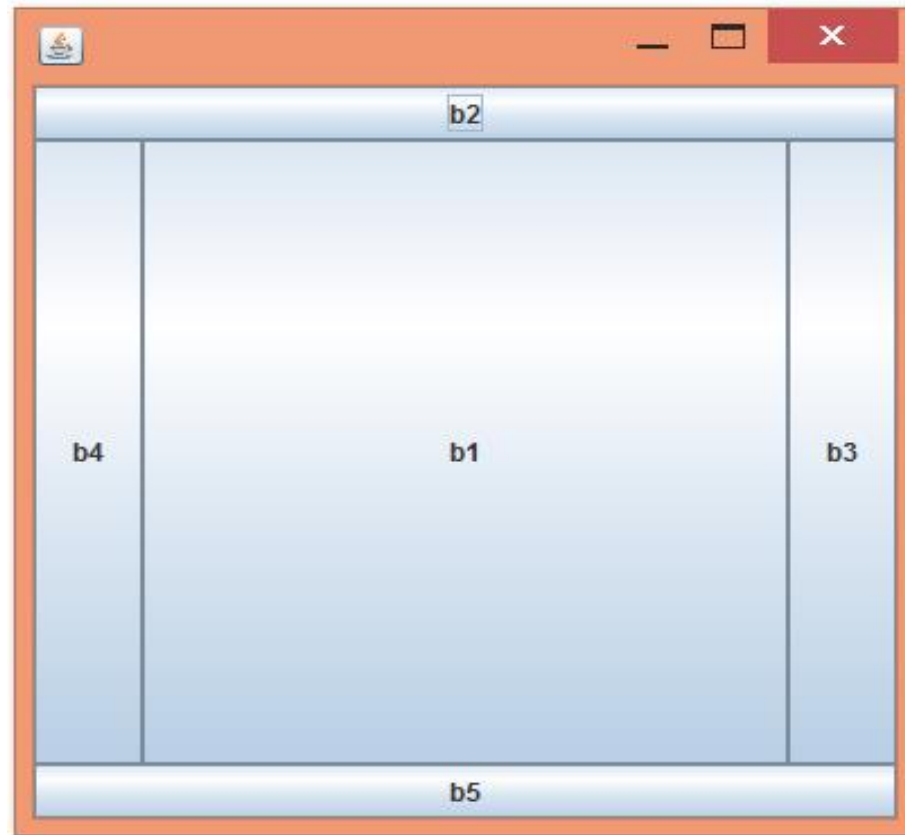
```
1 import java.awt.BorderLayout;
2
3 import javax.swing. ;
4 public class BL extends JFrame{
5     protected JButton b1,b2,b3,b4,b5;
6     public BL() {
7         b1=new JButton("b1");
8         b2=new JButton("b2");
9         b3=new JButton("b3");
10        b4=new JButton("b4");
11        b5=new JButton("b5");
12
13
14
15        JPanel p =new JPanel(new BorderLayout());
16        p.add(b1, BorderLayout.CENTER);
17        p.add(b2, BorderLayout.NORTH);
18        p.add(b3, BorderLayout.EAST);
19        p.add(b4, BorderLayout.WEST);
20        p.add(b5, BorderLayout.SOUTH);
21        setContentPane(p);
```

BorderLayout

Lines 20–40 / 92

```
1
2         setContentPane (p) ;
3         setSize (400,400) ;
4         setVisible (true) ;
5     }
6     public static void main(String[] args) {
7         BL f=new BL() ;
8     }
9 }
```


BorderLayout



GridBagLayout

- Positionnement des composants selon une grille, dans l'ordre d'ajout, et en tenant compte de contraintes sur :
 - ✓ la zone occupée par un composant
 - ✓ le placement d'un composant dans sa zone
 - ✓ le comportement de la zone en cas de redimensionnement

GridBagLayout

- Les contraintes sont exprimées par une instance de `GridBagConstraints`.
- Comme `add` copie la contrainte, on peut réutiliser le même objet pour plusieurs contraintes, en ne modifiant que le nécessaire :

```
1 GridBagConstraints gbc=new GridBagConstraints();
2 gbc.gridwidth=3;

3 gbc.fill=GridBagConstraints.BOTH;
4 gbc.weightx=1f;

5 gbc.weighty=1f;
6 p.add(new JButton("J'occupe trois cases"),gbc);

7 gbc.gridwidth=GridBagConstraints.REMAINDER;
8 p.add(new JButton("J'occupe une seule case"),gbc);
```

GridBagLayout

- Zone occupée par un composant=portion rectangulaire de la grille
- Les cases n'ont pas toutes la même taille
- Beaucoup de propriétés donc plus d'options de configurations que le GridLayout.

GridBagLayout

- gridx et gridy définissent les coordonnées de la cellule dans la zone d'affichage.
 - ✓ GridBagConstraints.RELATIVE : le composant sera placé dans la cellule de droite (gridx) ou dans la cellule en-dessous (gridy) du composant précédent.
- gridwidth : le nombre de cases en colonnes du composant courant.
- gridheight : le nombre de cases en en lignes du composant courant.
- Les constantes usivantes peuvent être utilisés avec gridwidth et gridheight :
 - ✓ GridBagConstraints.REMAINDER Le composant est le dernier de sa ligne ou de sa colonne.
 - ✓ GridBagConstraints.RELATIVE Le composant est l'avant-dernier de sa ligne ou de sa colonne.

GridBagLayout

- anchor permet de définir le point d'ancrage d'un composant dans la ou les cellules qu'il occupe.
- fill, détermine comment utiliser l'espace disponible lorsque la taille du composant est inférieure à celle qui lui est offerte :
 - ✓ GrigBagConstraint.NONE : ne pas redimensionner le composant.
 - ✓ GrigBagConstraint.HORIZONTAL : remplir l'espace horizontal offert.
 - ✓ GrigBagConstraint.VERTICAL : remplir l'espace vertical offert.
 - ✓ GrigBagConstraint.BOTH : remplir l'espace offert, horizontalement et verticalement.
- weightx et weighty déterminent comment se répartit l'espace supplémentaire entre les composants.
- ipadx et ipady définissent les marges internes minimales du composant.

GridBagLayout

Lines 1–21 / 20

```
1  import java.awt.GridBagConstraints;
2  import java.awt.GridBagLayout;
3  import java.awt.Insets;
4
5  import javax.swing.JButton;
6  import javax.swing.JFrame;
7  import javax.swing.JPanel;
8
9  public class GridBagLayoutDemo extends JFrame {
10     final static boolean shouldFill = true;
11     final static boolean shouldWeightX = true;
12
13
14     public GridBagLayoutDemo() {
15         JButton button;
16         JPanel pane= new JPanel();
17         pane.setLayout(new GridBagLayout());
18         GridBagConstraints c = new GridBagConstraints();
19
20         c.fill = GridBagConstraints.HORIZONTAL;
21
22         button = new JButton("Button_1");
```

GridBagLayout

Lines 20–40 / 20

```
1
2     button = new JButton("Button_1");
3     if (shouldWeightX) {
4         c.weightx = 0.5;
5     }
6     c.gridx = 0;
7     c.gridy = 0;
8     pane.add(button, c);
9
10    button = new JButton("Button_2");
11    c.gridx = 1;
12    c.gridy = 0;
13    pane.add(button, c);
14
15    button = new JButton("Button_3");
16    c.gridx = 2;
17    c.gridy = 0;
18    pane.add(button, c);
19
20    button = new JButton("Long-Named_Button_4");
21    c.ipady = 40;    //make this component tall
```


GridBagLayout

Lines 39–59 / 20

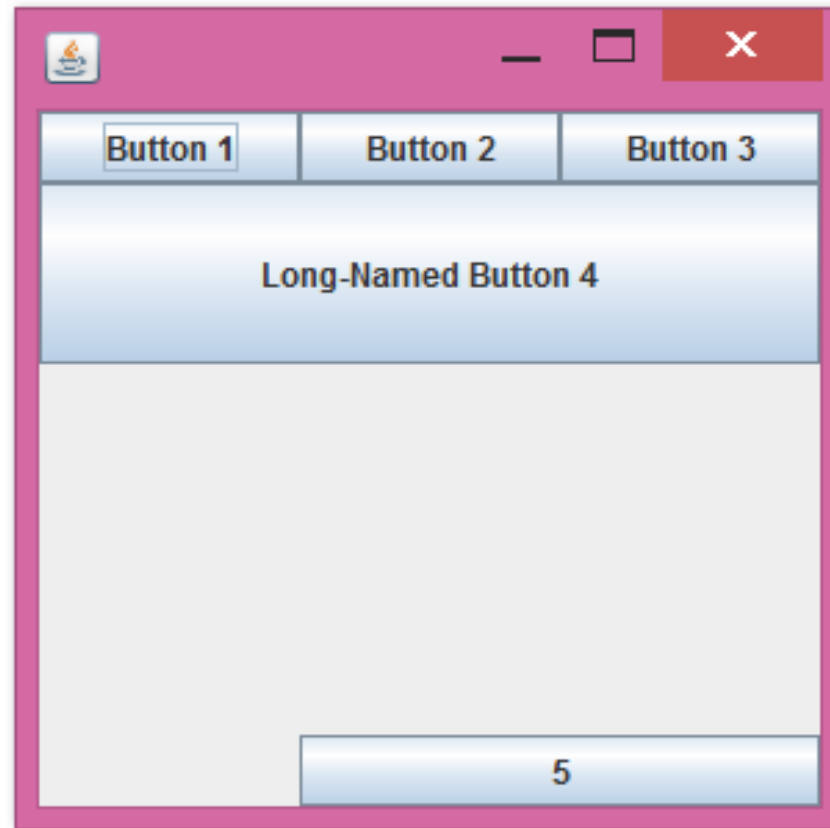
```
1      button = new JButton("Long-Named_Button_4");
2      c.ipady = 40;    //make this component tall
3      c.weightx = 0.0;
4      c.gridwidth = 3;
5      c.gridx = 0;
6      c.gridy = 1;
7      pane.add(button, c);
8
9      button = new JButton("5");
10     c.ipady = 0;    //reset to default
11     c.weighty = 1.0; //request any extra vertical space
12     c.anchor = GridBagConstraints.PAGE_END; //bottom of space
13     c.insets = new Insets(10,0,0,0); //top padding
14     c.gridx = 1;    //aligned with button 2
15     c.gridwidth = 2; //2 columns wide
16     c.gridy = 2;    //third row
17     pane.add(button, c);
18     setSize(500,500);
19     setContentPane(pane);
20     setVisible(true);
21 }
```

GridBagLayout

Lines 58–78 / 20

```
1         setVisible(true);  
2     }  
3  
4  
5     public static void main(String[] args) {  
6         GridBagLayoutDemo d = new GridBagLayoutDemo();  
7     }  
8 }
```

GridBagLayout



GridLayout

- Pour gérer des grilles de composants, sans qu'ils aient tous la même taille
- chaque composant doit être mis dans 2 groupes :
 - ✓ Un pour gérer l'alignement horizontal
 - ✓ Un pour gérer l'alignement vertical

GroupLayout

Lines 1–21 / 20

```
1 import java.awt. ;
2 import javax.swing. ;
3 import static javax.swing.GroupLayout.Alignment. ;
4 public class GPL extends JFrame {
5     JLabel label = new JLabel("Find What:");
6     JTextField textField = new JTextField();
7     JCheckBox caseCheckBox = new JCheckBox("Match Case");
8     JCheckBox wrapCheckBox = new JCheckBox("Wrap Around");
9     JCheckBox wholeCheckBox = new JCheckBox("Whole Words");
10    JCheckBox backCheckBox = new JCheckBox("Search Backwards");
11    JButton findButton = new JButton("Find");
12    JButton cancelButton = new JButton("Cancel");
13    public GPL() {
14        GroupLayout layout = new GroupLayout(getContentPane());
15        getContentPane().setLayout(layout);
16        layout.setHorizontalGroup(layout.createSequentialGroup()
17            .addComponent(label)
18            .addGroup(layout.createParallelGroup(LEADING)
19                .addComponent(textField)
20                .addGroup(layout.createSequentialGroup()
21                    .addGroup(layout.createParallelGroup(LEADING)
```

GroupLayout

Lines 20–40 / 20

```
1      .addGroup(layout.createSequentialGroup())
2      .addGroup(layout.createParallelGroup(LEADING)
3          .addComponent(caseCheckBox)
4          .addComponent(wholeCheckBox))
5      .addGroup(layout.createParallelGroup(LEADING)
6          .addComponent(wrapCheckBox)
7          .addComponent(backCheckBox)))
8      .addGroup(layout.createParallelGroup(LEADING)
9          .addComponent(findButton)
10         .addComponent(cancelButton))
11 );
12 layout.setVerticalGroup(layout.createSequentialGroup())
13     .addGroup(layout.createParallelGroup(BASELINE)
14         .addComponent(label)
15         .addComponent(textField)
16         .addComponent(findButton))
17     .addGroup(layout.createParallelGroup(LEADING)
18         .addGroup(layout.createSequentialGroup())
19             .addGroup(layout.createParallelGroup(BASELINE)
20                 .addComponent(caseCheckBox)
21                 .addComponent(wrapCheckBox))
```

GroupLayout

Lines 39–59 / 20

```
1         .addComponent(caseCheckBox)
2         .addComponent(wrapCheckBox))
3         .addGroup(layout.createParallelGroup(BASELINE)
4         .addComponent(wholeCheckBox)
5         .addComponent(backCheckBox))
6         .addComponent(cancelButton))
7     );
8     setTitle("Trouver");
9     pack();
10    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
11    setVisible(true);
12 }
13
14
15 public static void main(String args[]) {
16     GPL g =new GPL();
17 }
18 }
```

GridLayout

