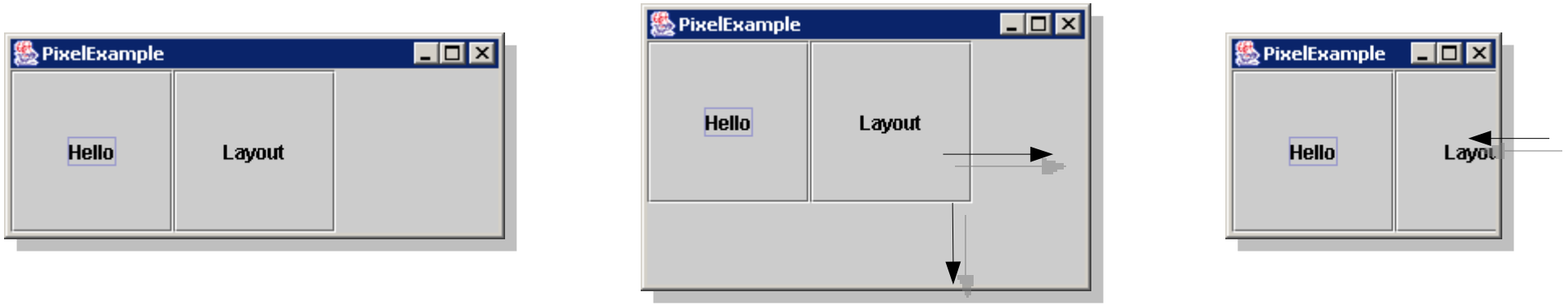
Gestionnaires de Géométrie

- Principe des gestionnaires de géométrie
- Gestionnaires simples
- Gestionnaires avec contraintes
- Ecrire son propre gestionnaire

Problème du placement des composants

- Problème : placement des composants au pixel près.



- Solution: Mécanisme de placement des objets qui lors d'un agrandissement/réduction re-place les objets au bon endroit.
- Intégrer le mécanisme au container ?
- Problème : plusieurs stratégies de placement.
- Solution: un objet gestionnaire de géométrie est associé au container (*design pattern decorator*).

Fonctionnement commun des layouts

- Les *gestionnaires de géométrie* utilisent les tailles :
  préférée : **`getPreferredSize()`**,
  maximale : **`getMaximalSize()`** et
  minimale : **`getMinimalSize()`**.
  de chaque composant pour les placer dans le conteneur.

- La méthode **`pack()`** sur une JFrame demande à chaque container sa taille *préférée* et redimensionne la fenêtre à sa taille préférée.

Fonctionnement commun des layouts (2)

- Pour suggérer ou imposer une autre taille, on redimensionne le composant avec **`setSize(largeur, hauteur)`** ou **`setPreferredSize(dimension)`**

- Ou encore, on réécrit la méthode **`getPreferredSize()`**, par exemple :

```
public Dimension() getPreferredSize() {
   return new Dimension(100,500);
}
```

- Nécessaire pour un **`JComponent`**, dont la taille est *nulle* par défaut.

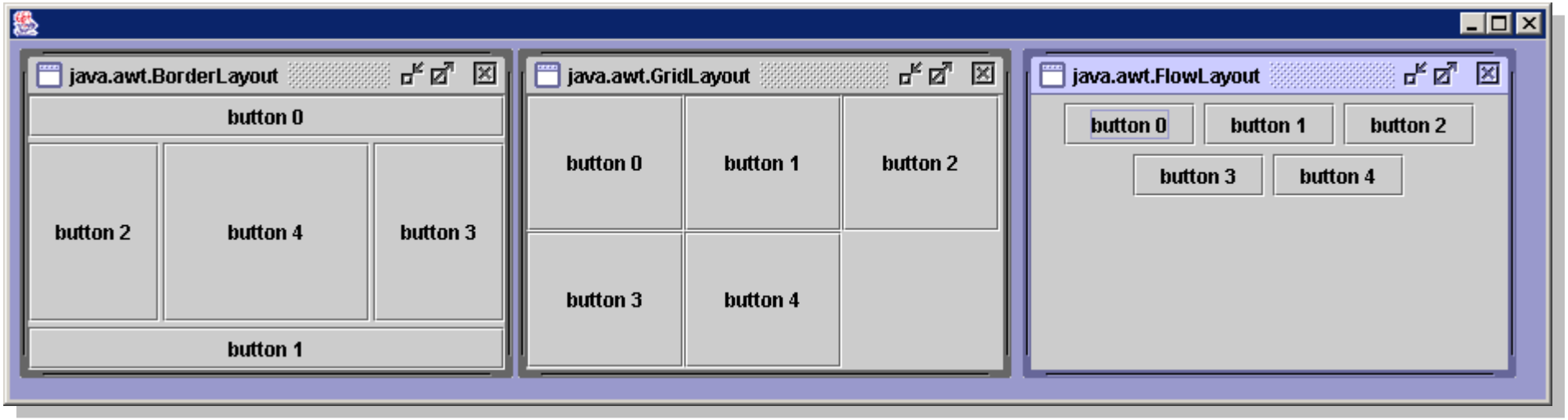
Fonctionnement commun des layouts (3)

- Tout composant a un placement par rapport à son container, donné par son origine et ses dimensions.
  - La position (un **`Point`**) est gérée par **`get/setLocation()`** et **`get/setX()`**, **`get/setY()`**
  - La taille (une **`Dimension`**) gérée par **`get/setSize()`** et **`get/setWidth()`**, **`get/setHeight()`**
  - Le rectangle est géré par **`get/setBounds()`**.

- Le gestionnaire de géométrie utilise ces méthodes pour placer le composant.

Vue d'ensemble

- Il existe deux types de gestionnaire de géométrie :
  - Les gestionnaires sans contrainte **`LayoutManager`**
  - Les gestionnaires avec contraintes **`LayoutManager2`** (hérite de **`LayoutManager`**)



Vue d'ensemble (2)

- Installer un gestionnaire de géométrie

```
LayoutManager layout=new FlowLayout();
Container c=...
c.setLayout(layout);
```

- Interaction avec un JPanel

```
LayoutManager layout=new FlowLayout();
Container c=new JPanel(layout);
```

- JPanel possède un constructeur sans paramètre qui utilise par défaut un FlowLayout

```
Container c=new JPanel();
```

Gestionnaires de géométrie simples

- Les gestionnaire de géométrie sans contrainte (**`LayoutManager`**) sont les suivants :
  - Le FlowLayout
  - Le GridLayout
  - Le BoxLayout
  - Le CardLayout

- Le CardLayout n'est plus utilisé et est remplacé par le container **`JTabbedPane`**

Le FlowLayout

- Le **`FlowLayout`** affiche les composants **de la gauche vers la droite**, et passe à la ligne s'il n'y a plus de place.
- après agrandissement, la disposition est recalculée.

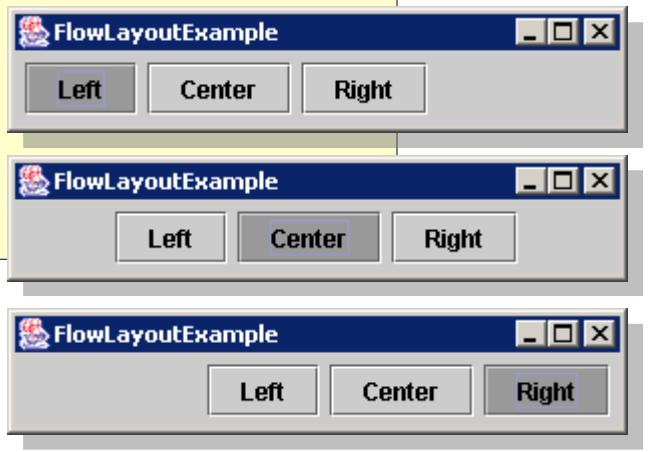
```
FlowLayout(int align,int hgap,int vgap)
```

  - **align** indique l'alignement de chaque ligne LEFT, CENTER (défaut), RIGHT et LEADING, TRAILING (selon l'orientation).
  - **hgap** (= 5) et **vgap** (=5) sont les espaces entre composants.
  - Les marges sont régies par le champ **insets** du conteneur.

Le FlowLayout (2)

- Les paramètres s'obtiennent et se modifient par des méthodes **set*** et **get***.

```
private static void addButton(final JPanel panel,String text,
                              final int align) {
    JToggleButton button=new JToggleButton(text);
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        FlowLayout layout=(FlowLayout)panel.getLayout();
        layout.setAlignment(align);
        layout.layoutContainer(panel);
        //ou panel.revalidate();
      }
    });
    panel.add(button);
  }
```

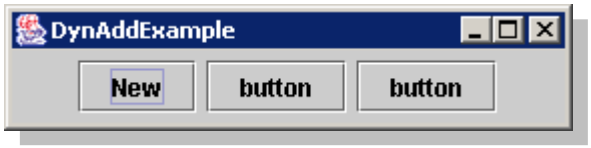


- Le changement ne prend effet
  - qu'après agrandissement/réduction,
  - ou en forçant par **layoutContainer()**.
  - ou en demandant une validation par **revalidate()**.

Modification dynamique de la hiérarchie

- De même que pour les modification des propriétés d'un layout, les modification de la hiérarchie doivent être signalées au layout.

```
final JPanel panel=new JPanel();
JButton newButton=new JButton("New");
newButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    JButton button=new JButton("button");
    panel.add(button);
    panel.revalidate();
  }
});
panel.add(newButton);
```



- De même, le changement ne prend effet
  - qu'après agrandissement/réduction,
  - ou en forçant par **`layoutContainer()`.**
  - ou en demandant une validation par **`revalidate()`.**

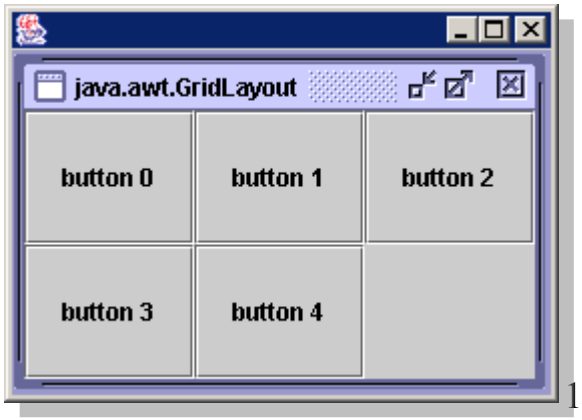
Le GridLayout

- Le **GridLayout** place les composants sur une ***grille***, ligne par ligne (dans l'ordre d'adjonction), les cellules ont la *même taille*.

- On fixe le nombre de lignes **ou** de colonnes,
  si **lignes>0**, alors **cols** est *ignoré*,
  si **lignes = 0**, alors **lignes** est *ignoré*.

```
GridLayout(int lignes, int cols, int hgap, int vgap)
```

- **hgap** et **vgap** sont nuls par défaut.



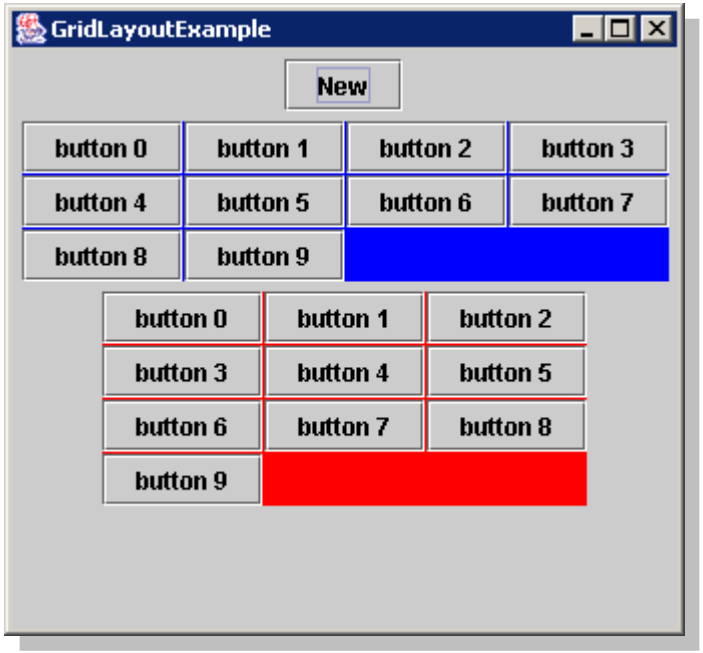
Le GridLayout (2)

- Exemple avec un gridLayout(3,0) et un gridLayout(0,3)

```
final JPanel rowPanel=new JPanel(
  new GridLayout(3,0,1,1));
rowPanel.setBackground(Color.BLUE);
final JPanel colPanel=new JPanel(
  new GridLayout(0,3,1,1));
colPanel.setBackground(Color.RED);

JPanel mainPanel=new JPanel();
JButton newButton=new JButton("New");
newButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    int value=count++;

    rowPanel.add(new JButton("button "+value));
    rowPanel.revalidate();
    colPanel.add(new JButton("button "+value));
    colPanel.revalidate();
  }
});
mainPanel.add(newButton);
mainPanel.add(rowPanel);
mainPanel.add(colPanel);
```



Le BoxLayout

- Arrange les composants en une *ligne* ou une *colonne*, en respectant la taille *préférée* de ceux-ci.

```
BoxLayout(Container c, int axe)
```

- L'axe est défini par les constantes :
  `X_AXIS` (horizontale) et **`Y_AXIS`** (verticale),
  `LINE_AXIS` et **`PAGE_AXIS`** (dépendent de l'orientation)

- Le container associé à un BoxLayout doit être créé **AVANT** le layout.

```
Container c = ...
BoxLayout layout=new BoxLayout(c, BoxLayout.X_AXIS);
c.setLayout(layout);
```

Le Box

- Le composant **`Box`** est un conteneur *transparent* dont le gestionnaire est **`BoxLayout`**.

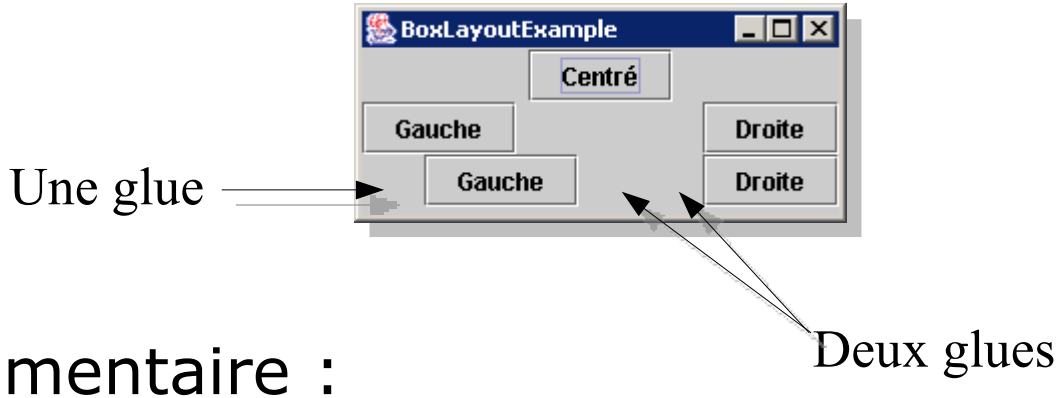
- On peut ajouter
  - de la *glue* qui fait du remplissage
  - des *struts*, blocs orientés de taille fixe
  - des *aires* (*area*) rigides

- Tout container utilisant un BoxLayout peut utiliser les composants *glue*, *struts* et *aire*.

```
Container c = ...
c.setLayout(new BoxLayout(c,BoxLayout.Y_AXIS));
c.add(Box.createHorinzontalStruts(5));
```

La glue

- La glue est en fait un ressort.
- Tout espace supplémentaire est absorbé par la glue.
- Plusieurs glues se partagent l'espace libre.



- Commentaire :
  - on centre en mettant une glue des deux côtés
  - on maintient aux bords en insérant une glue au milieu
  - Deux glues valent mieux qu'une : elle absorbent le double de place.

Exemple de BoxLayout

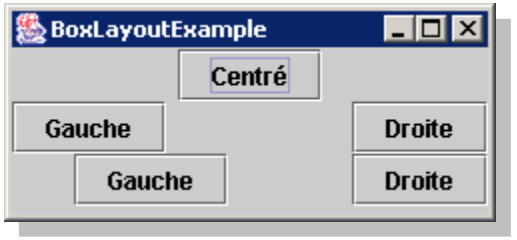
- Exemple de Glue, de Box et de BoxLayout.

```
Box line1=new Box(BoxLayout.X_AXIS);
line1.add(Box.createHorizontalGlue());
line1.add(new JButton("Centré"));
line1.add(Box.createHorizontalGlue());

Box line2=new Box(BoxLayout.X_AXIS);
line2.add(new JButton("Gauche"));
line2.add(Box.createHorizontalGlue());
line2.add(new JButton("Droite"));

Box line3=new Box(BoxLayout.X_AXIS);
line3.add(Box.createHorizontalGlue());
line3.add(new JButton("Gauche"));
line3.add(Box.createHorizontalGlue());
line3.add(Box.createHorizontalGlue());
line3.add(new JButton("Droite"));

JPanel mainPanel=new JPanel(null);
mainPanel.setLayout(
  new BoxLayout(mainPanel,BoxLayout.Y_AXIS));
mainPanel.add(line1);
mainPanel.add(line2);
mainPanel.add(line3);
```



- Evite la création d'un FlowLayout par défaut.

Gestionnaires de géométrie avec contraintes

- A chaque composant est indiqué une contrainte de placement

- Les gestionnaire de géométrie avec contraintes (**`LayoutManager2`**) sont les suivants :
  - Le BorderLayout
  - Le GridBagLayout
  - Le SpringLayout (1.4)
  - Le GroupLayout (1.6)

- Le SpringLayout est utilisé à ma connaissance uniquement par des *builders* d'interface.

Mécanisme interne de l'ajout d'un composant

- Les **contraintes** sont **stockées** sous forme d'une **table** d'association **dans** le gestionnaire de géométrie.

- Algorithme de l'ajout d'un composant

```
Container::add(Component c,Object constraints) {
  this.add(c);
  this.getLayout().addLayoutComponent(c, constraints);
}
```

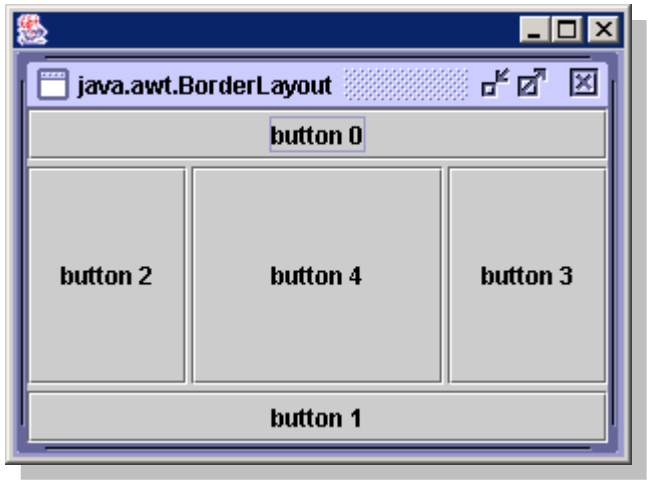
- En SWT, le contraintes (LayoutData) sont stockées dans le composant.

Le BorderLayout

- Divise sa zone en 5 régions : nord, sud, ouest, est et centre.

```
Container c=...
c.add(new JButton(),BorderLayout.NORTH);
```

- "nord" et "sud" occupent *toute la largeur*,
- "ouest" et "est" occupent *la hauteur qui reste*,
- "centre" occupe la *place restante*.



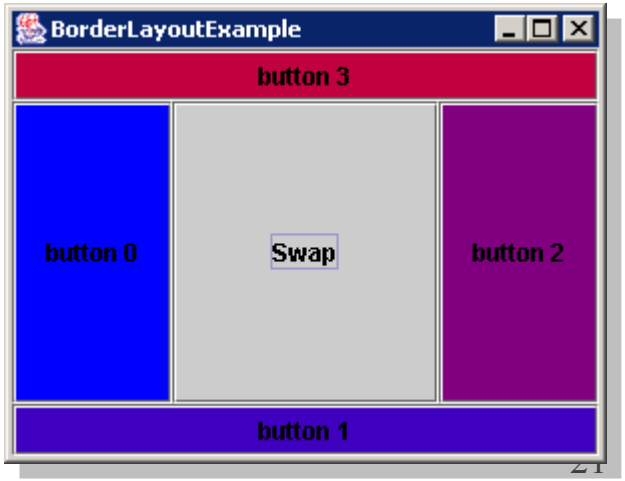
- Le placement est donc indépendant de l'ordre d'ajout dans le container

Un exemple de gestion des contraintes

```
static void swap(JPanel panel,LayoutManager2 layout,
                 Object[] constraints) {
  layout.invalidateLayout(panel);
  Collections.shuffle(Arrays.asList(constraints));
  for(int i=0;i<constraints.length;i++)
    layout.addLayoutComponent(panel.getComponent(i),
      constraints[i]);
  layout.layoutContainer(panel);
}
public static void main(String[] args) {
  JFrame frame=new JFrame("BorderLayoutExample");
  final String[] constraints=new String[]{
    BorderLayout.NORTH, BorderLayout.SOUTH,
    BorderLayout.EAST, BorderLayout.WEST
  };
  final BorderLayout layout=new BorderLayout();
  final JPanel panel=new JPanel(layout);
  for(int i=0;i<constraints.length;i++) {
    JButton button=new JButton("button "+i);
    button.setBackground(new Color(i*64,0,255-i*64));
    panel.add(button);
  }
  swap(panel,layout,constraints);

  JButton swapButton=new JButton("Swap");
  swapButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      swap(panel,layout,constraints);
    }
  });
  panel.add(swapButton);
```

- invalidateLayout() retire l'ensemble des contraintes.
- addLayoutComponent() fixe une contrainte pour un composant.
- layoutContainer() demande un recalcul puis un dessin de la zone.

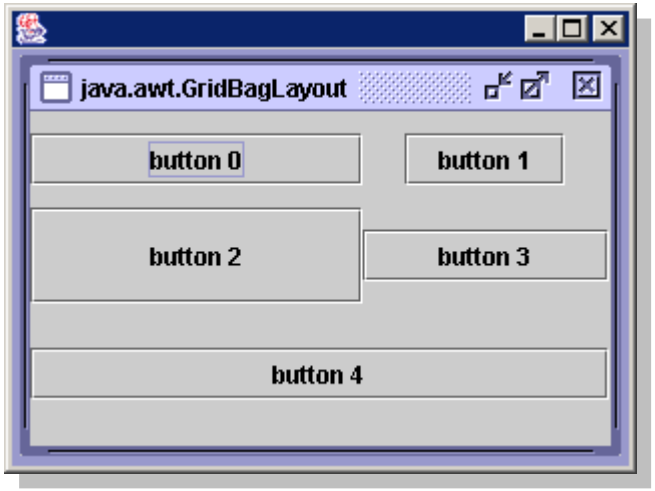


Le GridBagLayout

- Le **`GridBagLayout`** répartit les composants *dans une grille*, selon un ensemble de contraintes.

- Les contraintes sont exprimés par la classe **`GridBagConstraints.`**

- Les "contraintes" d'un composant concernent
  - la *zone* réservée dans la grille
  - la *place* occupée dans cette zone



Le GridBagConstraints

- Les champs de la classe GridBagConstraints définissent trois groupes de contraintes :
  - *Positionnement de la zone*
    gridx, gridy, gridwidth, gridheight
  - *Positionnement du composant dans la zone*
    ipadx, ipady, insets, anchor
  - *Redimensionnement de la zone et du composant*
    fill, weightx, weighty

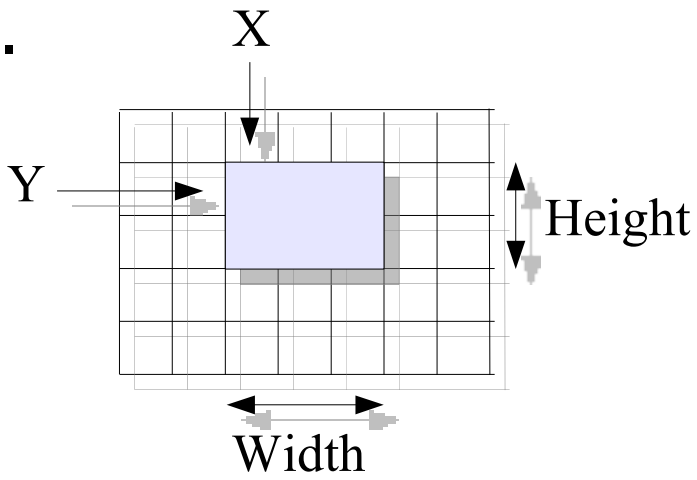
```
JPanel panel=new JPanel(new GridBagLayout());
GridBagConstraints gbc=new GridBagConstraints();
gbc.gridwidth=1;
gbc.anchor=GridBagConstraints.CENTER;
gbc.fill=GridBagConstraints.BOTH;

panel.add(new JButton(),gbc);
```

Effectue une copie des contraintes

Définition de la zone d'un composant

- Chaque fille réserve une ***zone*** dans la grille par son origine et sa dimension.
  Exemple : `gridx = 2, gridy = 2,`
  `gridwidth = 2, gridheight = 3`
- Par défaut : **`x`**=**`y`**=RELATIVE et **`width=height=1`**.
- Si **`gridx = RELATIVE`**,
  l'origine est à droite du précédent.
- Si **`gridwith = RELATIVE`**,
  avant dernière zone.
- Si **`gridwith = REMAINDER`**,
  dernière zone de la ligne.



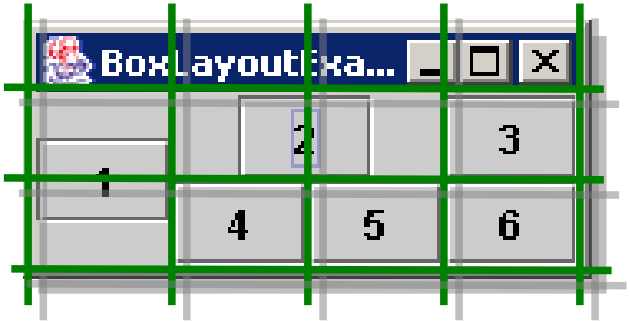
Exemple de zones

- On utilise ici le même objet **GridBagConstraints**.

```
JPanel panel=new JPanel(new GridBagLayout());

Object[] constraints=new Object[] {
  "1",new int[]{1,2},
  "2",new int[]{2,1},
  "3",new int[]{GridBagConstraints.REMAINDER,1},
  "4",new int[]{1,1},
  "5",new int[]{GridBagConstraints.RELATIVE,1},
  "6",new int[]{GridBagConstraints.REMAINDER,1}
};

GridBagConstraints gbc=new GridBagConstraints();
for(int i=0;i<constraints.length;i+=2) {
  JButton button=new JButton(
    constraints[i].toString());
  int[] grids=(int[])constraints[i+1];
  gbc.gridwidth=grids[0];
  gbc.gridheight=grids[1];
  panel.add(button,gbc);
}
```



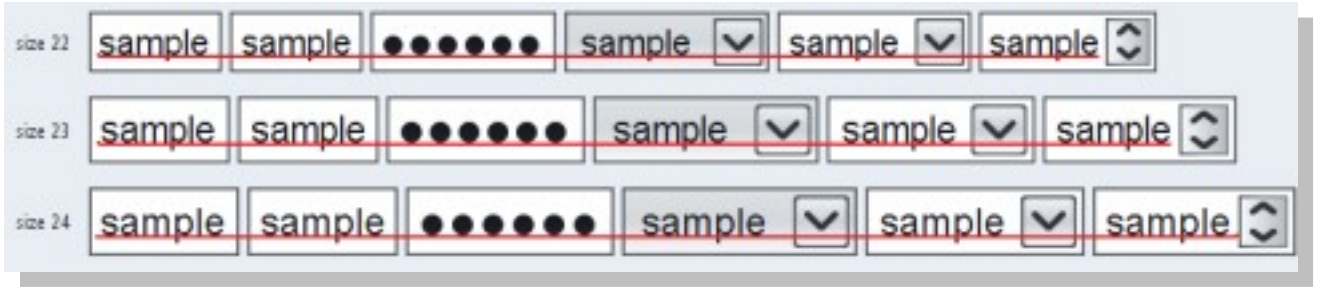
Positionnement du composant dans la zone

- Une fois la zone définie, si celle-ci est plus grande que le composant, il est possible de spécifier :
  - Le décalage par rapport au coin supérieur gauche (**`ipadx`**/**`ipady`**)
  - La marge du composant (**`insets`**)
  - Le placement du composant ou ancre (**`anchor`**)

- Les ancres possibles sont :
  NORTH, SOUTH, WEST, EAST, NORTHWEST, NORTHEAST, SOUTHWEST, SOUTHEAST et CENTER (défaut)
  PAGE_START, PAGE_END, LINE_START, LINE_END, FIRST_LINE_START, FIRST_LINE_END, LAST_LINE_START, LAST_LINE_END (dépend de l'orientation).
  Ainsi que BASELINE, BASELINE_LEADING, BASELINE_TRAILING, ABOVE_BASELINE, ABOVE_BASELINE_LEADING, ABOVE_BASELINE_TRAILING, BELOW_BASELINE, BELOW_BASELINE_LEADING, BELOW_BASELINE_TRAILING,

Placement par rapport à la baseline

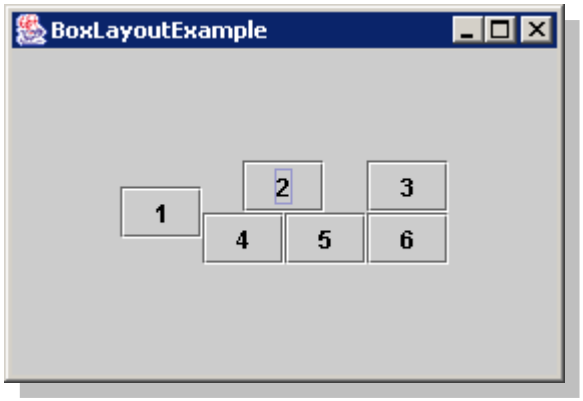
- Par défaut, chaque composant qui sait afficher du texte est capable de fournir sa baseline
  La baseline est la hauteur par rapport au début du texte



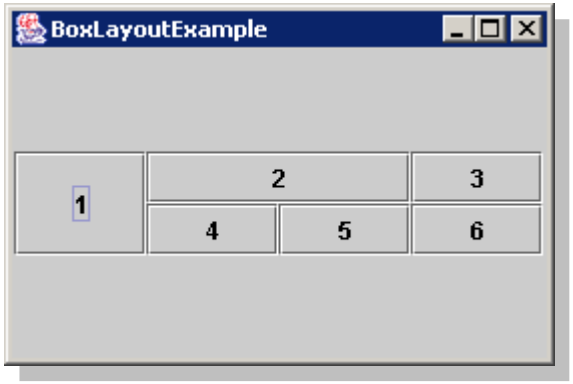
- L'ancre BASELINE, indique que les composants seront placés sur la même ligne de texte
- BASELINE_ABOVE (resp. BASELINE_BELOW) indique que le bas (resp. Haut) du composant sera placé au niveau de la baseline

Redimensionnement de la zone et du composant

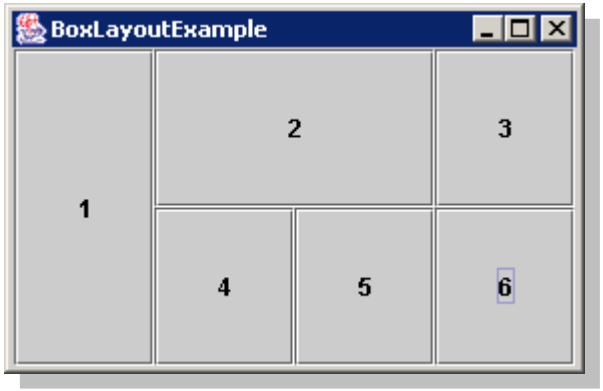
- Il reste trois champs qui permettent de jouer sur l'écartement entre chaque ligne de la grille et la façon dont le composant suit cet écartement :
  - Le redimensionnement du composant (**`fill`**) NONE(défaut), VERTICAL, HORINZONTAL, BOTH
  - Le poid en x/y (**`weightx`**/**`weighty`**) facteur d'agrandissement.



fill=NONE
wx=0.0, wy=0.0



fill=BOTH
wx=1.0, wy=0.0



fill=BOTH
wx=1.0, wy=1.0

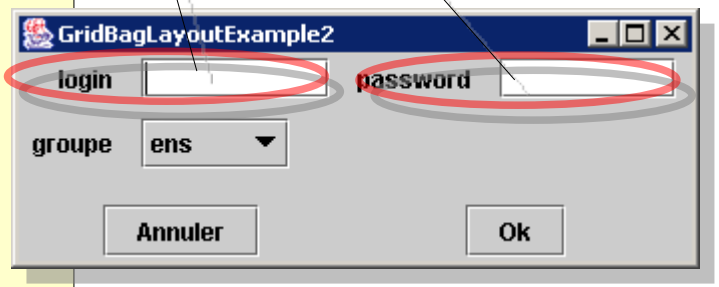
Un exemple complet

- Une fenêtre de création de login.
- L'application est séparé en plusieurs parties.

```
private static void addForm(JPanel panel,
  String text,JComponent component,int gridwidth) {

  JLabel label=new JLabel(text);
  GridBagConstraints gbc=new GridBagConstraints();
  gbc.gridwidth=1;
  gbc.insets=new Insets(2,7,2,7);
  gbc.anchor=GridBagConstraints.BASELINE_TRAILING;
  panel.add(label,gbc);
  gbc.gridwidth=gridwidth;
  gbc.fill=GridBagConstraints.HORIZONTAL;
  gbc.weightx=1.0;
  panel.add(component,gbc);
}
```

```
Public static void main(String[] args) {
  JPanel panel=new JPanel(new GridBagLayout());
  addForm(panel,"login",new JTextField(10),1);
  addForm(panel,"password",new JTextField(10),
    GridBagConstraints.REMAINDER);
  String[] groups={"ens","étudiant"};
  addCombo(panel,"groupe",new JComboBox(groups));
  addButton(panel,"Annuler");
  addButton(panel,"Ok");
```

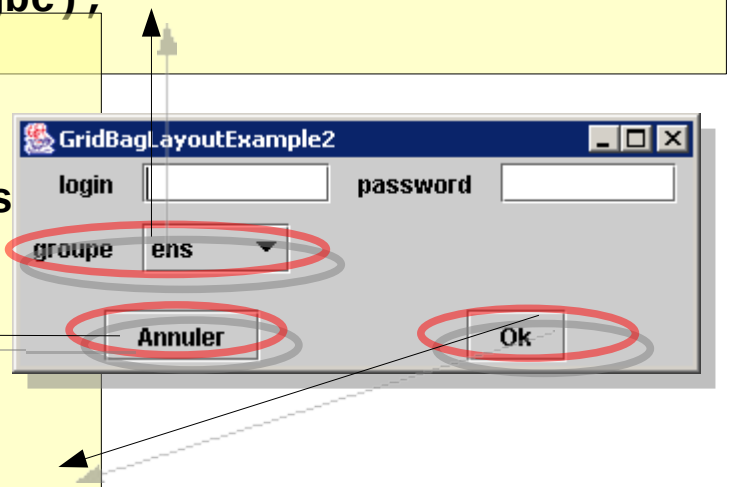


Un exemple complet (2)

- Chaque méthode correspond à un assemblage de composants particuliers.

```
private static void addCombo(JPanel panel,
  String text,JComponent component) {
    JLabel label=new JLabel(text);
    GridBagConstraints gbc=new GridBagConstraints();
    gbc.gridwidth=1;
    gbc.insets=new Insets(2,7,2,7);
    gbc.anchor=GridBagConstraints.BASELINE;
    gbc.weighty=1.0;
    panel.add(label,gbc);
    gbc.gridwidth=GridBagConstraints.REMAINDER;
    gbc.fill=GridBagConstraints.NONE;
    gbc.weightx=1.0;
    panel.add(component,gbc);
```

```
private static JButton addButton(JPanel panel,
  String text) {
    Jbutton button=new JButton(text);
    GridBagConstraints gbc=new GridBagConstraints
    gbc.gridwidth=2;
    gbc.insets=new Insets(10,0,2,0);
    gbc.anchor=GridBagConstraints.CENTER;
    gbc.fill=GridBagConstraints.NONE;
    panel.add(button,gbc);
    return button;
  }
```

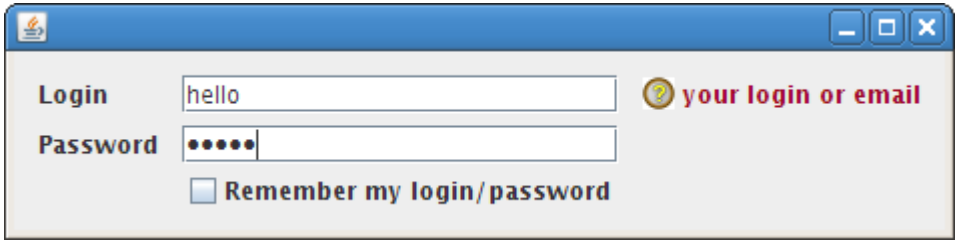


Le GroupLayout

- Le **`GroupLayout`** stock ses composants *dans deux groupes*, un pour le positionnement horizontale et un autre pour le positionnement vertical.

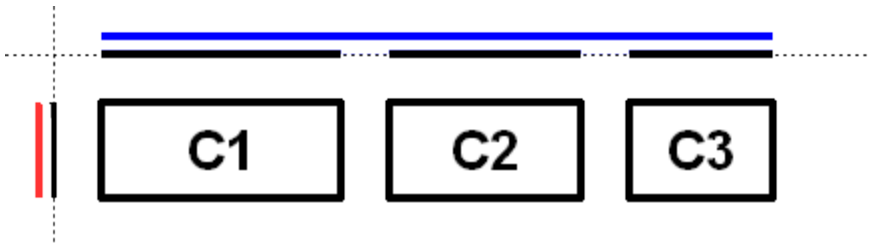
- Chaque composant du container est inclus dans les deux groupes permettant de calculer sont positionnement vertical et horizontal

- Les groupes sont des inner-class de GroupLayout, ils sont donc créé à partir de l'objet layout



GroupLayout.Group

- Un groupe peut contenir un composant, un groupe ou un espace (gap).
- Il existe deux sortes de groupes:
  - SequentialGroup
    - La taille est la somme des tailles
    - GroupLayout.createSequentialGroup(alignement)



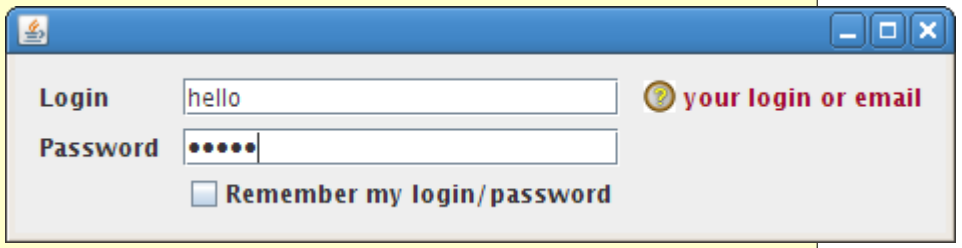
  - ParallelGroup
    - La taille est le max de la taille
    - GroupLayout.createParallelGroup(alignement)

Exemple

- En 2 étapes, on crée tous les composants puis on les mets dans les deux groupes

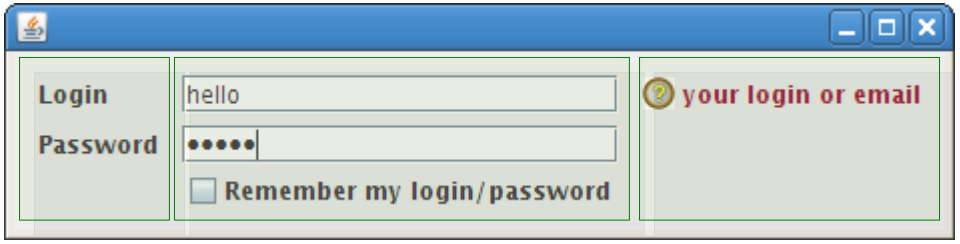
```
JPanel panel = new JPanel(null);
GroupLayout layout = new GroupLayout(panel);
layout.setAutoCreateGaps(true);
layout.setAutoCreateContainerGaps(true);
panel.setLayout(layout);

JLabel label1=new JLabel("Login");
JLabel label2=new JLabel("Password");
JTextField tf1=new JTextField(12);
JTextField tf2=new JPasswordField(12);
JLabel loginHelp=new JLabel("your login or email");
loginHelp.setIcon(new ImageIcon(
  GroupLayoutExample.class.getResource("help.png")));
loginHelp.setForeground(new Color(155,13,35));
JCheckBox checkbox=new JCheckBox("Remember my login/password");
```



Exemple

- Le groupe séquentiel horizontal (calcul la position horizontal) contient 3 groupes parallèles.



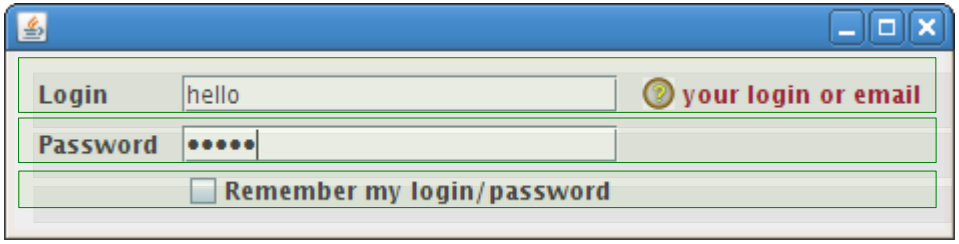

```
GroupLayout.SequentialGroup hGroup = layout.createSequentialGroup();

hGroup.addGroup(layout.createParallelGroup().
  addComponent(label1).addComponent(label2));
hGroup.addGroup(layout.createParallelGroup().
  addComponent(tf1).addComponent(tf2).
  addComponent(checkbox,Alignment.LEADING));
hGroup.addGroup(layout.createParallelGroup().
  addComponent(loginHelp));

layout.setHorizontalGroup(hGroup);
```

Exemple

- Le groupe séquentiel verticale (calcul la position vertical) contient aussi 3 groupes parallèles




```
GroupLayout.SequentialGroup vGroup = layout.createSequentialGroup();

vGroup.addGroup(layout.createParallelGroup(Alignment.BASELINE).
  addComponent(label1).addComponent(tf1).addComponent(loginHelp));
vGroup.addGroup(layout.createParallelGroup(Alignment.BASELINE).
  addComponent(label2).addComponent(tf2));
vGroup.addGroup(layout.createParallelGroup(Alignment.BASELINE).
  addComponent(checkbox));

layout.setVerticalGroup(vGroup);
```

Créer son propre layout manager

- Implanter un layout manager revient à implanter l'interface **LayoutManager**.

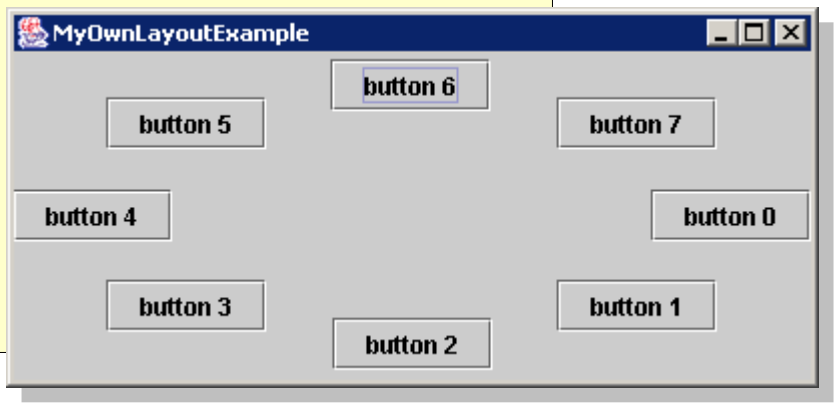
```
public interface LayoutManager {
  public Dimension preferredLayoutSize(Container parent);
  public Dimension minimumLayoutSize(Container parent);
  void addLayoutComponent(String name, Component c);
  void removeLayoutComponent(Component c);
  void layoutContainer(Container parent);
}
```

- **minimum/preferredLayoutSize()** renvoie la taille du container (donc en fonctions des fils)
- Les méthodes **add/removeLayoutComponent()** sont appelées lors de l'insertion/suppression d'un fils.
- **layoutContainer()** demande le placement des fils dans le container (avec **setBounds()** par ex.)

Un exemple

- Layout qui met les composants sur un cercle

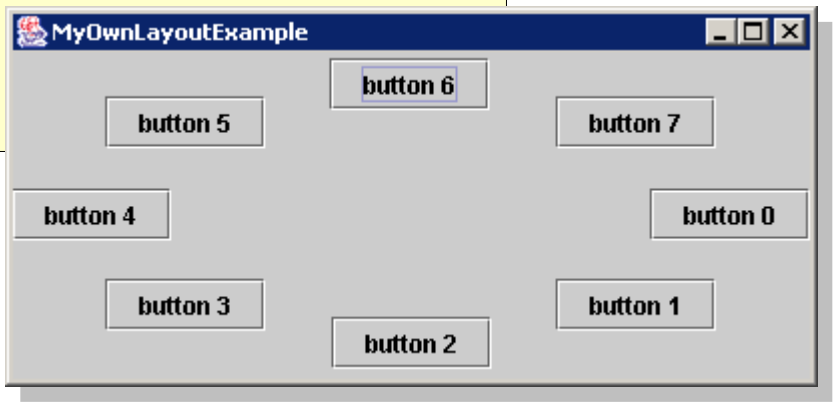
```
public class MyOwnLayoutExample implements LayoutManager {
  public void addLayoutComponent(String name, Component c) {
  }
  public void removeLayoutComponent(Component c) {
  }
  public Dimension minimumLayoutSize(Container parent) {
    return preferredLayoutSize(parent);
  }
  public Dimension preferredLayoutSize(Container parent) {
    int width=0,height=0;
    int count=parent.getComponentCount();
    for(int i=0;i<count;i++) {
      Component c=parent.getComponent(i);
      Dimension preferred=c.getPreferredSize();
      width+=preferred.getWidth();
      height+=preferred.getHeight();
    }
    return new Dimension(width,height);
  }
  ...
```



Un exemple (suite)

- Place les composants dans le container

```
  public void layoutContainer(Container parent) {
    int width=parent.getWidth()/2;
    int height=parent.getHeight()/2;
    int count=parent.getComponentCount();
    for(int i=0;i<count;i++) {
      double angle=2*Math.PI*i/count;
      int x=(int)(width+0.8*width*Math.cos(angle));
      int y=(int)(height+0.8*height*Math.sin(angle));
      Component c=parent.getComponent(i);
      Dimension preferred=c.getPreferredSize();
      c.setBounds(x-preferred.width/2,y-preferred.height/2,
        preferred.width,preferred.height);
    }
  }
}
```



Gestionnaire par défaut

- Le gestionnaire de géométrie par défaut du contentPane d'une JFrame est un BorderLayout

```
JFrame frame=new JFrame();
Container container=frame.getContentPane();
LayoutManager layout=container.getLayout();
// layout instanceof BorderLayout -> true
```

- Le gestionnaire par défaut du JPanel est un FlowLayout

```
Container c=new JPanel();
// c.getLayout() instanceof FlowLayout -> true
```

- Le gestionnaire par défaut du Box est un BoxLayout

```
Container c=new Box();
// c.getLayout() instanceof BoxLayout -> true
```