



---

**www.Mcours.com**  
Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

# Interface Graphique en Java 1.6

Containers, menus, barre d'outils  
et actions

Sébastien Paumier



# Les containers

---

- rappel: tous les composants Swing sont des containers potentiels, même s'il vaut mieux ne pas essayer pour la plupart d'entre eux
- règle d'or: on ne peut pas ajouter un même composant plusieurs fois:
  - ni dans le même container
  - ni dans des containers différents
- **JPanel** est le container le plus simple, mais il y en a d'autres...



# Les containers spécialisés

---

- containers qui imposent un layout manager, et parfois des fils
  - exemple: **JFrame** impose un fils unique **JRootPane**
- faciles à utiliser
- pas besoin d'en définir de nouveaux, à moins d'un besoin **très** particulier
- **attention:** on n'ajoute pas forcément les composants avec le **add** classique...



# La Box

---

- **Box** (pas de **J**) est presque équivalente à un **JPanel** muni d'un **BoxLayout**

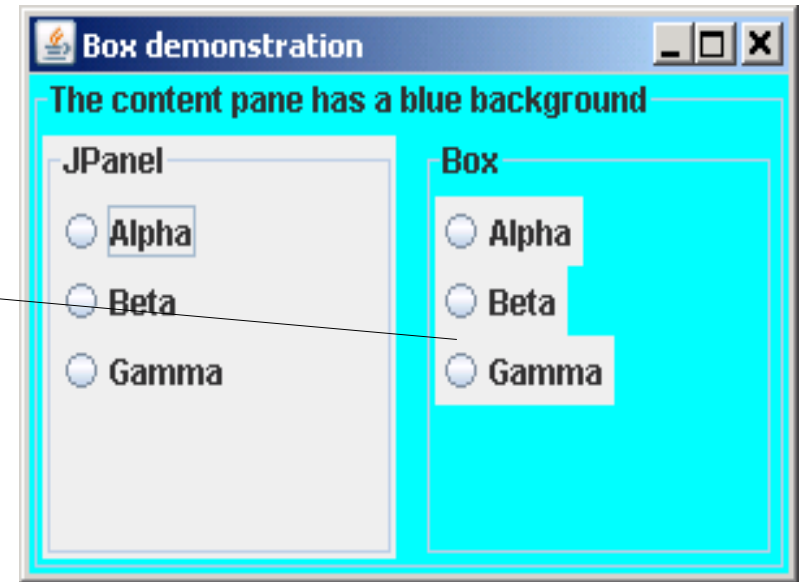
```
Box box=new Box (BoxLayout.Y_AXIS);
```

- remarque: l'orientation passée au constructeur est un champ de **BoxLayout** et non de **Box**
- **attention:** container transparent, ne pas l'utiliser comme content pane



# La Box

à n'utiliser que quand on  
veut vraiment de la  
transparence



```
Box b=new Box(BoxLayout.Y_AXIS);  
b.add(new JRadioButton("Alpha"));  
b.add(new JRadioButton("Beta"));  
b.add(new JRadioButton("Gamma"));  
b.add(Box.createVerticalGlue());  
/* This has no effect on a Box! */  
b.setOpaque(true);  
b.setBorder(BorderFactory.createTitledBorder("Box"));
```



# Le JSplitPane

---

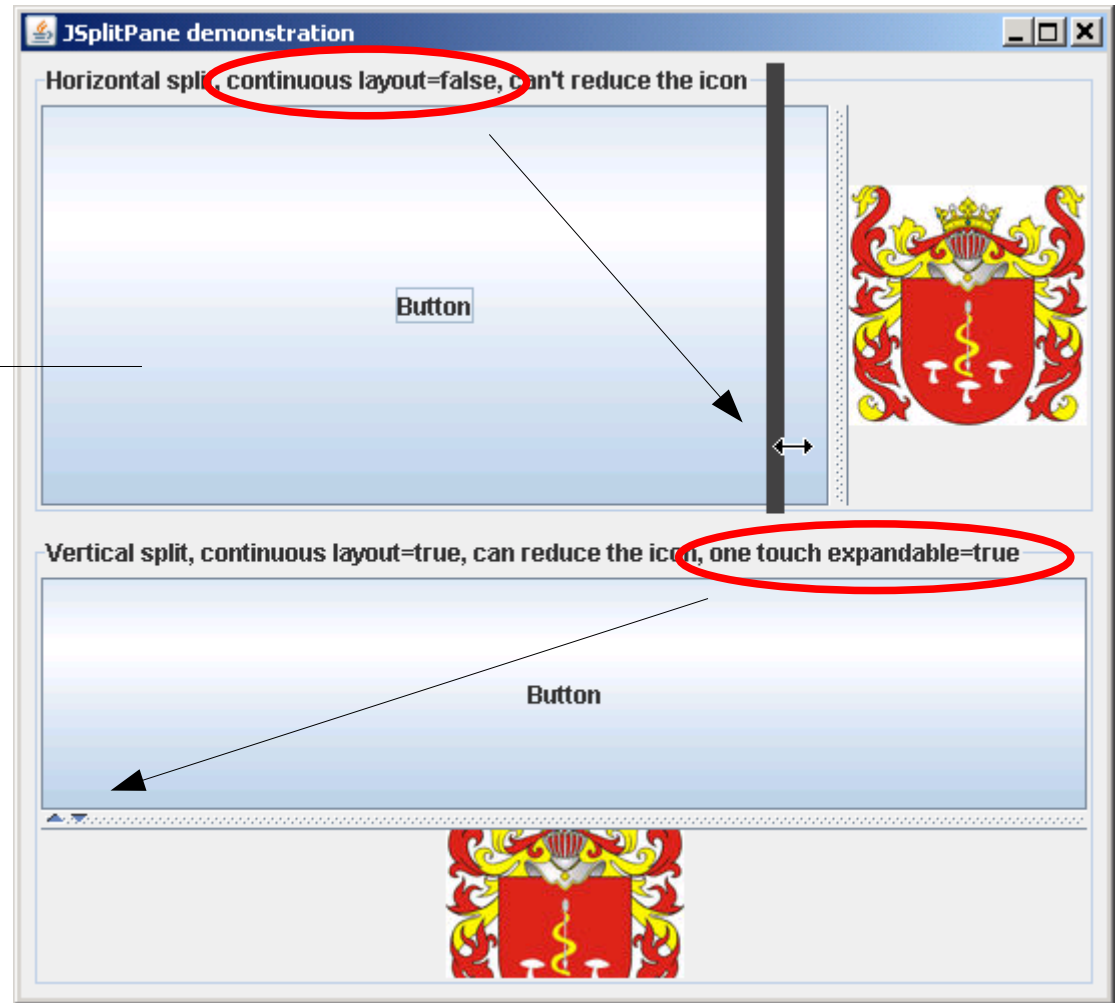
- container à 2 zones séparées par une barre de redimensionnement
- ne peut pas réduire un composant à moins de sa taille minimum
- ne peut recevoir que 2 composants qui sont passés au constructeur:

```
public JSplitPane(int newOrientation,  
                  boolean newContinuousLayout,  
                  Component newLeftComponent,  
                  Component newRightComponent)
```



# Le JSplitPane

les composants  
sont maximisés à  
la taille disponible  
dans leur zone





# Le JTabbedPane

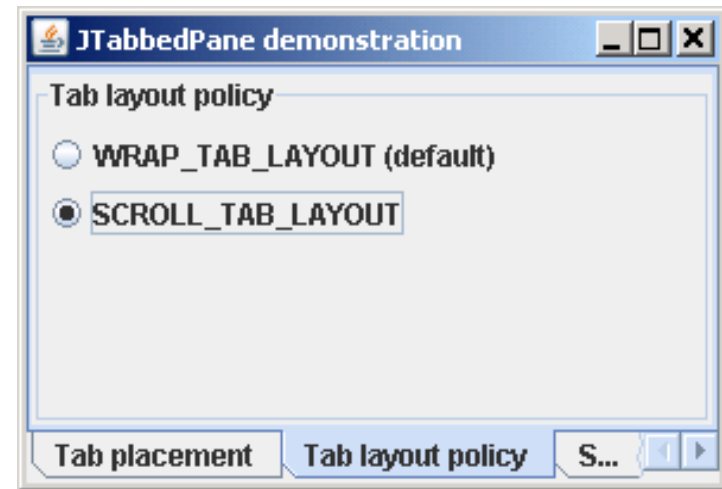
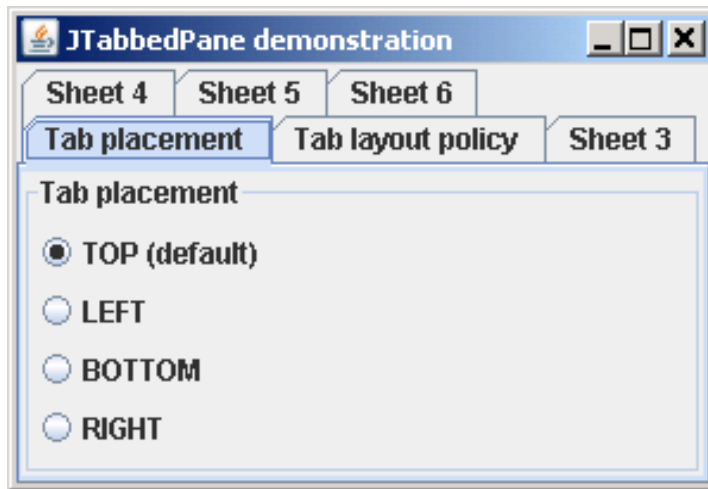
---

- container à onglets
- ajout des composants avec:
  - `addTab(String title, Component c)`
- possible de définir le placement des onglets avec:
  - `set/getTabPlacement`
- que faire quand ça déborde ?
  - `set/getTabLayoutPolicy`





# Le JTabbedPane

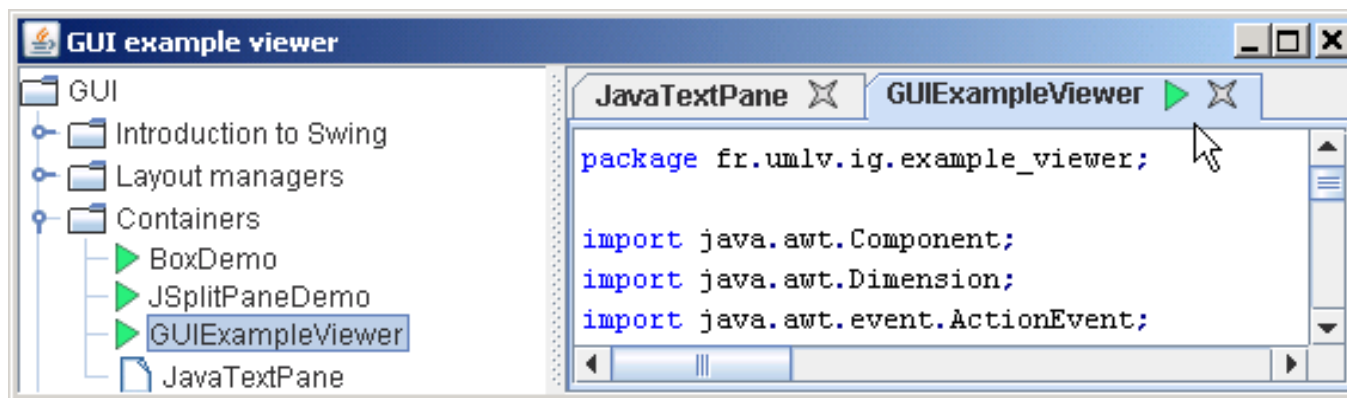


- chaque composant est maximisé
- accès au composant sélectionné:
  - `set/getSelectedComponent (Component c)`
  - ou `set/getSelectedIndex(int index)`



# Le JTabbedPane

- si on veut gérer soi-même l'onglet, il faut utiliser:
  - `setTabComponentAt(int index, Component c)`
- exemple: container avec label+2 boutons personnalisés





# Le JScrollPane

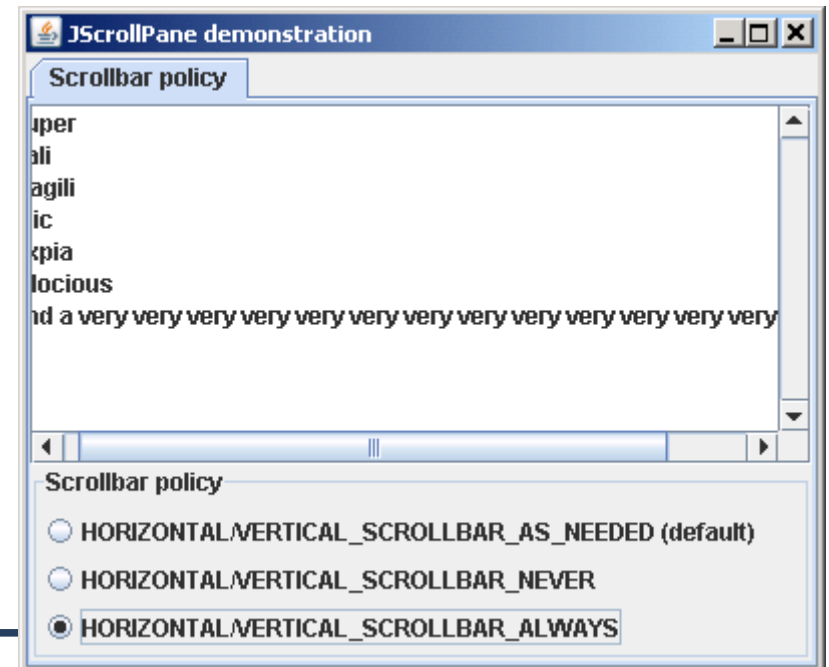
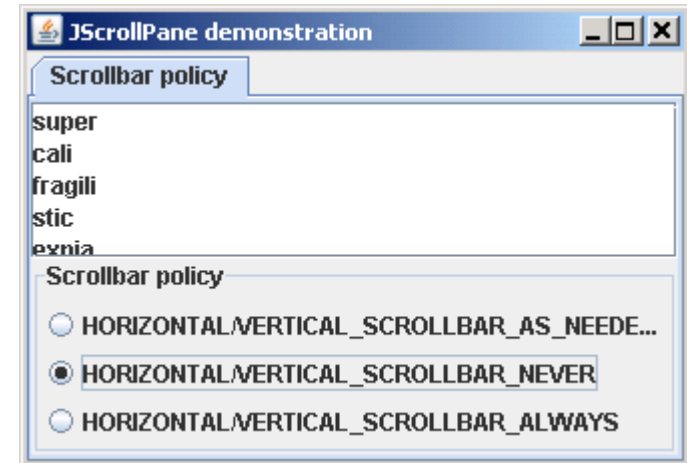
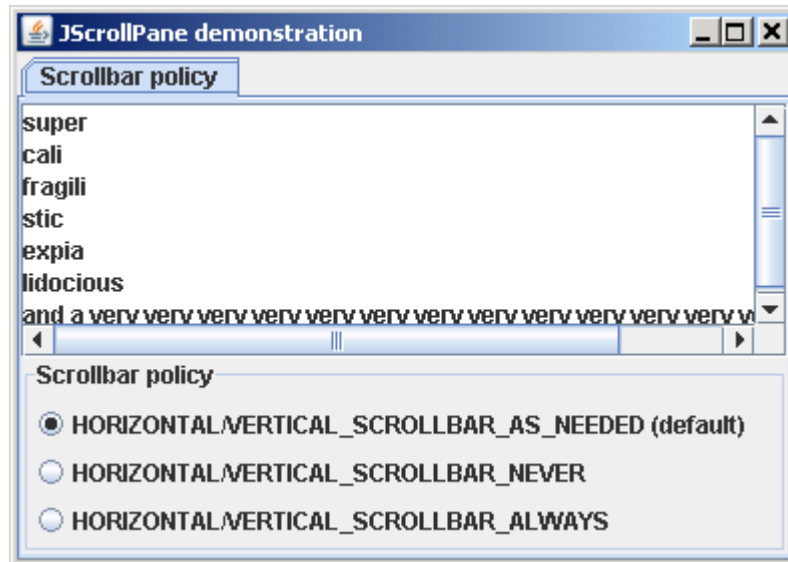
---

- permet d'afficher un composant à sa taille préférée, sans le réduire s'il n'y a pas assez d'espace
- le composant est passé au constructeur:
  - `JScrollPane(Component view, int vsbPolicy, int hsbPolicy)`
  - les 2 autres paramètres régissent la présence des barres de défilement



# Le JScrollPane

- gestion des barres de défilement:





# Le JScrollPane

---

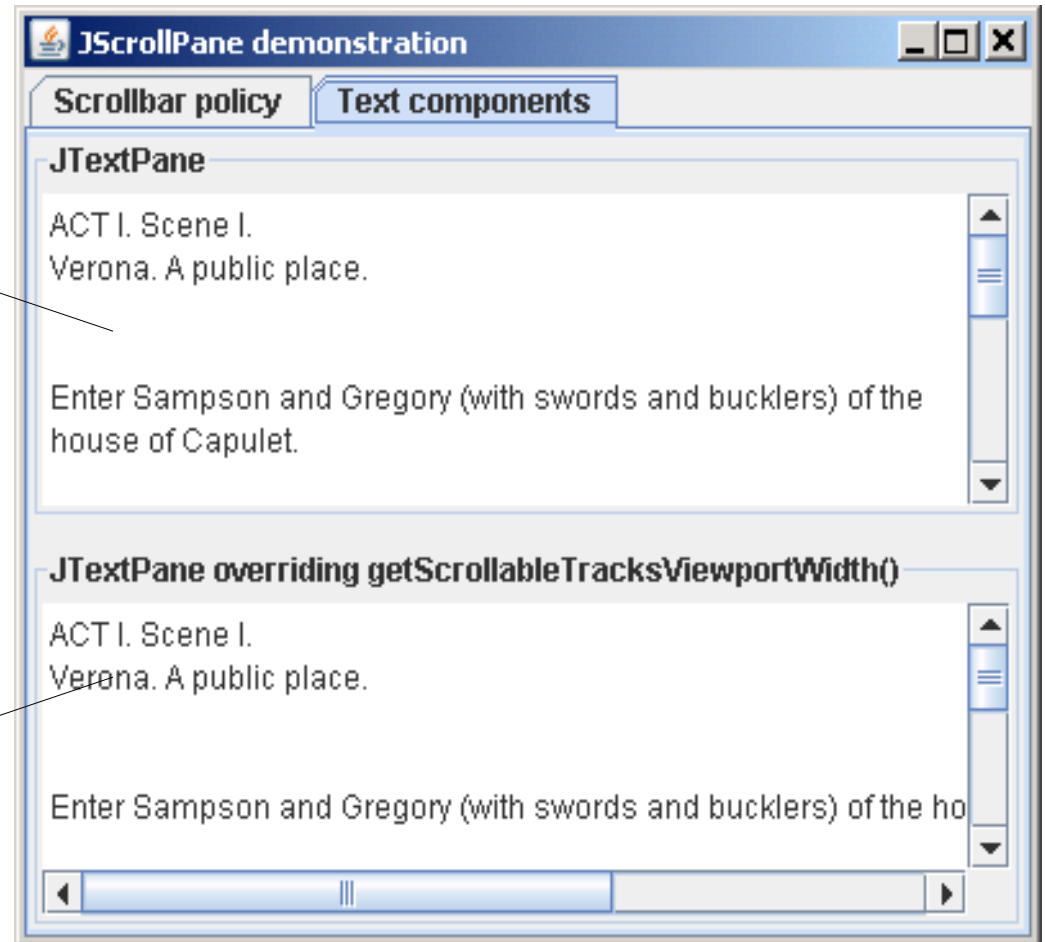
- pour que ça fonctionne, le composant inséré doit implémenter **Scrollable**:
  - composants textes, listes, tables, arbres
- pour un composant texte qui gère le multiligne, il faut ruser:
  - soit le mettre au centre d'un panel avec un **BorderLayout**
  - soit redéfinir **getScrollableTracksViewportWidth()** pour toujours retourner **false**



# Le JScrollPane

passage à la ligne  
géré par le **JTextPane**

pas de passage à la  
ligne, c'est le  
**JScrollPane** qui  
travaille

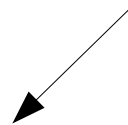




# Le JScrollPane

---

- possibilité d'ajouter des headers et un coin avec:
  - `setRowHeaderView(Component c)`
  - `setColumnHeaderView(Component c)`
  - `setCorner(String key, Component c)`



position du coin (s'il reste de la place):

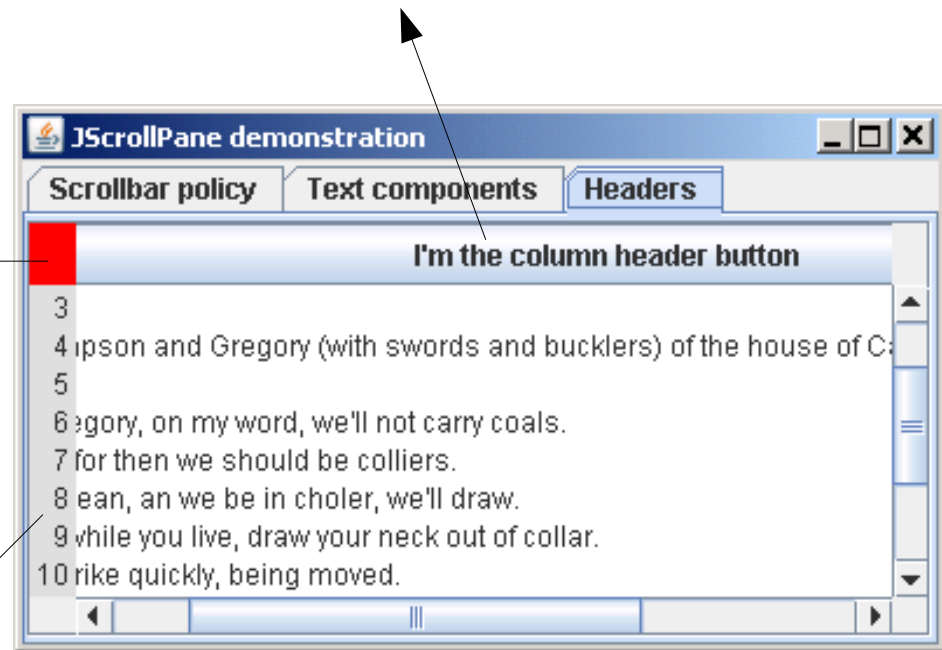
- `ScrollPaneConstants.UPPER_LEFT_CORNER`
- `ScrollPaneConstants.UPPER_RIGHT_CORNER`
- etc.



# Le JScrollPane

bouton dont la largeur a été calée manuellement sur la largeur préférée du **JTextPane**

**JPanel** qui prend les hauteur et largeur des headers



**JTextPane** servant à indiquer les numéros de lignes





# Le JLayeredPane

---

- affiche les composants par couches
- chaque couche est codée par un **Integer**:

~~add(binou, 14)~~ ⇒ `add(binou, Integer.valueOf(14))`

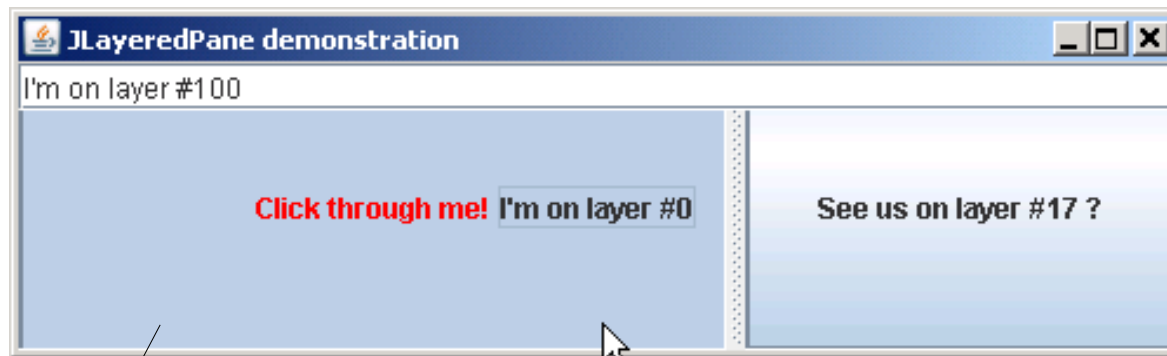
- par défaut, il n'y a **pas** de layout manager pour déterminer la taille des composants contenus dans le **JLayeredPane** :(
  - il faut donc en écrire un, mais on sait faire :)  
(cf. `fr.uml.v.ig.lesson1.LayeredLayoutManager`)



# Le JLayeredPane

---

- exemple à 3 couches:



- pour voir à travers la couche 17, il faut rendre le **JSplitPane** transparent
- pour pouvoir cliquer à travers le **JLabel**, il faut que le **JSplitPane** dise qu'il ne contient pas les clics sur le **JLabel**



# La méthode contains

---

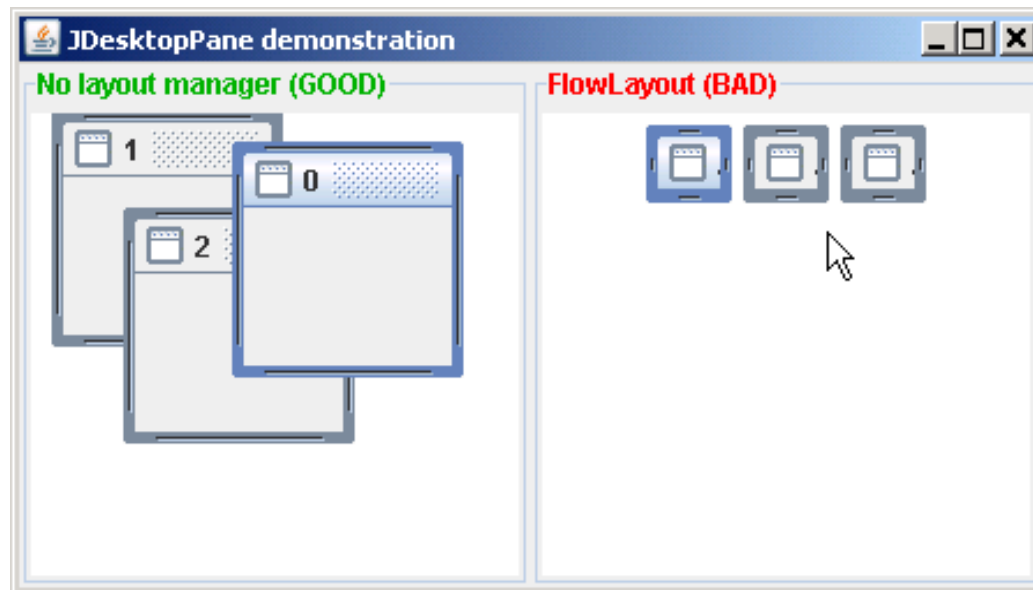
- **contains** est utilisée par Swing pour savoir (entre autres choses) qui est concerné par un clic
- donc, facile de faire mentir le **JSplitPane**:

```
JSplitPane split=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,label,b) {  
    /* These overridings are used to allow clicks on the background  
    * through the JLabel */  
    @Override public boolean contains(int x, int y) {  
        if (label.contains(x,y)) {return false;}  
        return super.contains(x,y);  
    }  
  
    @Override public boolean contains(Point p) {  
        if (label.contains(p)) {return false;}  
        return super.contains(p);  
    }  
};
```



# Le JDesktopPane

- container spécial destiné à recevoir des fenêtrés internes (**JInternalFrame**)
- ne **JAMAIS** lui donner de layout manager:





# La JInternalFrame

---

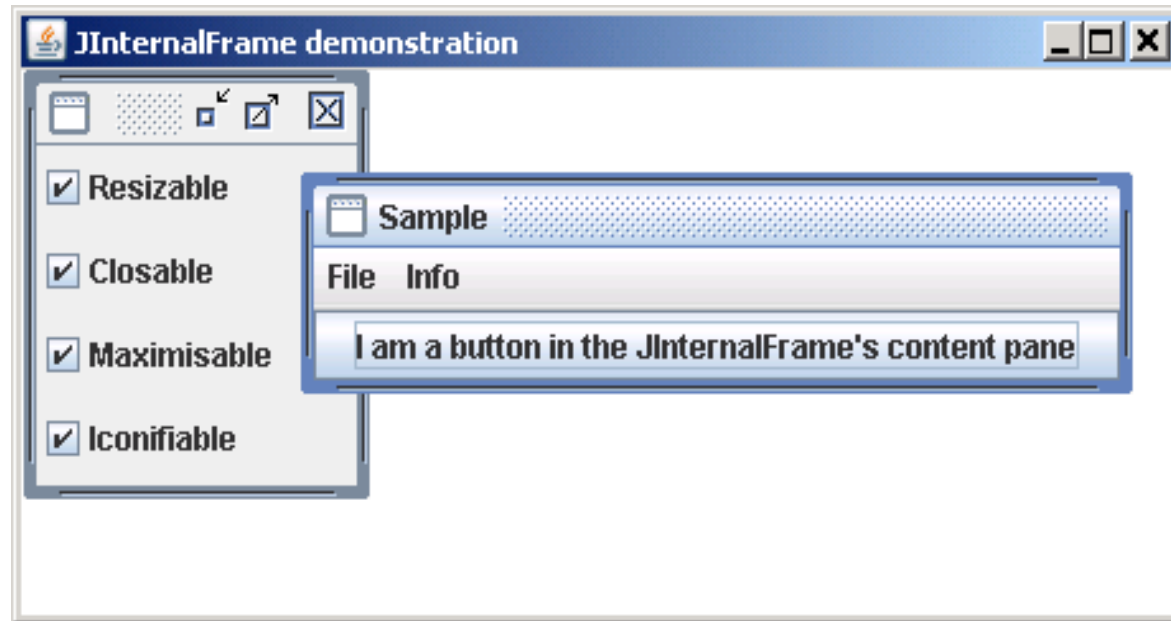
- fenêtre interne à ne mettre **que** dans un JDesktopPane:

`JInternalFrame(String title, boolean resizable, boolean closable, boolean maximizable, boolean iconifiable)`

- **ATTENTION**: invisible et de taille 0,0 par défaut!
- possède un root pane, donc on a accès à une barre de menu, un content pane et un glass pane



# La JInternalFrame



- toutes les fenêtres sont par défaut en haut à gauche
- pas de gestion des fenêtres en cascade, en grille, etc :(



# Les menus

---

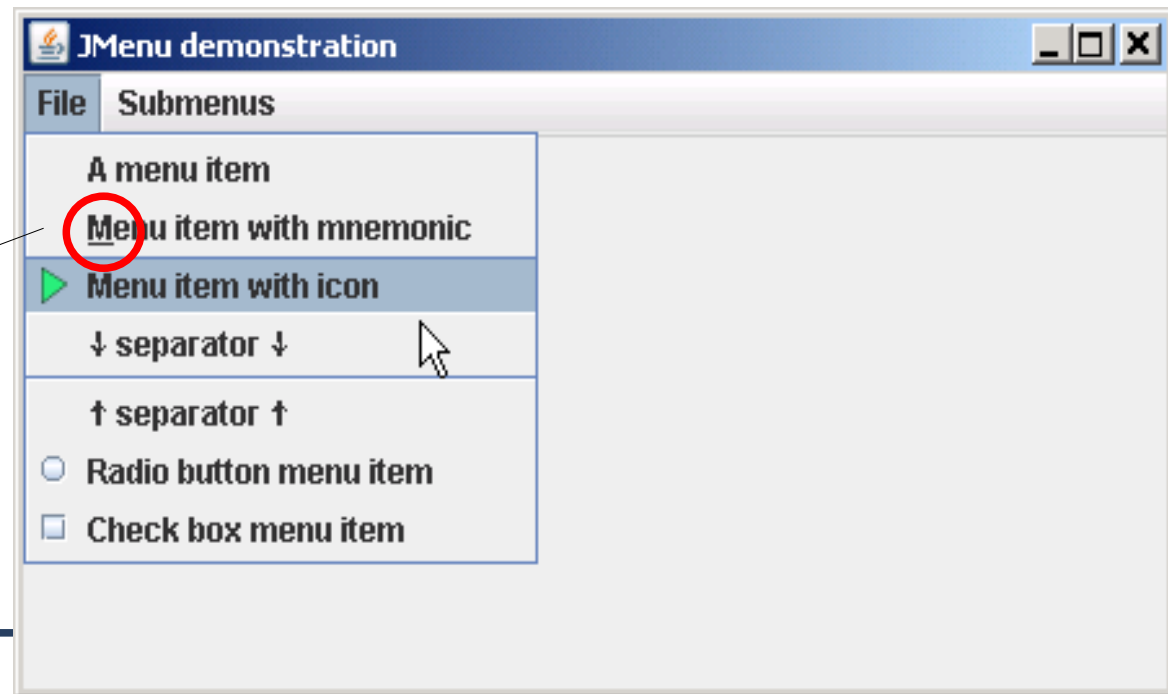
- on ajoute une barre de menu à une **JFrame/JInternalFrame** avec:
  - **setJMenuBar (JMenuBar bar)**
- une **JMenuBar** contient des **JMenu** qui peuvent contenir:
  - des boutons spéciaux: **JMenuItem**,  
**JRadioButtonMenuItem**,  
**JCheckBoxMenuItem**
  - des séparateurs: **addSeparator ()**
  - des sous-menus de type **JMenu**



# Les menus

- 3 modes de sélection:
  - clic de souris
  - validation clavier avec Entrée
  - mnémonique (caractère souligné)

item sélectionnable  
en pressant 'M'

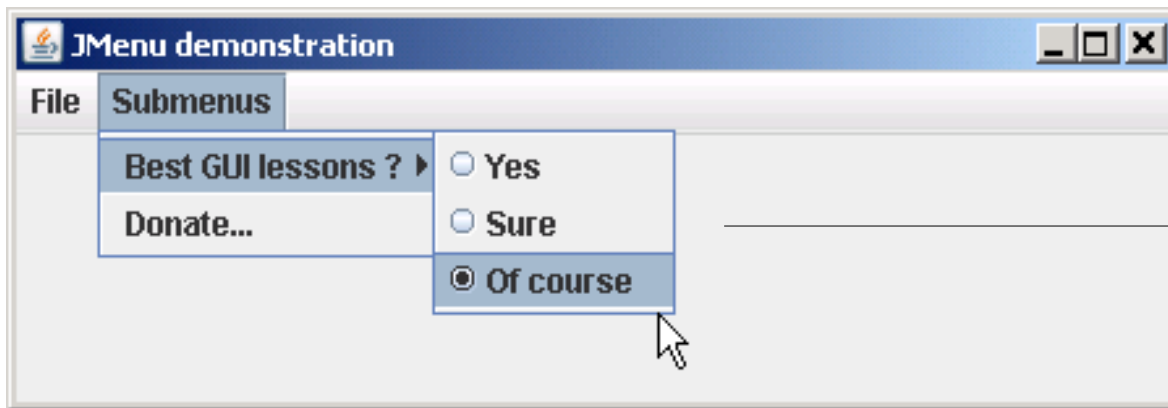






# Les menus

- exemple de sous-menu:



on utilise un `ButtonGroup` comme d'habitude

- on réagit aux sélections d'items avec des `ActionListener`, comme pour les boutons normaux



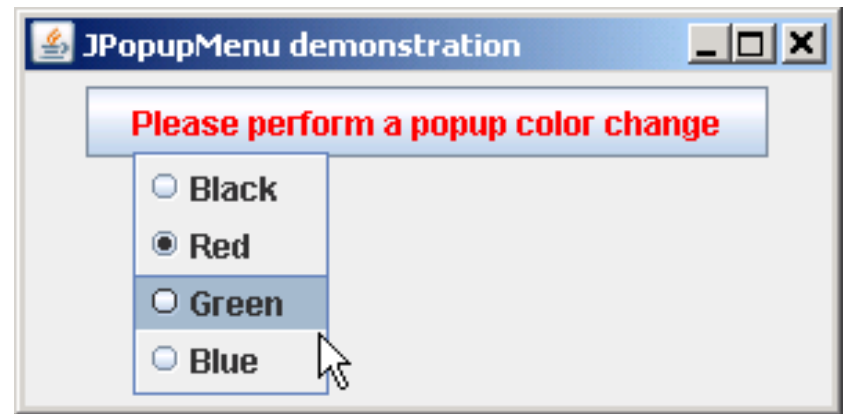
# Les menus contextuels

---

- tout composant peut recevoir un menu contextuel avec:

`setComponentPopupMenu (JPopupMenu menu)`

- fonctionne comme un **JMenu**
- apparaît quand on fait un clic droit sur le composant concerné





# Les actions

---

- une action définit:
  - un nom
  - un raccourci clavier (*accelerator*)
  - un mnémonique
  - une icône
  - une description courte
  - du code à exécuter
  - un état activé/désactivé
- permet d'éviter des duplications de code



# Les actions

- exemple: un bouton "Run"

```
final Action buttonAction=new AbstractAction("Run",
    new ImageIcon(
        fr.umlv.ig.example_viewer.GUIExampleViewer.class.getResource("run.png"))) {
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(f,"OK, I run.");
    }
};
/* KeyEvent.VK_R matches both 'r' and 'R' */
buttonAction.putValue(Action.ACCELERATOR_KEY,
    KeyStroke.getKeyStroke(KeyEvent.VK_R,Event.CTRL_MASK));
/* We need to use the following because 'U' would be
 * autoboxed into a Character and not an Integer */
buttonAction.putValue(Action.MNEMONIC_KEY,Integer.valueOf('U'));
buttonAction.putValue(Action.SHORT_DESCRIPTION,"Shows a message");
menu.add(new JMenuItem(buttonAction));
```

l'action est passée au constructeur du bouton

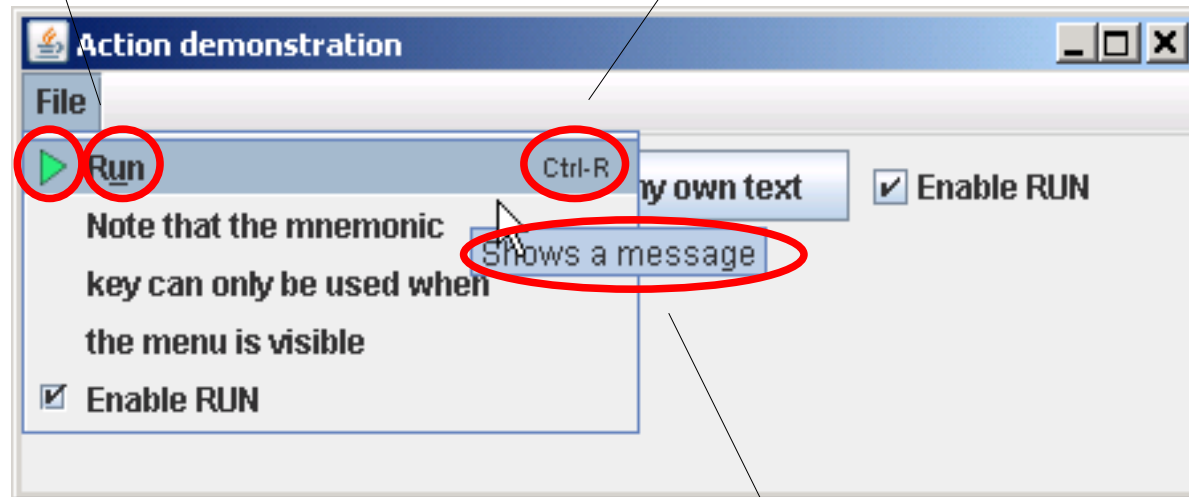


# Les actions

l'accélérateur

le nom avec le mnémonique

l'icône



la bulle d'aide



# Les actions

- on peut désactiver le texte de l'action:

```
JButton b=new JButton(buttonAction);  
b.setHideActionText(true);  
/* NOTE: setText must occur AFTER setHideActionText */  
b.setText("The same with my own text");
```

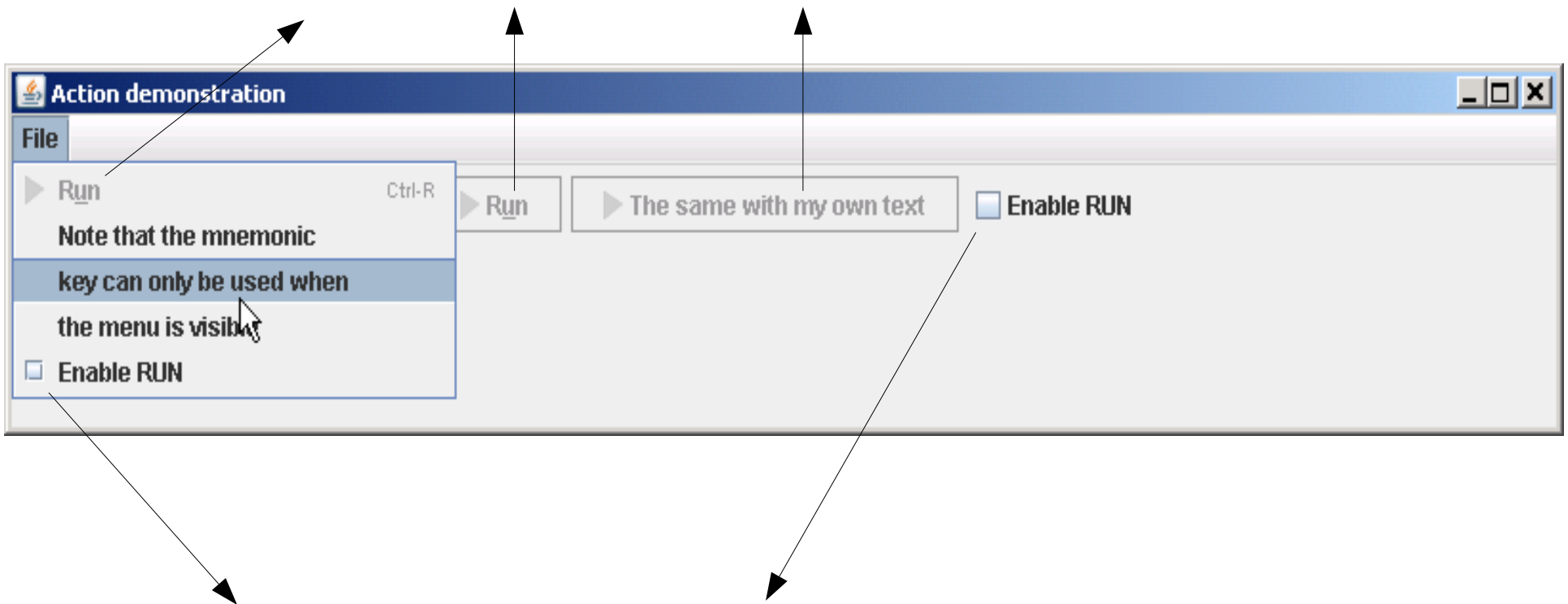
- on peut aussi gérer la sélection pour les boutons qui le supportent:

```
final Action enableAction=new AbstractAction("Enable RUN") {  
    @Override public void actionPerformed(ActionEvent e) {  
        /* The method isSelected does not exist in Action */  
        buttonAction.setEnabled((Boolean)getValue(Action.SELECTED_KEY));  
    }  
};  
/* You can replace 'true' by any non null value:  
* - null means: selection not taken into account  
* - null means: getValue(Action.SELECTED_KEY) will return a Boolean  
* However, if you use 'true', you will set the state as selected */  
enableAction.putValue(Action.SELECTED_KEY,true);  
menu.add(new JCheckBoxMenuItem(enableAction));
```



# Les actions

tous les boutons "Run" ont le même comportement

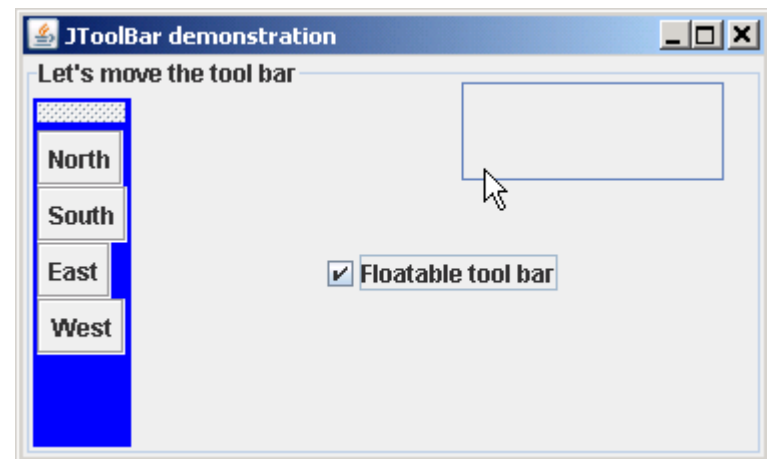
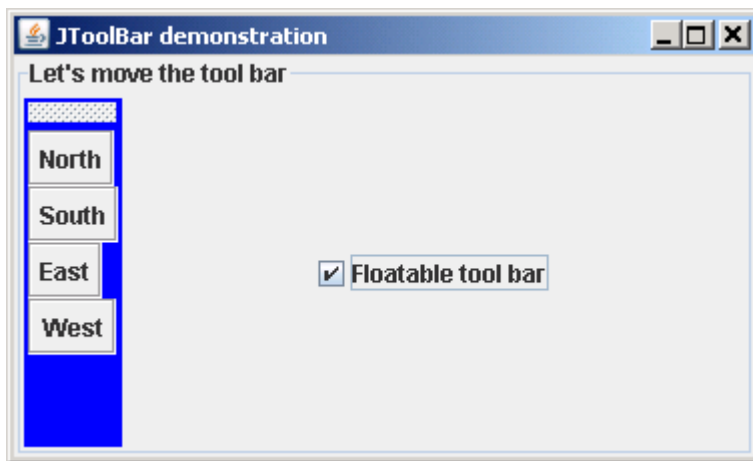


gestion centralisée du bouton "Run"



# La barre d'outils

- une **JToolBar** peut être placée sur les bords (et pas au centre!) d'un container muni d'un **BorderLayout**
- ne rien mettre d'autre qu'un composant au centre et la **JToolBar**







# La barre d'outils

- utilise un **BoxLayout**
- peut se détacher si elle est *floatable*
- quand on ferme une **JToolBar** flottante:
  - elle retourne à sa dernière position si est *floatable*
  - elle disparaît sinon

